

# A Multi-Objective Linear Program for Nutrition

From CU Denver Optimization Student Wiki

A Multi-Objective Linear Programs is a linear program with more than one objective function. This linear program solves how to minimize cost (based on the dollar value of meals) and maximize profit (based on the grams of protein) while meeting specific requirements for both the consumer and supplier. This multi-objective linear program will ultimately be developed by combining the separate objectives.

## Contents

- 1 Dictionary of Variables
- 2 Meal Codes
- 3 Objective Function 1: Minimize Cost
- 4 Objective Function 2: Maximize Profit
- 5 Multi-Objective Function: The Difference of Minimizing Cost and Maximizing Profit

# Dictionary of Variables

$x_j$	<i>number of meal type <math>j</math> purchased in a week</i>
$\beta_j$	<i>minimum number of meal type <math>j</math> purchased in a week</i>
$\mu_j$	<i>maximum number of meal type <math>j</math> purchased in a week</i>
$c_j$	<i>cost of meal type <math>j</math></i>
$a_{ij}$	<i>amount of nutrient type <math>i</math> in meal <math>j</math></i>
$\zeta_i$	<i>minimum number of nutrient type <math>i</math> needed in a week</i>
$\eta_i$	<i>maximum number of nutrient type <math>i</math> needed in a week</i>
$p_j$	<i>amount of protein in meal <math>j</math></i>
$\gamma$	<i>the maximum number of meals made in a week</i>
$\lambda_1 \in (0, 1]$	<i>scalar weight of minimize cost objective</i>
$\lambda_2 \in (0, 1]$	<i>scalar weight of maximize profit objective</i>

# Meal Codes

<i>SP</i>	<i>Shrimp Paella</i>
<i>SDN</i>	<i>Spicy Dan Noodles</i>
<i>TC</i>	<i>Turkey Chili with Beans</i>
<i>PAS</i>	<i>Supreme Pasta</i>
<i>OEB</i>	<i>Over Easy Burger</i>
<i>SCS</i>	<i>Sweet Chili Glazed Salmon</i>
<i>TMC</i>	<i>Tex Mex Chicken Bowl</i>
<i>OAT</i>	<i>ABJ Oatmeal Bowl</i>
<i>BP</i>	<i>B. Platter</i>
<i>BSC</i>	<i>Buf falo Style Chicken Bowl</i>
<i>CT</i>	<i>Chicken Teriyaki with Rice</i>
<i>TML</i>	<i>Turkey Meat Loaf</i>

## Objective Function 1: Minimize Cost

The nutritional data to minimize the cost of a weeks worth of meals was designed for a 45 year old, 6ft male, with a starting weight of 175 pounds. With a Basal Metabolic Rage of 1765 calories per day and expected expenditure of 1500 calorie from exercise, it is prescribed that the consumer has a minimum weekly nutritional intake ( $\zeta_i$ ) of the following: 12355 calories, 1239 grams of protein, 1393 grams of carbohydrates, and 203 grams of fat. To promote variety in the diet, each meal must be consumed at least once and may not be consumed more than 5 times (i.e.  $1 = \beta_j \leq x_j \leq \mu_j = 5$ ). Based on the cost and nutritional information of each of the meals, the objective function can be written as:

$$\begin{aligned}
 \min \quad & \sum_{j=1}^m c_j \cdot x_j \\
 \text{s.t.} \quad & \zeta_i \leq \min \sum_{j=1}^m a_{ij} \cdot x_j \leq \eta_i \\
 & i = 1, 2, \dots, n \\
 & j = 1, 2, \dots, m
 \end{aligned}$$

### Interpretation of Results

The optimal solution is \$354, which can be interpreted as the minimum cost to buy a weeks worth of food while meeting all nutritional requirements and sufficiently varying the meals. The meals that were least expensive but had lots of nutritional benefit were purchased were purchased in higher quantities. The following diet is optimal number of each meal that should be purchased:

<i>BP</i>	1
<i>BSC</i>	5
<i>CT</i>	5
<i>OAT</i>	5
<i>OEB</i>	1
<i>PAS</i>	5
<i>SCS</i>	3
<i>SDN</i>	5
<i>SP</i>	1
<i>TC</i>	4
<i>TMC</i>	1
<i>TML</i>	5

It is important to note that B. Platter (BP), Over-Easy Burger (OEB), Shrimp Paella (SP), and Tex-Mex Chicken Bowl (TMC) have the least nutritional benefits for this diet per dollar.



## Objective Function 2: Maximize Profit

The definition of profit in this model is to sell as much protein as possible. Each meal may only be made 5 (i.e.  $0 = \beta_j \leq x_j \leq \mu_j = 5$ ) times to promote variety, and only 50 meals may be made in a given week (i.e.  $\gamma = 50$ ) . The objective function for this model can be written as:

$$\begin{aligned} \max \quad & \sum_{j=1}^m p_j \cdot x_j \\ \text{s.t.} \quad & \sum_{j=1}^m x_j \leq \gamma \\ & j = 1, 2, \dots m \end{aligned}$$

### Interpretation of Results

The optimal solution is 1325 grams of protein. The results of this objective function are intuitive -- make 5 of the meal with the most protein and then move on the to the meal with the next most protein. The following is the optimal number of each meal that should be purchased:

<i>BP</i>	5
<i>BSC</i>	5
<i>CT</i>	0
<i>OAT</i>	5
<i>OEB</i>	5
<i>PAS</i>	5
<i>SCS</i>	5
<i>SDN</i>	0
<i>SP</i>	5
<i>TC</i>	5
<i>TMC</i>	5
<i>TML</i>	5

It is important to note that Spicy Dan Noodles (SDN) and Chicken Teriyaki (CT) have the least amount of protein, so neither were made. It is preferred to make 5 of each of the other 10 meals.

# Multi-Objective Function: The Difference of Minimizing Cost and Maximizing Profit

Combining the two previous objective functions to create the multi-objective linear program entails using the same data as from the single objective functions and writing one minimization objective function. Since previously the objective was to maximize profit, I can achieve this by subtracting the maximization of the profit from the minimization of cost. Utilizing  $\lambda_1$  and  $\lambda_2$  to weight each of the objective functions, the multi-objective function can therefore be written as:

$$\min \quad (\lambda_1 \cdot \sum_{j=1}^m c_j - \lambda_2 \cdot \sum_{j=1}^m p_j) \cdot x_j$$

$$s.t. \quad \zeta_i \leq \min \sum_{j=1}^m a_{ij} \cdot x_j \leq \eta_i$$

$$\sum_{j=1}^m x_j \leq \gamma$$

$$i = 1, 2, \dots n$$

$$j = 1, 2, \dots m$$

## Interpretation of Results

The interpretation of the multi-objective optimal solution is difficult since units of the minimization of cost objective function (dollars) is different than units of the maximization of profits objective function (grams of protein). The value of the objective function is of little significance. It is more important to examine how many of each meal satisfies the multi-objective function. Below are 3 cases where each separate objective function is weighted:

Case 1 ( $\lambda_1 = \lambda_2 = 1$ )

The optimal solution is -868.06. When each part of the mutli-objective function receives equal weight, maximizing grams of protein is greater than minimizing dollars, which results in a negative optimal solution. The following is the optimal number of each meal that should be purchased:

<i>BP</i>	5
<i>BSC</i>	5
<i>CT</i>	1
<i>OAT</i>	5
<i>OEB</i>	5
<i>PAS</i>	5
<i>SCS</i>	1
<i>SDN</i>	5
<i>SP</i>	3
<i>TC</i>	5
<i>TMC</i>	5
<i>TML</i>	5

Case 2 ( $\lambda_1 = 1, \lambda_2 = .01$ )

The optimal solution is 344.68. When significantly more weight is given to the minimization of cost objective function, the resulting optimal solution becomes positive. Again, it is important to examine how many of each meal should be purchased to achieve the optimal solution:

<i>BP</i>	1
<i>BSC</i>	5
<i>CT</i>	5
<i>OAT</i>	5
<i>OEB</i>	1
<i>PAS</i>	5
<i>SCS</i>	3
<i>SDN</i>	5
<i>SP</i>	1
<i>TC</i>	4
<i>TMC</i>	1
<i>TML</i>	5

Case 3 ( $\lambda_1 = .5, \lambda_2 = .01$ )

The optimal solution is 167.72. Even though less weight was given to the minimization of cost objective function compared to Case 2, the optimal solution decreased but

remained positive. It is interesting to note that although the optimal solution was lower in Case 3 than Case 2, the number of each meal that should be purchase to achieve the optimal solution is the same as in Case 2:

<i>BP</i>	1
<i>BSC</i>	5
<i>CT</i>	5
<i>OAT</i>	5
<i>OEB</i>	1
<i>PAS</i>	5
<i>SCS</i>	3
<i>SDN</i>	5
<i>SP</i>	1
<i>TC</i>	4
<i>TMC</i>	1
<i>TML</i>	5

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=A\\_Multi-Objective\\_Linear\\_Program\\_for\\_Nutrition&oldid=789](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=A_Multi-Objective_Linear_Program_for_Nutrition&oldid=789)"

- This page was last modified on 5 December 2017, at 23:05.
- This page has been accessed 10,846 times.

# A Wheelie Good Time: Safe Biking in Denver

From CU Denver Optimization Student Wiki

Hello and welcome to Angela Morrison and Weston Grewe's project page for A Wheelie Good Time: Safe Biking in Denver. Please read about our project below. You can find relevant code, slides, and more graphics in our GitHub repository ([https://github.com/DillWithIt77/D2P\\_Spring\\_2022](https://github.com/DillWithIt77/D2P_Spring_2022)). Also, you can click on our names to find out more about us and other projects we have worked on.

## Contents

- 1 Abstract
- 2 Project
- 3 Data Cleaning
- 4 Algorithms
- 5 Results
  - 5.1 Frequency
  - 5.2 Feasibility
  - 5.3 Burden
- 6 Policy Recommendation
  - 6.1 A Note on Southwest and Northeast Denver
- 7 References

## Abstract

Denver has always been a car-centric city, but with the rise of global warming more environmentally-friendly forms of transportation are required. However, Denver streets are not safe enough for most to feel comfortable biking; 60% of Denver residents report being “interested, but concerned” about commuting by bike, only 4% feel “highly confident.” Thus, there is ample room for growth if streets were safer. In 2016, The Department of Transportation and Infrastructure (DOTI) of Denver introduced the Denver Moves: Bicycles Program, which is dedicated to building 125 miles of bike lanes by 2024. With 74 miles already completed, a natural question is “Will this be enough to move Denver towards eco-friendly transportation?” The COVID-19 pandemic has reshaped some parts of the city to accommodate outdoor dining and activities, proving that Denver is ready and willing to take back some roads from motor vehicles.

It is not sufficient to just build more lanes; Denver should prioritize building lanes that can benefit the most people and extend or connect existing routes. We analyze the shortest paths bikers may safely take (restricting the distance allowed on shared roads and unprotected bike lanes) to key points in Denver (Auraria Campus, City Park, DU, Five Points, 16th St Mall) to understand where (protected) bike lanes can significantly improve a rider’s commute. After accessing these results, we then suggest several locations that the DOTI should either build lanes or upgrade current lanes to protected bike lanes to incentivize eco-friendly commuting while also decreasing travel times between key locations. This is done to ensure the DOTI are getting the most bikes for their lanes from the Denver Moves project.

## Project

The Department of Transportation and Infrastructure (DOTI) for the City of Denver is invested in encouraging more eco-friendly modes of transportation, i.e. bikes, e-bikes, scooters, busses, and light rail. In 2016, the DOTI<sup>[1]</sup> found that 60% of Denver residents are interested but concerned in commuting by bike, compared to the only 4% who feel highly confident. It is not just residents' presumption that biking is dangerous. The data bears it out and in particular, 45% of bicycle crashes in Denver<sup>[2]</sup> occurred from a car veering into a rider in a lane. More protected bike lanes would save lives. The Denver Moves: Bicycle Program was born. This initiative is dedicated to building 125 miles of bike lanes in Denver by 2024. Roughly 74 miles have already been completed.

Distance is not the only factor when it comes to commuting. Road and path type are also important. Most would prefer a 5 mile commute primarily on the Cherry Creek Trail to a 2 mile commute through busy roads. There is a hierarchy of roads that cyclists prefer to ride on, and it is not surprising the least to most preferred types are: shared roads, unprotected bike lanes, protected bike lanes, neighborhood bikeways, and paved shared use paths. As a clarification, shared roads are roads that cyclists share with cars, shared use paths are paths that cyclists share with pedestrians, and neighborhood bikeways are paths shared with pedestrians that occasionally cross neighborhood streets.

One way to encourage more residents to commute to work is to develop a bike infrastructure that allows commuters to stay primarily on road types they are comfortable with (protected bike lanes and shared use paths). We analyze shortest paths and compare them to paths that restrict the amount of time that is allowed to be spent on less preferred road types (unprotected bike lanes and shared roadways). For this project, we look at 5 destinations in Denver: Auraria Campus (specifically Student Commons Building), Union Station, Five Points (specifically Denver Central Market), City Park, and University of Denver. These locations were chosen since many people work in these areas and all five have limited parking, making they are ideal places to bike to instead of drive.

Simply building more bike lanes is not the only way to encourage people to ride. This year (2022) Colorado has introduced two major policies to encourage bicycle commuting. The first is making the Idaho Stop (Safety Stop) legal for cyclists<sup>[3]</sup>. That is, cyclists may treat stop signs as yield signs and red lights as stop signs. This is a counterintuitive policy, first introduced in Idaho (hence the name), that greatly reduces bicycle accidents, especially those caused from drivers turning into cyclists at a light. The second policy is \$400 grant (\$1200 for low-income) for residents to purchase an e-bike<sup>[4]</sup>. The e-bike has been a major development in making cycling accessible, their ability to hold speed can make a 5 mile ride along a path essentially painless. With these developments, Denver is entering a critical period when it comes to bicycle commuting. The city needs to be sure it has adequate infrastructure.

## Data Cleaning

We began with a collection of LineStrings and MultiLineStrings representing bikeable roads in Denver. Each record also contained the type of road and its name. In order to transform this file to make it compatible for the work that needed to be done in R, the MultiLineStrings needed to be broken up into individual LineStrings. This was done in python using the explosion function for GeoPandas DataFrames<sup>[5]</sup>. With this done, we could move onto the cleaning that needed to be done in R.

To transform this into a network that we can perform our analysis on, we used R and the sfNetworks package<sup>[6]</sup>. The package sfNetworks has functions that easily clean up a network. The first thing we did was round the lat/long coordinates of our data to five decimal points. This resolved an issue of our network being very disconnected. For example, we had edges that look like  $(A, B)$  and  $(B', C)$  where points  $B$  and  $B'$  are across the street from each other.  $B$  and  $B'$  should really be the same point, otherwise our network is disconnected at that point, rounding the lat/long solves this problem.

Our network also lacked intersections of paths, using sfNetworks subdivision tool we were able to account for this. After this, we took the largest connected component as nodes in the other connected components cannot reach the points in our network. The subdivision feature is not perfect and introduced some redundant nodes. The last bit of cleaning removed these nodes. With our network cleaned up a bit, we could calculate the information needed to ready out data for the algorithms.

With R having cleaned up the network, the last bit of information needed was the distance of each edge in the network. This was done by importing the network back into Python and computing the distance in meters for each edge. Finally, we exported the edges with their distances and the nodes as a CSV file and performed our analysis in Python. Check out our data cleaning here ([https://github.com/DillWithIt77/D2P\\_Spring\\_2022/tree/main/Data%20Cleaning%20and%20Plotting!](https://github.com/DillWithIt77/D2P_Spring_2022/tree/main/Data%20Cleaning%20and%20Plotting!)).

## Algorithms

We analyzed our network using two algorithms and compared the results. The first algorithm we used was the shortest path algorithm. We computed shortest paths from all nodes in our network to our 5 desired locations detailed above. For the implementation, we used label-correcting to find the shortest paths. More detail can be found in our GitHub repository ([https://github.com/DillWithIt77/D2P\\_Spring\\_2022](https://github.com/DillWithIt77/D2P_Spring_2022)).

The second algorithm we used is a cost-constrained shortest path algorithm, a detailed explanation can be found in Chapter 3 of Network Flows by Ahuja, Magnanti, and Orlin<sup>[7]</sup>. Essentially, given a cost  $\tau$  we compute the shortest path from a source node to the target node that does not exceed  $\tau$ . For this project, we interpret  $\tau$  as a safety factor and find paths that do not exceed this safety factor. To initialize safety factors on each edge  $(i, j)$ , we multiply the length of the edge (in miles) to a predetermined number<sup>[8]</sup> indicating risk and assign that value as  $\tau_{ij}$  for that edge. The algorithm is essentially a dynamic program that computes runs in  $O(mn\tau)$ . Since our chosen values of  $\tau$  are small (10 and 20), the program still runs quickly, about 5 seconds. Again, more detail can be found in ([https://github.com/DillWithIt77/D2P\\_Spring\\_2022](https://github.com/DillWithIt77/D2P_Spring_2022)).

# Results

Our results can be broken down into 3 categories: frequency (how often paths are used), feasibility (where can we ride in Denver), and burden (how much harder is it to ride safely).

## Frequency

Using only the results from the shortest paths, we can discover which roads are most frequently used. Doing so, we found that 8 of the 20 most used edges in the network are shared roads or unprotected bike lanes. By frequency alone, these 8 edges should be upgraded (if possible) to a protected bike lane. The graphics below are a heat map (left) of the most used roads and a graphic of the type of path (right) for each edge in the network. In the heat map, the brighter the color (the more red it is) the more frequently that edge is used in the shortest paths computed for each desired location. For the type of path, blue are for shared use paths, green for neighborhood byways, yellow for shared roadways, orange for unprotected bike lanes, and finally pink for protected bike lanes.

Frequency of Edges in Regular Shortest Paths



Heatmap of Denver Bike Routes. Dark blue is least used, red is most used.

Types of Edges in Regular Shortest Paths



Map of Denver Bike Routes. Blue-Shared Path, Green-Neighborhood Byway, Light Orange-Shared Roadway, Dark Orange-Unprotected Bike Lane, Pink-Protected Bike Lane

While there are 8 edges of the top 20 that are shared roadways or unprotected bike lanes, these do not necessarily correspond to 8 individual roads. Due to the cleaning methods in the network, there are some streets that were subdivided, cause some repeats in the street names. This means that there are only about 4 actual roads that could be upgraded. they are listed in the table below.

Street	From	To
Emerson	1 <sup>st</sup> Ave	Louisiana
Louisiana	Logan	Franklin
Franklin	Buchtel	Dartmouth
Iliff	University	Humboldt

## Feasibility

The following figures are visualizations of points that can actually reach one of our desired locations. The cautious rider (left) will only spend as much as 2 miles on a shared road or 3 miles on an unprotected bike lane. The more adventurous cyclist (right) doubles these values. We see that the cautious cyclist cannot access much of Denver. Even the adventurous cyclist cannot access all of Denver. In the graphic for the adventurous cyclist, we see a "dead zone" in northeast Denver, this represents the Park Hill and Central Park neighborhoods. We recommend building more paths in these neighborhoods to boost accessibility.

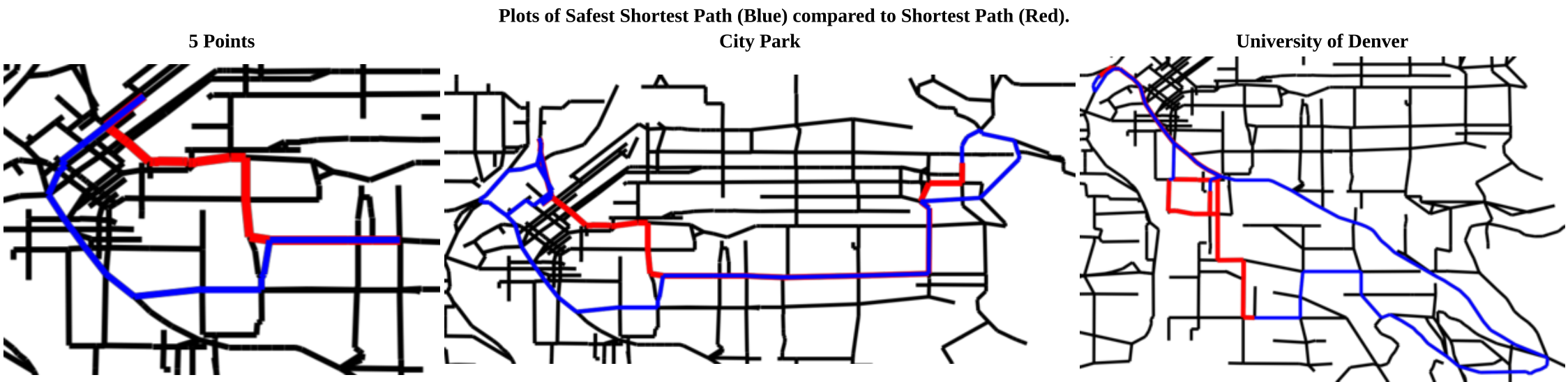




In both maps, we can see that feasible nodes are clustered around Denver's Cherry Creek Trail, Highline Canal Trail, and Platte River Trail. These trails are major infrastructure that make biking in Denver a joy. However, those who live far away from one of these trails cannot safely ride in most of Denver. We recommend building more protected lanes that can connect cyclists to these trails. Moreover, there is an inequity here. Many of Denver's richest neighborhoods (Washington Park, LoHi, RiNo, LoDo, Cherry Creek, etc) are adjacent to one of these paths. Building more protected bike lanes would make biking safer for marginalized and less-privileged communities.

## Burden

Lack of safe biking infrastructure can cause cyclists to take longer routes than might be necessary. A casual cyclist riding to work may only ride at a 7mph pace as they are likely not riding a road bike and not trying to work up a sweat. With this in mind, adding an extra 1 or 2 miles to a commute can add 10 to 20 minutes of riding, a significant burden on the rider. The plots below give a visualization of a path an adventurous cyclist would take within their level of safety (the blue route), the red lines indicate where the shortest path would divert them. In the most extreme example, we see that when riding from Sloan's Lake to University of Denver, both the safe route and shortest route take the rider along the Cherry Creek Trail until Washington Park. The shortest route then takes the rider off the Cherry Creek Trail following Emerson to Buchtel to DU for an extra 3.5 miles of riding. The safe route on the other hand brings the rider another 6 miles along the Cherry Creek trail to the Highline Canal Trail. The rider then rides the Highline Canal trail for more than 6 miles to arrive at DU. All in all, maintaining safety adds around 10 miles to the ride. At least the rider is safe.



Another area of concern is downtown Denver. In the plot for 5 Points we see that riders are diverted around downtown instead of through downtown. If downtown had safer roads, riders could easily ride through downtown. Thus, we recommend building a 2 protected bike lanes through downtown, one north/south, one east/west. We do not have a recommendation on the particular street, our heatmap indicates some good choices, but it may be in the best interest of the city to choose a street that will have a minimal impact on traffic.

## Policy Recommendation

We found that bike infrastructure in Denver is insufficient. Many neighborhoods in the city lack access to safe routes to key points in the city. However, Denver does have an outstanding network of routes consisting of shared paths and neighborhood trails. It should not be understated how important these routes are for bicycle commuting. Residents who live near these paths will find commuting along these paths safe and enjoyable. We recommend building more routes that connect to these paths to expand a safe network of riding for more residents. Our analysis illustrated that a protected bike lane connecting Washington Park to Buchtel Blvd would be beneficial. From an understanding of the geography of the University of Denver, it may be best to have this route extend to Evans Ave. This route could be built on Downing or Franklin, but Franklin is the better candidate. South High School is on Franklin; a protected bike line would provide these high school students with a safer route.

Finally, we saw that City Park/Park Hill/Central Park all need better access. City Park itself is safe for riders, however, we recommend building a protected bike lane from City Park to downtown to make cycling downtown safer in its own regard. Additionally, we recommend having a safe route from City Park to 5 Points. The route from City Park to 5 Points does more than just make biking safer. Many young people live in City Park and drink in 5 Points. The addition of this route may cut down on drunk driving if more people who are drinking are biking instead of driving. (We recognize that

drinking and biking is still illegal, but we cannot deny that intoxicated bikers pose much less risk to residents than intoxicated drivers). As for particular routes to build, Michael Schmidt's, Evan Shapiro's, and Em Gibbs's project: Finding Optimal Shared Streets in Denver from this year's D2P has some suggestions.

## A Note on Southwest and Northeast Denver

We did not have strong recommendations on what to do in Southwest and Northeast Denver. This is an artifact of the desired locations we chose. As Auraria, Union Station, and 5 Points are all clustered around North Central Denver many of our routes are biased in showing that routes in these areas are more important. However, in both Southwest and Northeast Denver, we can see that we likely need more protected bike lanes there are not many of them. More analysis would be needed to determine if this is actually the case.

## References

1. ↑ Denver Moves: Bicycles Program. City and County of Denver. (n.d.). Retrieved April 11, 2022, from [1] (<https://www.denvergov.org/Government/Agencies-Departments-Offices/Agencies-Departments-Offices-Directory/Department-of-Transportation-and-Infrastructure/Programs-Services/Bicycles>).
2. ↑ Denver Public Works Transportation and Mobility (2016) Bicycle Crash Analysis: Understanding and Reducing Bicycle and Motor Vehicle Crashes. [2] ([https://www.denvergov.org/content/dam/denvergov/Portals/705/documents/visionzero/denver-bicycle-motor-vehicle-crash-analysis\\_2016.pdf](https://www.denvergov.org/content/dam/denvergov/Portals/705/documents/visionzero/denver-bicycle-motor-vehicle-crash-analysis_2016.pdf))
3. ↑ McKenzie, M. (2022, May 2). Statewide safety stop legislation expects to create 'bicycle-friendly' communities, including Greeley. Greeley Tribune. Retrieved May 2, 2022, from [3] (<https://www.greeleytribune.com/2022/05/02/statewide-safety-stop-legislation-expects-to-create-bicycle-friendly-communities-including-greeley/>).
4. ↑ Brasch, Sam. “Trying to Electrify Your Home, Add Solar Panels or Buy an e-Bike? Denver Wants to Help Pay for It.” Denverite, Denverite, 15 Apr. 2022,[4] (<https://denverite.com/2022/04/14/trying-to-electrify-your-home-add-solar-panels-or-buy-an-e-bike-denver-wants-to-help-pay-for-it/>).
5. ↑ Jordahl, K. (2014). GeoPandas: Python tools for geographic data. [5] (<https://github.com/Geopandas/Geopandas>).
6. ↑ Lucas van der Meer, Lorena Abad, Andrea Gilardi and Robin Lovelace (2022). sfnetworks: Tidy Geospatial Networks. R package version 0.5.5. [6] (<https://CRAN.R-project.org/package=sfnetworks>).
7. ↑ Ahuja, Ravindra K., Thomas L. Magnanti, and James B. Orlin. "Network flows." (1988).
8. ↑ Teschke, Kay, et al. "Bicycling: Health risk or benefit." UBC Med J 3.2 (2012): 6-11.  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=A\\_Wheelie\\_Good\\_Time:\\_Safe\\_Biking\\_in\\_Denver&oldid=3823](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=A_Wheelie_Good_Time:_Safe_Biking_in_Denver&oldid=3823)"

- This page was last modified on 3 May 2022, at 13:47.
- This page has been accessed 1,178 times.

# Abigail Nix

From CU Denver Optimization Student Wiki

## Contents

- 1 About me
- 2 Education
- 3 Projects
- 4 GitHub

## About me

Hi! My name is Abigail Nix and I am a second year PhD student at CU Denver. I am interested in discrete mathematics (and in particular graph theory) and optimization.

## Education

I received my Bachelor's in Mathematics at Middlebury College in May 2023.

## Projects

In the spring of 2024, I worked on the project In-N-Out Of Kilter as part of the course on Combinatorial Optimization.

For the Applied Graph Theory course in Spring 2025, I worked with Ari Holcombe Pomerance on a project on Graph Coloring Variants.

## GitHub

Github (<https://github.com/abigail-nix>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Abigail\\_Nix&oldid=4846](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Abigail_Nix&oldid=4846)"

Category: Contributors



- This page was last modified on 7 April 2025, at 16:48.
- This page has been accessed 127 times.



# Acadia Larsen

From CU Denver Optimization Student Wiki

Acadia Larsen is currently a non-degree student at University of Colorado: Denver. Prior to this, he worked as a Denver Math Fellows at the Denver Institute for International Studies at Montebello where he taught and tutored 9th grade math in small groups. Acadia earned his BA in mathematics at Whittier College in California where he was awarded several fellowships for his work in math. When not working or in class, you can find him running, hiking, or bouldering.

Contributions: Convex Hull Finding Algorithms

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Acadia\\_Larsen&oldid=400](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Acadia_Larsen&oldid=400)"

Category: Contributors

- 
- This page was last modified on 25 April 2017, at 18:34.
  - This page has been accessed 2,281 times.

# Alana Saragosa

From CU Denver Optimization Student Wiki

## About Me

Hello there! My name is Alana Saragosa, and I am a first-year master's student at CU Denver. A few of my favorite things include watching all the new movies, eating at new restaurants, and hanging out with my family.

## Education

Metropolitan State University of Denver: B.S. in Mathematics

## Projects

Fall 2023: I worked with Paul Guidas and Colin Furey on Emissions and Equality: Colorado Car Share Optimization a facility locations optimization project. Where the goal was to bring cheap car access, via federally funded Colorado CarShare Non-profit, to targeted groups in Denver.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Alana\\_Saragosa&oldid=4432](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Alana_Saragosa&oldid=4432)"

Category: Contributors

- 
- This page was last modified on 7 November 2023, at 13:54.
  - This page has been accessed 52 times.

# Alex Semyonov

From CU Denver Optimization Student Wiki

## About Me

Hello, my name is Alex Semyonov and I am currently pursuing a PhD in applied mathematics here at CU Denver. I received my B.S. in mathematics (with a minor in neuroscience) from CU Denver in 2021. Currently, I am interested in probabilistic/statistical modeling as well as graph theory. Outside of mathematics, I enjoy freerunning (this is similar to parkour but focuses on self-expression through movement as opposed to the efficiency of movement), mountaineering, and gaming.

## Project

Clustering Neighborhoods in Order to Analyze Policy Needs

([http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Clustering\\_Neighborhoods\\_in\\_Order\\_to\\_Analyze\\_Policy\\_Needs](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Clustering_Neighborhoods_in_Order_to_Analyze_Policy_Needs))

I am currently working on a project (alongside Jacob Dunham and Orlando Gonzalez) to try and identify the best neighborhood in Denver subject to various personal constraints (finances, crime tolerance, etc.).

Graph Theory Project: [1] ([https://math.ucdenver.edu/~sborgwardt/wiki/index.php/Counting\\_Eulerian\\_Cycles\\_in\\_Graphs](https://math.ucdenver.edu/~sborgwardt/wiki/index.php/Counting_Eulerian_Cycles_in_Graphs))

## Contact Info

Please contact me by email: [alexander.semyonov@ucdenver.edu](mailto:alexander.semyonov@ucdenver.edu)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Alex\\_Semyonov&oldid=4930](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Alex_Semyonov&oldid=4930)"

Category: Contributors

- 
- This page was last modified on 21 April 2025, at 09:32.
  - This page has been accessed 80 times.

# Alexa desautels

From CU Denver Optimization Student Wiki

I am currently an applied math master's student at CU Denver. I got my bachelor's degree in math at Penn State University before moving to Colorado in 2016. I hope to use my education to work in industry. In addition to being a student, I also work at Stack Subs and teach after school coding classes at a few elementary schools. I love to apply math to my every day life, so for my final project I applied column generation to cutting loaves of bread at work.

Follow this link ([http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Applications\\_of\\_Column\\_Generation](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Applications_of_Column_Generation)) to my application.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Alexa\\_desautels&oldid=967](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Alexa_desautels&oldid=967)"

Category: Contributors

- 
- This page was last modified on 7 December 2017, at 13:02.
  - This page has been accessed 1,873 times.



# Alfred P Wahabby

From CU Denver Optimization Student Wiki

Alfred P. Wahabby is a graduate student at the University of Colorado, Department of Mathematical and Statistical Sciences. My background is physics, engineering, philosophy and English literature. My passion for mathematics was triggered when I was undergraduate in physics; the main topic in physics that intensified my desire to commit to the math program was a general relativity summer course. I am a math teacher and I love my job. Ultimately I want to put to use the math I learn in the field of physics although I may continue to be a teacher for life.

Brief History Of Linear Programming

The amazing Fourier, progenitor of linear systems

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Alfred\\_P\\_Wahabby&oldid=730](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Alfred_P_Wahabby&oldid=730)"

Category: Contributors

- 
- This page was last modified on 5 December 2017, at 15:03.
  - This page has been accessed 2,454 times.

# Alyssa Newman

From CU Denver Optimization Student Wiki

## About Me

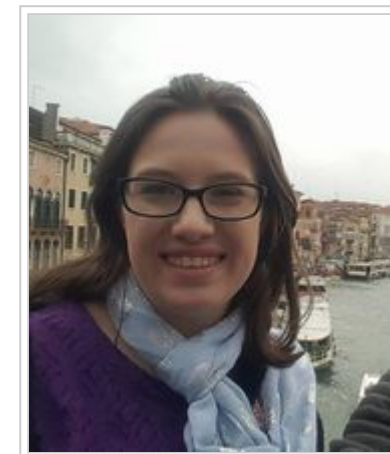


cat tax

I am a Ph.D. student at CU Denver, I have a bachelor's degree in mathematics from William Smith College in Geneva New York and a Master's Degree from CU Denver. When not doing math I enjoy anything artistic, my apartment is decorated with original pieces! My more recent artsy endeavors are mostly based around my niece. I have made fun and nerdy onesies in the past and am trying to crochet things for her, but there is quite the learning curve. I have a very spoiled cat who is frequently featured in my video feed during zoom classes or meetings (her name is Chloe, she is not helping). My favorite TV show is and forever will be Futurama.

## Projects

My current project is Circut Analysis of the Maximum Clique Problem. This project is for the topics in optimization course.I have previously worked on the project What are multicommodity flows? working with Zane Showalter-Castorena, that project was for the network flows course, Optimizing Highschool Graduation Rates working with Collin Powell and Zane Showalter-Castorena for the linear programming course, and Return To School Success In Times of COVID, for the integer programming course.



Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Alyssa\\_Newman&oldid=3464](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Alyssa_Newman&oldid=3464)"

Category: Contributors

- This page was last modified on 29 November 2021, at 19:12.
- This page has been accessed 13,897 times.

# Amanda Ward

From CU Denver Optimization Student Wiki

## About Me

My name is Amanda Ward and I'm a second-year Master's student at CU Denver.

## Projects

MATH 6404 Final Project (Spring 2025): van der Waerden's Theorem

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Amanda\\_Ward&oldid=4935](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Amanda_Ward&oldid=4935)"

Category: Contributors

- 
- This page was last modified on 23 April 2025, at 14:34.
  - This page has been accessed 27 times.

# Amber Rosacker

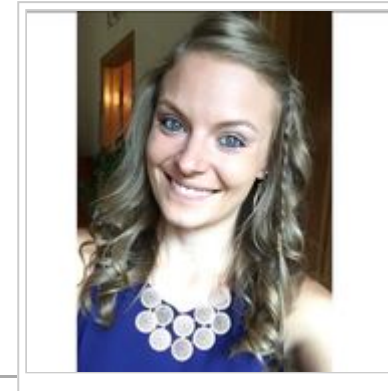
From CU Denver Optimization Student Wiki

Hello! My name is Amber Rosacker. I am currently working on my Master's in Mathematics Education from CU Denver. I received my Bachelor of Science in Mathematics Education from the University of Northern Colorado in 2014. I am a high school mathematics teacher in a suburb of Denver. I love being a teacher, and I could not imagine doing anything else with my life.

Follow this link to view my project on Class Scheduling

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Amber\\_Rosacker&oldid=2273](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Amber_Rosacker&oldid=2273)"

Category: Contributors



- This page was last modified on 27 November 2019, at 19:19.
- This page has been accessed 1,070 times.

# Amit Sengupta

From CU Denver Optimization Student Wiki

I am a second year PhD student in statistics at University of Colorado at Denver. I have obtained MS in statistics from Washington State University and have MS in computer science as well. I have worked for Oracle Corp. at San Francisco Bay Area as a database software developer and have successfully taught computer science at colleges and universities in the Bay Area. My publications in the area of efficient group communication algorithms in distributed systems have appeared in the leading international journal and conference proceedings. My research interest lies at the intersection of statistics and algorithms.

Shortest Path Routing Algorithms Amortized Analysis

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Amit\\_Sengupta&oldid=2546](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Amit_Sengupta&oldid=2546)"

Category: Contributors

- 
- This page was last modified on 22 April 2020, at 22:48.
  - This page has been accessed 1,271 times.

# Amortized Analysis

From CU Denver Optimization Student Wiki

My project involves amortized analysis using the aggregate method and potential method for the stack operations and incrementing a binary counter.

## Abstract

Amortized analysis is used when most operations are fast but an occasional operation is slow. Time needed to do a sequence of operations is averaged over all operations. This analysis involves no probability distribution and is not sensitive to a rare instance. The aggregate method and potential method for stack operations and incrementing a binary counter are discussed here and improvement of the amortized cost of  $n$  operations over the worst case cost of  $n$  operations is demonstrated. Another application is operations in hash table where most insert/or find operations require  $O(1)$  time but an occasional insert or find can take  $O(n)$  time due to collision.

## Introduction

The goal of the analysis of an algorithm is to check how the running time of the algorithm scales with the size of the put. The analysis is classified into four categories: (a) empirical analysis; (b) average case analysis; (c) worst case analysis, and (d) amortized analysis. The empirical analysis consists of writing a program for the algorithm and test the performance of the algorithm on some problem instances. Its drawbacks are the following: (a) it is expensive and time consuming; (b) it depends on the computing resources and programmer skills, and (c) it is often inconclusive. The idea of the average case analysis is to estimate the expected number of steps of the algorithm based on a probability distribution. The average case analysis suffers from the following drawbacks: (a) analysis depends on the choice of the probability distribution; (b) analysis is difficult, and (c) performance prediction depends on situations where you solved many problem instances. The worst case analysis gives an upper bound on the number of steps the algorithm takes on an instance. The analysis offers the following benefits: (a) it is independent of the computing environment; (b) analysis is easier, and (c) it is conclusive about comparing algorithms. But its limitation is that a rare instance can determine the performance of the algorithm. Amortized analysis is used when most operations are fast but an occasional operation is slow. Time needed to do a sequence of operations is averaged over all operations. This analysis involves no probability distribution and is not sensitive to a rare instance.

## Aggregate Method

Consider a stack  $S$  (of size  $n$ ) and these operations: (a) Push ( $S, x$ ) pushes object  $x$  into stack  $S$ ; (b) Pop ( $S$ ) pops the top of the stack  $S$  and returns the object, and (c) Multipop ( $S, k$ ) pops  $k$  top objects of the stack  $S$ . Suppose we have a sequence of  $n$  Push, Pop, and Multipop operations. The worst case cost of a Multipop operation is  $O(n)$  implying that the worst case cost of any operation is  $O(n)$ . So the cost of a sequence of  $n$  operations is  $O(n^2)$ . We can obtain a better bound using the aggregate method. Each object can be popped at most once for each time it is pushed. So the number of times Pop is called including calls in Multipop is at most the number of Push which is at most  $n$ . Hence a sequence of  $n$  Push, Pop, and Multipop operations require  $O(n)$  time implying that the amortized cost of one operation is  $O(1)$ , an improvement over  $O(n)$ . Let us see how the aggregate method is useful in the case of incrementing a binary counter.

Increment algorithm is described in this way. Increment ( $A$ ) {

```
i=0;
While ( i < length (A) and A[i]=1) {
Typesetting math: 100% i=0;
```

```
        i=i+1;
    }
    If ( i < length(A) )
        A[i]=1;
```

} Notice that all increment operations in the worst case take  $O(k)$  time if all  $k$  bits are 1. So  $n$  increment operations take  $O(nk)$  time in the worst case. But we can get a better bound with the aggregate method. A[0] flips each time increment is called. A[1] flips every second time increment is called. A[2] flips every fourth time increment is called. So the amortized cost of  $n$  increments =  $n(1 + 1/2 + 1/4 + \dots \text{ up to } \log n \text{ terms}) \leq 2n$ . Consequently, the amortized cost of one operation is  $(O(n))/n = O(1)$ , an improvement over  $O(k)$ .

## Potential Method

The potential method is defined as follows. Let  $c_i$  be the actual cost of the  $i$ -th operation and  $D_i$  is the data structure after the  $i$ -th operation. Let  $\phi$  be the potential function that maps  $D_i$  to a real number  $\phi(D_i)$ . Then the amortized cost of the  $i$ -th operation is given by  $(cc_i) = c_i + \phi(D_i) - \phi(D_{i-1})$ . Consider the example of stack. Define  $\phi$  as the number of elements in the stack. Then the amortized cost of Push is  $1 + ((k+1) - k) = 2$ . The amortized cost of Pop is  $1 + ((k-1) - k) = 0$ . Similarly, the amortized cost of Multipop =  $k + (-k) = 0$ . It follows that the amortized cost of  $n$  operations is  $O(n)$  and the amortized cost of one operation is  $O(1)$ , an improvement over  $O(n)$ .

Consider the example of incrementing a binary counter. Define the potential as the number of 1s after the  $i$ -th operation, say  $y_i$ . Let the  $i$ -th operation resets  $x_i$  bits. So the cost of the  $i$ -th operation is  $x_i + 1$  (1 for setting one bit). The number of 1s after the  $i$ -th operation is  $y_i \leq y_{i-1} - x_i + 1$ . So  $\phi(D_i) - \phi(D_{i-1}) \leq (y_{i-1} - x_i + 1) - y_{i-1} = 1 - x_i$ . Therefore  $(cc_i) = c_i + \phi(D_i) - \phi(D_{i-1}) \leq (x_i + 1) + (1 - x_i) = 2$ . The amortized cost of  $n$  operations is  $O(n)$  or equivalently, the amortized cost of one operation is  $O(1)$ , an improvement over  $O(n)$ .

## Summary

Amortized analysis is used when most operations are fast but an occasional operation is slow. The aggregate method and potential method for stack operations and incrementing a binary counter are discussed here and improvement of the amortized cost of  $n$  operations over the worst case cost of  $n$  operations is demonstrated. Another application is operations in hash table where most insert/or find operations require  $O(1)$  time but an occasional insert or find can take  $O(n)$  time due to collision.

## Reference

Thomas Cormen, Charles Leiserson, and Ronald Rivest, *Introduction to Algorithms*, MIT Press (1990).

Ravindra Ahuja, Thomas Magnanti, and James Orlin, *Network Flows*, MIT Press (1993)

## Github link

<https://github.com/amitsengupta1/amortized-analysis> Amit Sengupta  
Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Amortized\_Analysis&oldid=2647"

- This page was last modified on 2 May 2020, at 16:10.
- This page has been accessed 1,266 times.



# Amortized Time

From CU Denver Optimization Student Wiki

**Amortized analysis** is a technique for finding the time complexity of algorithms which gives a much tighter upper bound on the actual run time than worst-case analysis.

## Contents

- 1 Motivation
- 2 Techniques
- 3 Examples
  - 3.1 Dynamic Array
  - 3.2 Fibonacci Heap
- 4 Applications

## Motivation

Worst-case analysis can give horrible upper bounds on the running time of an algorithm. An iteration may, in the worst case, take a long time. For worst-case analysis, we would assume every iteration takes a long time. However, this worst case does not happen very often (for instance, only once every  $2^k$  iterations). Amortized analysis is a precise way to ``average out *these worst-case scenarios to produce a more realistic upper bound on the running time of the algorithm. It is most applicable when there is a data structure whose state is made ``better* by the worst case step, so the worst case will not happen again for a long while after.

## Techniques

*Aggregate analysis* calculates an upper bound,  $T(n)$ , on the running time of  $n$  iterations, and returns  $T(n)/n$  as the amortized running time.

The *accounting method* assigns an *amortized cost* to each step of the algorithm, which is different from the actual running time of that step. The idea is that steps which run faster than their amortized cost will build up "credit" which can be spent on slower running steps later in the algorithm.

The *potential method* is a version of the accounting method where the accumulated credit is tracked by a function of the data structure called the potential function. The amortized cost is the actual cost plus the change in the potential function.

# Examples

## Dynamic Array

An *array* is ordered data stored all in one block of memory. The array is *dynamic* when we expand the block of memory depending on how much data is being stored. Consider a dynamic array with an initial block of memory of size 1 set aside for the array. As data is added to it, the size of the array will double if it is full. This requires a new block of memory and the copying of all the data from the old block of memory to the new one.

Consider inserting  $n$  pieces of data into the dynamic array. In the worst case, an insert will require copying all of the data to a new block of memory, which will take  $O(k)$  time, where  $k$  is the number of elements currently in the array. Since there can be up to  $n$  elements in the array, this means each insert takes  $O(n)$  time in the worst case. Since there are  $n$  inserts, this takes  $O(n^2)$  time according to worst-case analysis.

Notice that most insertions only require  $O(1)$  time. The insertions which require  $O(k)$  time occur relatively infrequently. In particular, after an insertion which requires  $O(k)$ , the next  $k$  insertions will require only  $O(1)$  time. These facts suggest that amortized analysis may be beneficial.

Let  $\phi = 2k - K$  be the potential function, where  $k$  is the number of elements currently in the array and  $K$  is the size of current block of memory set aside for the array. Since, after the first insertion, the array is never less than half full, the potential is always nonnegative.

An insert into a non-full array takes 1 unit of actual time. The potential function will increase by 2, since  $k$  increases by 1 and  $K$  does not change. Therefore, the amortized time for such an insert is 3 units. This is constant time.

An insert into a full array takes  $k$  units of actual time. Before the insert,  $k = K$ , meaning the potential is  $K$ . After copying (but before the insert), the new block of memory has size  $2K$ , and there are still  $K$  pieces of data in it, so the potential is 0. After inserting, the number of elements currently in the array is  $K + 1$ , and so the potential is 2. Therefore, the total change in potential is  $2 - K$ . This means that the amortized time for inserts of this type is  $k + 2 - K = K + 2 - K = 2$  units. This is also constant time.

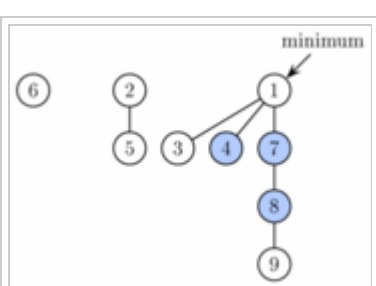
From the above analysis, we see that any insert takes constant amortized time. Therefore,  $n$  inserts will take  $O(n)$  amortized time. Looking at the actual run times of  $n$  insertions to a dynamic array show that this is closer to the reality.

## Fibonacci Heap

As part of Dijkstra's algorithm, a data structure known as a priority queue ([https://en.wikipedia.org/wiki/Priority\\_queue](https://en.wikipedia.org/wiki/Priority_queue)) is needed. The most efficient implementation of a priority queue known so far is a Fibonacci heap ([https://en.wikipedia.org/wiki/Fibonacci\\_heap](https://en.wikipedia.org/wiki/Fibonacci_heap)). This stores the data with key in a series of trees. Each tree maintains the heap property ([https://en.wikipedia.org/wiki/Heap\\_\(data\\_structure\)](https://en.wikipedia.org/wiki/Heap_(data_structure))), which is that the key of the parent is less than or equal to the keys of the children. This ensures that the minimum key occurs in the root of one of the trees. The heap also maintains the property that each non-root node can have at most one child cut from it. This is ensured by marking non-root nodes when a child is cut from them. A pointer is maintained to the tree root with minimum key.

The basic operations in a Fibonacci heap are `cut_child` and `combine_trees`.

- `cut_child` removes a node from its parent, making it the root of a new tree. If the parent is unmarked, it is marked. If the parent is marked, `cut_child` is performed on



An example of a Fibonacci heap. The blue nodes are marked.

- `combine_trees` roots of the same degree are combined into one tree by making the root with larger key the child of the other root. This is continued until all trees have different degrees.

These basic operations are combined to create the priority queue operations:

- **Insert:** Add the new node as its own tree.
- **Find-min:** Return the node which the minimum pointer is pointing to.
- **Decrease-key:** Decrease the key of the appropriate node. If its key is now smaller than the key of its parent, perform a `cut_child` operation on it. (This may cause a cascade of `cut_childs`.)
- **Delete-min:** Perform a `cut_child` on the children of the root with minimum key, then delete that root. Perform a `combine_trees` operation to reduce the number of trees. Then search through the roots for the new minimum and update the minimum pointer.

Let the potential function be  $\phi = t + 2m$ , where  $t$  is the number of trees in the heap, and  $m$  is the number of marked nodes. Using this, the amortized time of the above operations can be determined.

- **Insert:** Adding a new node as its own tree takes constant time, and increases the potential by one. Therefore, the amortized time for this operation is constant ( $O(1)$ ).
- **Find-min:** Returning the node pointed to takes constant time, and has no effect on the potential. Therefore, the amortized time for this operation is constant ( $O(1)$ ).
- **Decrease-key:** A single `cut_child` operation can, if there are several marked nodes, cause an arbitrarily long cascade of `cut_child` operations, and so take a long time. Let  $k$  be the number of `cut_child` operations done during a certain decrease-key operation. Each `cut_child` creates a new tree, and so  $k$  such operations will increase the potential by  $k$ . However, all but the first node to which `cut_child` was applied were marked, decreasing the number of marked nodes by  $k - 1$ . The parent of the last node `cut_child` was applied to was not marked, and now will be, increasing the number of marked nodes by one. In total, the number of marked nodes has decreased by  $k - 2$ . This means the potential has decreased by  $2k - 4$ . In total, the potential has changed by  $k - 2k + 4 = -k + 4$ . Since  $k$  `cut_child` operations take  $O(k)$ , as each individually takes constant time, the amortized time for this operation is  $O(k - k + 4) = O(1)$ .
- **Delete-min:** Let  $m$  be the degree of the root with minimum key. It can be shown that  $m = O(\log n)$ . The first phase of this operation involves  $m$  `cut_child` operations, which take  $O(m)$  time and increase the potential by  $m$ . Therefore, the amortized time for the first phase is  $O(m) = O(\log n)$ . The second phase of this operation involves a single `combine_trees` operation, which takes time proportional to the number of trees. It can be shown that the number of trees at the beginning of the delete-min operation is  $O(\log n)$ , and so at the beginning of the second phase, the number of trees is  $O(m + \log n) = O(m)$ . Therefore, the `combine_trees` operation takes  $O(m)$  time. At the end of the `combine_trees` operation, there are again  $O(\log n)$  trees, and so the potential decreases by  $O(\log n - m)$ . Therefore, the amortized time for the second phase is  $O(m + \log n - m) = O(\log n)$ . The third phase involves searching through the  $O(\log n)$  trees for the new minimum, and so takes  $O(\log n)$  time and has no effect on the potential. This shows that the total amortized time for the delete-min operation is  $O(\log n)$ .

Using a Fibonacci heap as the priority queue in Dijkstra's algorithm gives the best known strongly polynomial running time of  $O(m + n \log n)$ , where  $n$  is the number of nodes in the network, and  $m$  is the number of arcs.

## Applications

- Heaps ([https://en.wikipedia.org/wiki/Heap\\_\(data\\_structure\)](https://en.wikipedia.org/wiki/Heap_(data_structure))) are the preferred way to implement priority queues ([https://en.wikipedia.org/wiki/Priority\\_queue](https://en.wikipedia.org/wiki/Priority_queue)) because of their good amortized running times. For comparison, the naive implementation of a priority queue has  $O(1)$  worst-case time for insert and decrease-key, but  $O(n)$  for find-min, and delete-min.
- Binary Heaps ([https://en.wikipedia.org/wiki/Binary\\_heap](https://en.wikipedia.org/wiki/Binary_heap)) have  $O(1)$  amortized time for find-min, and  $O(\log n)$  amortized time for all other operations (insert, delete-min, decrease-key). They are sometimes preferred over Fibonacci heaps because of their simplicity.
- Fibonacci Heaps ([https://en.wikipedia.org/wiki/Fibonacci\\_heap](https://en.wikipedia.org/wiki/Fibonacci_heap)) have  $O(1)$  amortized time for all operations except delete which takes  $O(\log n)$  amortized time. This is the best so far with exact heaps.
- Soft Heaps ([https://en.wikipedia.org/wiki/Soft\\_heap](https://en.wikipedia.org/wiki/Soft_heap)) have  $O(1)$  amortized time for all operations, but "corrupt" a fraction of the priorities inserted. Specifically, for a chosen  $\epsilon$  between 0 and  $1/2$ , the soft heap corrupts  $\epsilon n$  priorities, where  $n$  is the number of elements inserted so far.
- Binary Search Trees ([https://en.wikipedia.org/wiki/Binary\\_search\\_tree](https://en.wikipedia.org/wiki/Binary_search_tree)) have many different implementations, most of which have  $O(\log n)$  amortized time for search, insertions, and deletions. These are another possible way to implement priority queues.
- Disjoint Sets ([https://en.wikipedia.org/wiki/Disjoint-set\\_data\\_structure](https://en.wikipedia.org/wiki/Disjoint-set_data_structure)) (also known as Union Find), when implemented efficiently, has  $O(\alpha(n))$  amortized time for union and find operations, where  $\alpha(n)$  is the inverse of the Ackermann function ([https://en.wikipedia.org/wiki/Ackermann\\_function](https://en.wikipedia.org/wiki/Ackermann_function)).

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Amortized\\_Time&oldid=1233](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Amortized_Time&oldid=1233)"

- This page was last modified on 28 April 2018, at 14:09.
- This page has been accessed 1,596 times.

# An Integer Linear Programming Approach to Graph Coloring

From CU Denver Optimization Student Wiki

The authors of this project are Megan Duff and Rebecca Robinson.

## Contents

- 1 Abstract
- 2 Introduction to Graph Coloring
- 3 Applications of Graph Coloring
- 4 Assignment Linear Program
- 5 Set Covering Linear Program
- 6 Extension to a Sudoku Model
- 7 Links to Code
- 8 Presentation
- 9 References

## Abstract

A coloring of the vertices of a graph  $G$  is called a proper vertex coloring of  $G$  if no two adjacent vertices get the same color. The chromatic number of  $G$ , denoted  $\chi(G)$ , is the fewest number of colors that give a proper vertex coloring of  $G$ . Finding  $\chi(G)$  is an NP-hard problem. We consider two different integer linear programs to find  $\chi(G)$ . The assignment linear program approach assigns colors to the vertices of  $G$  under the constraints of a proper coloring and then counts the number of colors used in total. The set covering linear program instead minimizes the number of independent sets (that is, subsets of the vertices such that none of the vertices are adjacent to each other) that cover the vertices of  $G$ . We can then color each independent set a different color to form the proper coloring. This gives us the chromatic number of  $G$ . We discuss the benefits and downfalls of each program. In addition, we discuss the applications of these programs to solving a Sudoku puzzle, providing an alternative approach to methods done in class previously.

## Introduction to Graph Coloring

A **graph** is a set of vertices connected by edges. We can color the vertices of a graph  $G$  to form a vertex coloring of  $G$ . A vertex coloring is considered a **proper vertex coloring** when no two adjacent vertices are colored the same color. Here, we refer to a proper vertex coloring just as a proper coloring. The **chromatic number** of a graph  $G$ , denoted  $\chi(G)$  is the smallest number of colors needed to have a proper coloring.

There are several known bounds on the chromatic number. There is the trivial bound  $\chi(G) \leq n$ , where  $n$  is the number of vertices in  $G$ . We can lower this bound with a bound based off the independence number,  $\alpha(G)$ , which is the size of the largest independent set in  $G$ . An **independent set** is a set of vertices such that none of the vertices in the set are adjacent to each other. Using the independence number,  $\chi(G) \leq \frac{n}{\alpha(G)}$ . Brooks' theorem is another upper bound on the chromatic number. In this case, we can bound the chromatic number by the maximum degree  $\Delta$ :  $\chi(G) \leq \Delta(G)$  unless  $G$  is a complete graph or an odd cycle, in which case  $\chi(G) = \Delta(G) + 1$ .

The greedy coloring algorithm is an approach to try to find a proper coloring of a graph. Then, from the proper coloring, we can get the number of colors used for that coloring. For a graph  $G$ , label the vertices  $v_1, v_2, \dots, v_n$  and for each vertex in order, color it with the lowest color available. Greedy coloring can be done in linear time, but unfortunately does not often use the lowest number of colors possible. There is always an ordering of the vertices that will produce an optimal coloring, but there are  $n!$  different orderings of the vertices. Consider a complete bipartite graph with the perfect matching edges removed. Since this is a bipartite graph, only two colors are needed to properly color it. However, there is a labeling that produces a coloring with  $\frac{n}{2}$  colors. Thus, greedy coloring isn't the best method to try to find the chromatic number. In fact, finding  $\chi(G)$  is NP-hard. We compare two linear programs that find the chromatic number of a graph -- an assignment linear program and a set covering linear program.

## Applications of Graph Coloring

Having programs that find the chromatic number is useful due to the applications of graph coloring. For example, consider the situation of assigning radio frequencies. We cannot assign the same frequency to radio stations that are too close together, otherwise there would be interference. If we create a vertex for each radio station, and connect two vertices if the corresponding radio stations are too close together. Then, a proper coloring of the graph would give an assignment of radio frequencies to the stations. The chromatic number would be the smallest number of frequencies needed.

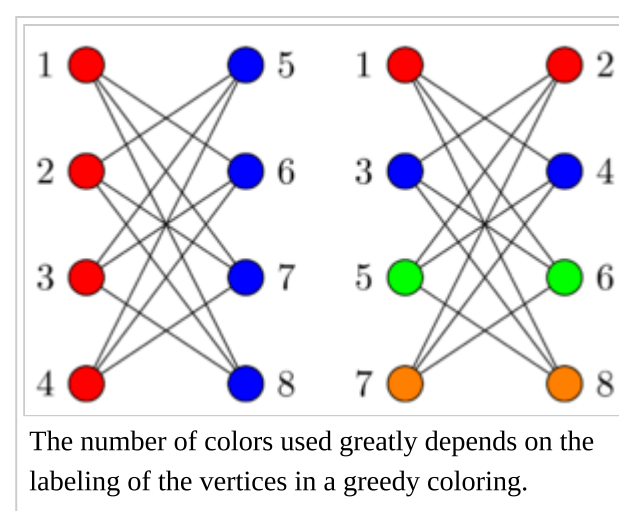
We can also consider a scenario when zookeepers are trying to put different animals into enclosures. Perhaps they are trying to minimize the number of enclosures needed due to cost. A vertex is created for each animal, and two vertices are adjacent if the corresponding animals can not be in the same enclosure. A proper coloring is an assignment of the animals to enclosures, and the chromatic number gives the smallest number of enclosures needed.

Resource allocation is also an application of graph coloring. We can create a vertex for each task that needs to be done and draw an edge between the two vertices if the two tasks share a resource (and thus cannot be done simultaneously). A proper coloring of the task graph ensures that no two tasks that share a resource are done at the same time. The chromatic number would give the most efficient way to perform the tasks simultaneously.

Graph coloring can also be used to create Sudoku puzzle solvers, which will be done later.

## Assignment Linear Program

The assignment linear program assigns a color to every vertex and ensures that adjacent vertices are not assigned the same color. In minimizing the number of colors used, this linear program finds the chromatic number for a graph.



With  $x_{v,i} = 1$ , if vertex  $v$  is assigned color  $i$  and  $x_{v,i} = 0$  otherwise, and  $w_i = 1$  is color  $i$  is used and  $w_i = 0$  otherwise. The number  $H$  is the largest number of colors possibly needed which would be the number of vertices for a particular graph.

The downfalls of this program is the exponentially many equivalent solutions as well as there being no continuous relaxation to this program. Thus it remains an NP-hard problem.

Running this program on a selection of different graphs provides the following results:

- Peterson Graph -- Run Time: 0.102745 s
- 70 Vertices Path Graph--Run Time: 0.606761 s
- 140 Vertices Path Graph--Run Time: 2.16679 s
- 200 Vertices Path Graph--Run Time: 7.1204 s
- 60 Vertices Crown Graph -- Run Time: 19.8884 s
- Mycielski Graph of Order 6 -- Run Time: 338.305 s

As one can see, increasing the number of vertices and edges does increase the run time for graph, in comparison to similar structured graphs with less vertices, but having a lot of vertices and edges does not inherently make the program run longer. The longest running graph has less vertices and edges than the 60 vertices crown graph but due to more complicated structures of the graph itself it caused the program to take over 5 minutes to find the chromatic number versus 19s for the 60 vertices semi-complete bipartite graph.

## Set Covering Linear Program

The set covering linear program assigns a minimal independent set vertex covering of a graph. An independent set is a set of vertices that are not pairwise adjacent. Thus searching for a minimal independent set vertex covering results in the chromatic number of the graph. By assigning each independent set a color, this results in a proper vertex coloring since all vertices in the set are not adjacent and therefore no two adjacent vertices would be assigned the same color. Looking for the least amount of independent sets to cover all the vertices thus results in the smallest number of colors needed.

The downfalls of this program are the exponential number of variables. For example, the Peterson graph with 10 vertices and 15 edges has more than 50 independent sets. Finding all of the independent sets is also not straightforward and could be room for error since the user has to come up with all the independent sets of the graph and is not simply given them like they are with vertices and edges. Our original hope was to compare the run times of the two programs but even in our simplest example (the Peterson graph) finding all the independent sets proved challenging. Thus the set covering linear program offers another alternative for finding the chromatic number of the graph but in actuality is not very practical.

## Extension to a Sudoku Model

The first step in extending the assignment linear program to solving a Sudoku puzzle is to translate the Sudoku grid into a graph. We construct a vertex for every square in the grid. We connect two vertices if and only if the corresponding squares on the Sudoku grid are in the same row, column, or subgrid. Here, for an easier calculation, we use a 4x4 graphs become large quite quickly as the size of the Sudoku grid increases. A 4x4 grid contains 16 vertices and 56 edges. A proper coloring with 4 colors of

(ASS-S)

$$\begin{array}{ll} \min & \sum_{1 \leq i \leq H} w_i \\ \text{s.t.} & \sum_{i=1}^H x_{vi} = 1 \quad \forall v \in V \\ & x_{ui} + x_{vi} \leq w_i \quad \forall (u, v) \in E, i = 1, \dots, H \\ & x_{vi}, w_i \in \{0, 1\} \quad \forall v \in V, i = 1, \dots, H \end{array}$$

(COV)

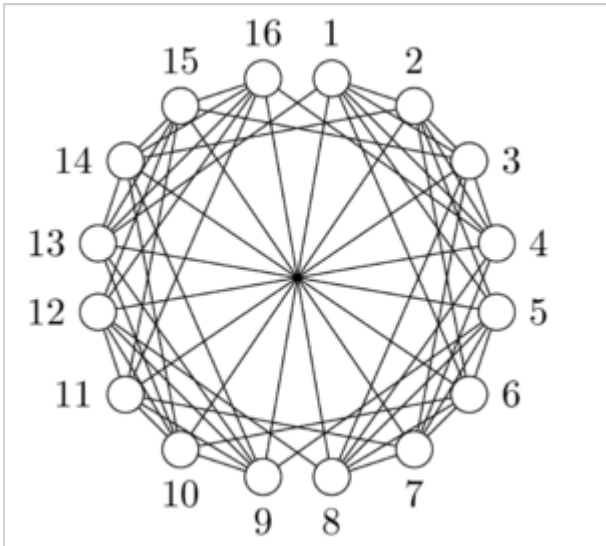
$$\begin{array}{ll} \min & \sum_{s \in S} x_s \\ \text{s.t.} & \sum_{s \in S: v \in s} x_s \geq 1 \quad \forall v \in V \\ & x_s \in \{0, 1\} \quad \forall s \in S \end{array}$$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

The labeling of the 4x4 Sudoku grid.

this graph would be a solution to the Sudoku puzzle. The next step in extending the assignment linear program to solving this Sudoku puzzle is to be able to account for the squares that are already filled in. These pre-filled squares correspond to pre-colored vertices. In our program, we will need to account for these vertices. Then, having the linear program find the coloring will be analogous to a solution to the puzzle.

Consider the 4x4 Sudoku puzzle. We label each square in the Sudoku grid as in the image on the left. With each vertex being labelled as such, we can input the set of edges into the assignment linear program as before. We also need to input the pre-filled squares. We simply create a set that we fill with the pre-assignment of colors. Then, a constraint is added to set  $x_{vi} = 1$  if vertex  $v$  gets color  $i$  in the pre-coloring. The assignment linear program will then extend the pre-coloring to a full 4-coloring, which corresponds to a solution.



The corresponding graph to the 4x4 Sudoku grid.

## Links to Code

The AMPL code for the assignment integer linear program and the Peterson, Path, Crown Graph and Mycielski Graph of Order 6 is located at:

<https://github.com/duffme/graphcoloringLP/>

The AMPL code for the extension of the assignment integer linear program to a Sudoku solver is located at:

<https://github.com/rebrobin/sudokugraphcoloring/>

## Presentation

Our presentation slides can be found at:

<https://github.com/rebrobin/presentation/blob/master/An%20Integer%20Linear%20Programming%20Approach%20to%20Graph%20Coloring.pdf>

## References

Droogendijk, Leen. “A Triangle-Free, 6-Chromatic Graph with 44 Vertices.” Mathematics Stack Exchange, 1 Feb. 1966, <https://math.stackexchange.com/questions/1561029/a-triangle-free-6-chromatic-graph-with-44-vertices>.

Dhaharavath, Bharathi. Sudoku Game Solution Using Graph Coloring. <http://www.cs.kent.edu/~dragan/ST-Spring2016/SudokuGC.pdf>.

“Independent Set (Graph Theory).” Wikipedia, Wikimedia Foundation, 25 Nov. 2019, [https://en.wikipedia.org/wiki/Independent\\_set\\_\(graph\\_theory\)](https://en.wikipedia.org/wiki/Independent_set_(graph_theory)).



- This page was last modified on 3 December 2019, at 16:00.
- This page has been accessed 7,799 times.

# An Overview of Dijkstra's Algorithm

From CU Denver Optimization Student Wiki

Author: Lillian makhoul Github repo link: <https://github.com/makhoullillian153/An-Overview-of-Dijkstra-s-Algorithm>

This page was put together by Lillian makhoul as a part of a project for the course Applied Graph Theory.

## Contents

- 1 Minimum Spanning Trees
- 2 Outlining Dijkstra's Algorithm
- 3 Interesting applications
  - 3.1 Finding a Shortest Path Route for People with Mobility Impairments
  - 3.2 Vehicle Routing Using Oil Consumption Based Weights
- 4 Github
- 5 References

## Minimum Spanning Trees

A spanning tree of a graph  $G$  is defined to be a subgraph  $T$  that is a tree which spans every vertex in  $G$ . Further, the edge set of a minimum spanning tree includes the edges of minimal weight such that the definition of a spanning tree is satisfied.

The goal of Dijkstra's algorithm is to return a minimum spanning tree for any graph  $G$ .

## Outlining Dijkstra's Algorithm

Let  $G = (V, E)$  be a graph such that each  $uv \in E$ , has some weight  $w(uv)$ , and each  $uv \notin E$  has weight  $w(uv) = \infty$ . Setting the weight to infinity ensures that we will not select a non-existing edge to be in our minimum spanning tree.

**Input:** A weighted, connected, undirected graph  $G$  whose edge weights are non-negative; a source vertex  $u \in V(G)$ .

**Output:** A spanning tree  $T$  (also known as the "Dijkstra Tree") of  $G$ , rooted at vertex  $u$  such that

1. The unique path from  $u$  to each vertex  $z \in V(G)$  in  $T$  is a shortest path from  $u$  to  $z$  in  $G$ , and
2. The vertex labeling gives the distance from  $u$  to each vertex.

**Dijkstra( $G, u$ )**<sup>[1]</sup>:

- **initialize** the Dijkstra tree  $T$  as the given source vertex  $u$
- **initialize** the set of vertices  $S$  in  $T$  as  $\{u\}$
- **write** label 0 on vertex  $u$ :  $t(u) = 0$
- **while**  $S \neq V(G)$ 
  - **for each** vertex  $v \notin S$
  - **let**  $t(v) = \min_{z \notin S} t(z)$
  - **add**  $v$  to  $S$
  - **for each** edge  $vz, z \notin S$ 
    - **let**  $t(z) = \min\{t(z), t(v) + w(vz)\}$
    - **if**  $t(z) = \infty$ 
      - **return** Dijkstra tree  $T$  and its vertex labels
- **return** Dijkstra tree  $T$  and its vertex labels

## Interesting applications

### Finding a Shortest Path Route for People with Mobility Impairments

Lack of ramps impose a restriction on these routes. Streets with particularly bumpy sidewalks, stairs, etc. are known to have higher weights than those with smoother sidewalks and ramps, for example. Arellano, et. al.<sup>[2]</sup> takes the node of the tree to be the starting point of the person, and look for the shortest path to the destination such that edge weights are minimized. They implement Dijkstra's algorithm to reduce the time and effort for mobility-impaired people in the city of Quito. In the results, we see a comparison the average travel time generated by the algorithm to the actual average travel time to reach the destination using this generated path. This second average was found by having 45 participants who fit the population of interest execute the routes given by the algorithm.

**Route Times -- Application and Test Comparative**<sup>[2]</sup>

Route	Average Time (Algorithm)	Average Time (Experimental)
Plaza Grande	08:47:00	09:15:00
Circulo Militar	04:26:00	04:23:00
San Francisco	04:21:00	03:51:00

For the Plaza Grande neighborhood, the algorithm underestimated how long it took to actually travel the given paths, however for the other two neighborhoods, we saw a decrease in the actual average time it takes to travel the paths, with San Franciso's paths seeing the greatest difference.

# Vehicle Routing Using Oil Consumption Based Weights

By implementing an appropriate route-planning algorithm, drivers save time in their journey, reduce oil consumption, and avoid congestion. This will in turn improve overall energy efficiency in a world that is becoming increasingly more energy consumptive. The algorithm as presented by Zhang, et. al.<sup>[3]</sup> bases its oil consumption weights (OCW) on distance, speed, driving time, idling time, driving oil consumption, and idling oil consumption. Zhang, et. al. performs a simulation-based experiment on both a Dijkstra's algorithm with distance-based weights, and on one with OCWs. Their results show that compared to the algorithm that only considers distance, accounting for oil consumption reduces ones' travel time significantly compared to using just distance. Furthermore, the actual travel time using OCW much better aligns with the theoretical time to reach a destination.

Simulation Results and Comparison<sup>[3]</sup>

Simulation*	Theoretical time, s	Actual time, s (Distance-based Dijkstra Algorithm)	Oil consumption (Distance-based Dijkstra Algorithm)	Actual time, s (OCW-Dijkstra Algorithm)	Oil consumption (OCW-Dijkstra Algorithm)
1	16.80	2044.80	750.34	42.17	57.53
2	29.93	405.93	178.53	46.37	63.70
3	17.39	99.39	55.07	19.19	28.19
4	35.99	435.99	196.86	55.17	58.41
5	118.00	118.00	115.23	118.00	115.23
6	184.43	409.41	184.43	74.99	101.32

\*Details on which source and destination nodes are used is given in detail in paper.

For simulation 5, we see that the theoretical and both actual times are equal. This is due to the idling time being zero seconds in this particular simulation. Otherwise, there is a strict improvement from one algorithm to the other.

## Github

In the GitHub repository, one can find the slide deck used to present this information.

## References

1.

↑

[1] ([https://www.google.com/books/edition/Combinatorial\\_Mathematics/bApVzQEACAAJ?hl=en&gbpv=0](https://www.google.com/books/edition/Combinatorial_Mathematics/bApVzQEACAAJ?hl=en&gbpv=0))Douglas B. West. Combinatorial Mathematics. Cambridge University Press, 2021. isbn: 9781107058583.

2.

↑

<sup>2.0</sup> <sup>2.1</sup> [2] ([https://link.springer.com/chapter/10.1007/978-3-030-16184-2\\_22](https://link.springer.com/chapter/10.1007/978-3-030-16184-2_22))Arellano, L., Del Castillo, D., Guerrero, G., Tapia, F. (2019). Mobile Application Based on Dijkstra’s Algorithm, to Improve the Inclusion of People with Motor Disabilities Within Urban Areas. In: Rocha, Á., Adeli, H., Reis, L., Costanzo, S. (eds) New Knowledge in Information Systems and Technologies. WorldCIST'19 2019. Advances in Intelligent Systems and Computing, vol 931. Springer, Cham.

Typesetting math: 100%

0.1007/978-3-030-16184-2\_22

3. <sup>3.0</sup> <sup>3.1</sup> [3] (<https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-its.2015.0168>)Zhang, J.-d., Feng, Y.-j., Shi, F.-f., Wang, G., Ma, B., Li, R.-s. and Jia, X.-y. (2016), Vehicle routing in urban areas based on the Oil Consumption Weight -Dijkstra algorithm. IET Intell. Transp. Syst., 10: 495-502. <https://doi.org/10.1049/iet-its.2015.0168>  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=An\\_Overview\\_of\\_Dijkstra%27s\\_Algorithm&oldid=4948](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=An_Overview_of_Dijkstra%27s_Algorithm&oldid=4948)"

---

- This page was last modified on 1 May 2025, at 12:15.
- This page has been accessed 29 times.

# Angela Morrison

From CU Denver Optimization Student Wiki

## Contents

- 1 Social Media/Contact Links
- 2 Little Bit About Me
  - 2.1 Education
  - 2.2 Programming Languages/Experience
  - 2.3 Fun Facts
- 3 Professional Things
  - 3.1 Projects



## Social Media/Contact Links

- Email (<mailto:angela.morrison@ucdenver.edu>)
- Twitter (<https://twitter.com/AngieM2630>)
- LinkedIn (<https://www.linkedin.com/in/angela-morrison-9a80006a/>)

## Little Bit About Me

Hello all! My name is Angela Morrison, and I am a fourth-year in the Ph.D. program at CU Denver. My hobbies include video games, rollerblading, bowling, and doodling. My office is 4216 in the Student Commons building if you ever want to stop by and chat.

## Education

1. Albion College, B.A. in Mathematics with minors in Computer Science and Economics
2. Michigan State University, M.S. in Industrial Mathematics

## Programming Languages/Experience

- COBOL
- CICS
- FORTRAN
- Java
- Matlab

- Python
- R
- SPSS
- SQL

## Fun Facts

- My favorite Pokemon is Jolteon
- I am the oldest of 3 siblings (which makes me the coolest)
- I'm left-handed, but do some activities like batting and throwing right-handed
- Back when I was in shape, I was a national qualifier for Division 3 Indoor and Outdoor Track Field (in the pentathlon and heptathlon respectively)
- Teaching is one of my biggest passions
- My go-to song for karaoke is "Without Me" by Eminem (I mean it sort of has to be since I'm from Michigan.)
- I hate coffee

## Professional Things

### Projects

In the fall of 2020 for Linear Programming, I worked with Weston Grewe on Creating Fair Voting Districts. In this project, we use a clustering algorithm to design voting districts that are easy to understand and minimize the distance a voter has to travel to a polling location.

In the spring of 2021 for Integer Programming, I worked with Weston Grewe on Using Trees to Get Into College. For this project, we used data from Massachusetts public schools to develop a decision tree to understand the most important features that lead to a student enrolling in college directly out of high school.

In the summer of 2021 as part of a readings course, I started the process of exploring a possible relationship between the simplex method and network simplex method in Exploring the Network Simplex Method.

In the fall of 2021 as part of the Topics in Optimization, I worked with Weston Grewe (I know, what a shocker!) on The Circuit Less Travelled: A Path of Gentrification Through Denver Neighborhoods. In this project we looked at the circuit walks that get us from one clustering of Denver neighborhoods to another. The neighborhood clusters were determined based on a k-means clustering of attributes that one can associate with gentrification.

For the spring of 2022, Weston Grewe and I (take that Connor and Zach!), as part of the Network Flows course, worked on the project A Wheelie Good Time: Safe Biking in Denver.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Angela\\_Morrison&oldid=4831](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Angela_Morrison&oldid=4831)"

Category: Contributors

- 
- This page was last modified on 8 August 2024, at 10:55.
  - This page has been accessed 2,494 times.

# Anne Kreeck

From CU Denver Optimization Student Wiki

## About Me

My name is Anne Kreeck and I am a first-year master's student at University of Colorado - Denver. I moved to Denver in August of 2024 following 7 years in Montana and have found it to be quite the adventure living in such a big city!

## Education

Montana State University: B.S. in Mathematics - Applied Mathematics Option

## Projects

Spring 2025: Exploring the Blossom Algorithm, a look at the blossom algorithm for my final project in MATH 6404 - Graph Theory.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Anne\\_Kreeck&oldid=5027](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Anne_Kreeck&oldid=5027)"

Category: Contributors

- 
- This page was last modified on 7 May 2025, at 13:14.
  - This page has been accessed 25 times.



# Applications of Column Generation

From CU Denver Optimization Student Wiki

## Contents

- 1 Overview
- 2 Application
  - 2.1 Problem Description
  - 2.2 AMPL Code
  - 2.3 Results
  - 2.4 Looking Forward
- 3 References

## Overview

Column generation is a useful method for solving linear programs with a large number of variables. Consider the cutting stock problem: Given rolls of a certain length and demand for rolls of shorter lengths, in what way can we cut the larger roles to satisfy the demand while minimizing waste? As the number of desired shorter lengths increases, the number of possible patterns exponentially increases. This makes the cutting stock problem a perfect candidate for column generation.

Instead of considering each possible pattern explicitly, we can ignore variables with a non-negative reduced cost. In this way, we can narrow our search to variables that have a negative reduced cost, i.e. the variables that decrease the total waste. This is done by splitting the problem into two problems. The master problem is the original problem but with a reduced number of variables. The subproblem (also known as the knapsack problem) finds the variables with a negative reduced cost for the master problem to consider.

## Application

### Problem Description

As mentioned in my personal page ([http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Alexa\\_desautels](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Alexa_desautels)), I work at a sandwich shop called Stack Subs. The wheat bread that we buy comes in loaves approximately 28" long and the possible sandwich sizes are 4", 6", and 8". We sell around 1200 sandwiches a week, of which 75 are 4" long, 625 are 6",long and 500 are 8" long. If we create a set of patterns to cut the whole loaves, what set will minimize the number of whole loaves cut?

First, let's write this problem in canonical form:

$$\min \sum_{i=1}^n x_i$$

Typesetting math: 100%

$$\begin{aligned} \text{s.t. } & \sum_{i=1}^n a_j^i x_i \geq d_j, & \forall j = 1, \dots, m \\ & x_i \geq 0 \end{aligned}$$

Here,  $x_i$  is the number of loaves cut in the  $i^{th}$  pattern,  $a_j^i$  is the number of  $j$  length sandwiches cut in pattern  $i$ , and  $d_j$  is the number of each size of sandwich demanded by the customers. This problem can be solved using AMPL.

## AMPL Code

The following is the code for the master problem:

[1]

```

set WIDTHS;                # set of widths to be cut
param orders {WIDTHS} > 0;  # number of each width to be cut

param nPAT integer >= 0;    # number of patterns
set PATTERNS = 1..nPAT;    # set of patterns
param roll_width >= 0;      # size of uncut rolls

param nbr {WIDTHS,PATTERNS} integer >= 0;

check {j in PATTERNS}:
    sum {i in WIDTHS} i * nbr[i,j] <= roll_width;

                                # defn of patterns: nbr[i,j] = number
                                # of rolls of width i in pattern j

var Cut {PATTERNS} integer >= 0;  # rolls cut using each pattern

minimize Number:                # minimize total raw rolls cut
    sum {j in PATTERNS} Cut[j];

subject to Fill {i in WIDTHS}:
    sum {j in PATTERNS} nbr[i,j] * Cut[j] >= orders[i];

```

In this portion of the model, we are minimizing the number of loaves cut. This is done by first checking that the length cut away from the loaf for each pattern does not exceed the loaf size. Then our objective is to minimize the number of loaves cut while making sure to keep up with the number of orders for each sandwich size.

Now we will look at the code for the subproblem:

[1]

```

param price {WIDTHS} default 0.0;

var Use {WIDTHS} integer >= 0;

```

Typesetting math: 100%

```

minimize Reduced_Cost:
  1 - sum {i in WIDTHS} price[i] * Use[i];

subject to Width_Limit:
  sum {i in WIDTHS} i * Use[i] <= roll_width;

```

Here we are finding patterns that have a negative reduced cost and eliminating those that don't.

The main code in the run file is:

[1]

```

let nPAT := 0;
for {i in WIDTHS} {
  let nPAT := nPAT + 1;
  let nbr[i,nPAT] := floor (roll_width/i);
  let {i2 in WIDTHS: i2 <> i} nbr[i2,nPAT] := 0;
}

repeat {
  solve Cutting_Opt;
  let {i in WIDTHS} price[i] := Fill[i].dual;

  solve Pattern_Gen;
  if Reduced_Cost < -0.00001 then {
    let nPAT := nPAT + 1;
    let {i in WIDTHS} nbr[i,nPAT] := Use[i];
  }
  else break;
}
display nbr, Cut;

```

The first loop initializes the first patterns to use. It does this by dividing the length of a full loaf by each sandwich size and creating a pattern with that number for that size and 0 for every other size.

In the next loop, we alternate solving the master problem and the subproblem. When no patterns have a reduced cost of less than -0.00001, the loop breaks and the results are displayed.

## Results

The patterns that waste the least amount of bread are shown in the following table, along with how many times they are used to satisfy the demand:

Pattern	Number Used
7 4" cuts	6
4 6" cuts	0
3 8" cuts	2
2 6" cuts, 2 8" cuts	247
1 4" cut, 4 6" cuts	33

As shown, there are 5 patterns but the vast majority of the patterns we need to cut is 2 4" sandwiches and 2 6" sandwiches.

I experimented using this cut at work and discovered that the loaves are not made of a uniform size. Some loaves were a bit longer and resulted in more waste and others were just short enough to only be able to make 3 sandwiches. I wanted to know if I would get different results if I varied the length of the loaves by 1". When I ran the code for 29", I had the same results. However, I recieved different results for 27". The following table displays my results for the 27" loaves:

Pattern	Number Used
6 4" cuts	0
4 6" cuts	0
3 8" cuts	55
5 4" cuts, 1 6" cuts	0
3 6" cut, 1 8" cuts	183
1 4" cut, 1 6" cut, 2 8" cuts	76

Many of these patterns have changed because of the shorter length and we now have 6 patterns instead of 5.

## Looking Forward

I began to wonder if it would be possible to include a range of sizes in one model. There are several ways:

First, you can change the length of the loaves to a random variable between 27 and 29. You would need to generate a new variable for every iteration once you have initialized the patterns in the run code. Another way of including a range of sizes would be to create a set of sizes (27, 28, and 29) and introduce a supply limit. This way we must use loaves of all three sizes, not just 28" which would be optimal.

## References

- ↑ <sup>1.0</sup> <sup>1.1</sup> <sup>1.2</sup> AMPL, R. Fourer, D.M. Gay, and B.W. Kernighan, Duxbury Press, 2002.

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Applications\_of\_Column\_Generation&oldid=1000"

- This page was last modified on 7 December 2017, at 23:44.
- This page has been accessed 3,133 times.

# Ari Holcombe Pomerance

From CU Denver Optimization Student Wiki

Hello! My name is Ari, and I'm a PhD student in the math department at University of Colorado Denver. My research interests include graph theory, optimization, and linear programming.

## Projects

For the Applied Graph Theory course in Spring 2025, I worked with Abigail Nix on a project on Graph Coloring Variants.

## External links

Github (<https://github.com/aripom>)

LinkedIn (<https://www.linkedin.com/in/ari-holcombe-pomerance-4b146217a/>)

NYT Mini Crossword Leaderboard (<https://www.nytimes.com/puzzles/leaderboards/invite/9560a017-4c64-4a2e-9e3c-6cf69200b5c3>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Ari\\_Holcombe\\_Pomerance&oldid=4844](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Ari_Holcombe_Pomerance&oldid=4844)"

Category: Contributors

### Ari D. Holcombe Pomerance



**Education** Macalester College (B.A.)

**Birth date** 2001

**Birth place** Boston, United States

- This page was last modified on 7 April 2025, at 16:34.
- This page has been accessed 30 times.

# Location Fluctuations in Denver Area Bias Motivated Crime Explored Through Linear Programming

From CU Denver Optimization Student Wiki

(Redirected from Assignment Application: Hate Crimes)

Redirect page

↳ Denver Hate Crime Mapping: Visualizing Fluctuations through Linear Programming

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?

title=Location\_Fluctuations\_in\_Denver\_Area\_Bias\_Motivated\_Crime\_Explored\_Through\_Linear\_Programming&oldid=1718"

---

- This page was last modified on 26 November 2018, at 10:13.
- This page has been accessed 666 times.

# Assignment of Reservoirs for Pumped Hydro Storage Systems

From CU Denver Optimization Student Wiki

This project is for MATH5490 Network Flows at the University of Colorado Denver, Spring 2020. Student contributor: Quinsen Joel

## Contents

- 1 Abstract
- 2 Context
- 3 Potential Formulations
  - 3.1 Matchings vs. Other Subgraphs
  - 3.2 Optimal vs. all Matchings
  - 3.3 Bipartite vs. General Matching
  - 3.4 Weighted Bipartite Matching vs. Maximum Bipartite Matching
  - 3.5 Balanced vs. Unbalanced Assignment
- 4 Assignment Representation
  - 4.1 Network Representation
  - 4.2 Solution methods
- 5 Conclusion
- 6 Link to GitHub page
- 7 References

## Abstract

The problem of matching disjoint sets of reservoirs for use in a Pumped Hydro Energy Storage system can be modeled as a network flow problem. Specifically; a bipartite network assignment problem.

## Context

Energy storage is a key requirement for the transition to wind and solar-based energy economy and still in the early stages of integration. Pumped Hydro Storage (PHS) is the world's most widely adopted and economical form of utility-scale energy storage so finding ways to expand and improve this technology is a worthwhile endeavor.



While old hydropower paradigms made use of existing river systems to generate power, a new paradigm of "closed-system" hydropower hopes to search for more environmentally-friendly reservoir sites away from existing rivers. A closed-system PHS site requires (at least) two reservoirs with enough vertical elevation difference to generate sufficient power, but as little horizontal distance as possible to minimize construction costs. Comprehensive searches for appropriate PHS sites can span thousands of square kilometers and are computationally expensive, so efficient search methods are of great interest.

A notable previous attempt of this comprehensive search by Bin Lu et. al.<sup>[1]</sup> has split the process into two parts; First, they performed a search for applicable reservoirs, and second, found PHS units from the reservoirs. The first part of their process has been open-sourced and made available for public use, while the second part has been left private. This project hopes to give a method for replicating the second part of this process, making use of the previous researchers' reservoir model outputs.

## Potential Formulations

In attempting this, an algorithmic formulation must be decided on that offers the best possible discovery of suitable PHS sites given the outputs of the reservoir model. Several considerations can help us arrive at this best option:

### Matchings vs. Other Subgraphs

It is possible that an economic configuration of more than two reservoirs in a single PHS system exists and should be studied further. However, for this method it is assumed that any individual PHS system must consist of exactly one upper and one lower reservoir for several reasons: First, the formulation behind a search for arbitrarily large connections of reservoirs would be very complex and possibly computationally intractable. Additionally, the popular mode of existing PHS systems is by and large that of exactly one upper and one lower reservoir. Lastly, providing this assumption allows us to model the comprehensive search for PHS sites as a search for matchings in a graph, or a certain set of arcs that are node-disjoint. Because matching problems are more well-known and offer standard algorithmic approaches for solving, they help narrow down our search for a robust algorithmic approach.

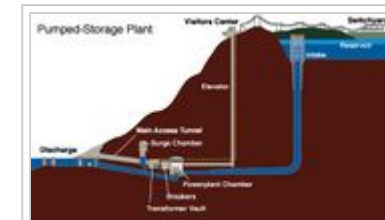
### Optimal vs. all Matchings

One possible goal of PHS siting could be to discover all potential matchings between reservoirs and let individual developers choose from them based on their own cost metrics and construction planning. From an algorithmic standpoint, this is essentially just generating a fully connected graph where each node is a reservoir and each arc represents a potential PHS system.

However, optimization of these matches would allow for a more useful and succinct summary of the potential for hydropower, saving developer time and money. No matter which cost metrics are applied and by whom, optimization will undoubtedly be used to increase the chances of successful deployment at some point, so a flexible framework using an optimization approach could be useful. Considering the large optimization task that is still left for stakeholders, taking care of this problem early, and with prudence could increase the chances for successful project completion.

### Bipartite vs. General Matching

A maximum horizontal distance threshold between areas of suitable elevation difference was one of the criteria used in the reservoir model to delineate areas for upper and lower reservoirs. It is not obvious that this distance threshold should be used to limit reservoirs matches, though; a match between very distant reservoirs could be viable if the



**Figure 4:** Pumped Hydro Storage

grow in a complex way as distance between reservoirs exceeds the distance threshold, disqualifying it from consideration (ie. a maximum distance threshold will be used to limit reservoir matches).

An interesting consequence of applying a maximum distance threshold on the reservoirs from the reservoir model is that it justifies the use of a bipartite network as the main structure for the formulation. A reservoir can be an "upper" reservoir for a PHS match within the maximum distance threshold if and only if it lies within the scouted "upper" reservoir area. The same is true for "lower" reservoirs and "lower" scouted areas. Thus, matches between any reservoirs that are within the maximum distance threshold must consist of reservoirs that belong to both sets. A simple transformation shows that a matching of two “upper” reservoirs can be relabeled as a match of an “upper” and “lower” reservoir (and the same is true for a match of two "lower" reservoirs).

Modeling the problem with a bipartite network has the added benefit of accessing various straightforward network flow algorithms meant for such a structure, narrowing our problem yet again. Non-bipartite matching, while a tempting method for discovering optimal, arbitrarily distant PHS sites, is not as easily solved by network flows (or any method) due to complications arising from negative cycles<sup>[2]</sup>.

### Weighted Bipartite Matching vs. Maximum Bipartite Matching

One major decision to make in the formulation is whether it is most important to discover as many PHS sites as possible (for further filtering and decision-making), or as productive (in terms of energy storage) of PHS sites as possible. Due to the high construction and environmental costs of even a single hydropower plant, this project makes the assumption that a set of optimally productive sites is best.

The choice of quality over quantity corresponds to the choice between maximum-weight matching formulation (classically, the Assignment problem) and a maximum matching formulation. Maximum-weight matchings are generally considered harder problems to solve because they cannot be reduced to max-flow algorithms like their counterpart, and must take arc costs into consideration<sup>[3]</sup>. Choosing this more difficult option follows the application goals more closely.

Choosing the weight metric to use in this maximum weight matching problem comes down to choosing the single metric that best captures the productiveness of a PHS system. The maximizing qualities of PHS systems are their energy storage/generation capacity and their minimizing qualities are construction costs. A generalized notion of PHS site productiveness can be expressed as  $\frac{V \times H}{W \times D}$  where  $V$  is the volume of the upper reservoir,  $H$  is the head height of the upper reservoir,  $W$  is the total water-rock ratio (a common measure of construction cost for dams) of the reservoir match, and  $D$  is the horizontal distance between reservoirs in the match. Each of these items has the potential to vary with each reservoir match.

### Balanced vs. Unbalanced Assignment

So far, the problem of finding PHS sites can be formulated as a weighted bipartite, or Assignment problem, for which several efficient algorithms exist to solve. For example, the successive shortest paths algorithm and the Hungarian algorithm are both designed to solve this problem. However, these algorithms work off of the assumption that the bipartite graph is balanced (both sets of nodes are equal in size), which will often not be the case for sets of upper and lower reservoirs.

One option for taking advantage of the bipartite assignment methods is to transform the output of the reservoir model into a balanced bipartite graph by adding a number of artificial nodes needed to make the two sets of equal size. Then, for each of these artificial nodes, create an arc to every node in the other set with 0 weight (the 0 weight ensuring the optimal solution will not consider their connections<sup>[4]</sup>). This transformation increases the input size of the problem, so methods specifically designed for solving unbalanced assignment problems may be more desirable.

# Assignment Representation

The result of all our assumptions on PHS system finding leaves us with a formulation of the most general class of network flow problems, Minimum-Cost Flows. Luckily, many methods are known for tackling Minimum-Cost Flows. Minimum-Cost Flows come with several additional assumptions not covered above, however, such as the need for integral arc costs, directed arcs, all nodes supply/demands sum to  $0$ , an uncapacitated path between every pair of nodes, and that costs are non-negative. None of these assumptions come with major implications for our PHS search when accounted for; PHS weights can easily be rounded and rescaled to be integral while keeping their ranking. PHS weights are non-negative and reservoir demands are  $0$  by default. Transforming a PHS matching into a directed arc does not change its interpretation. And finally, arcs are uncapacitated by default.

## Network Representation

To represent the outputs from the reservoir model as a fitting bipartite network, a separate algorithm must be designed to calculate accurate horizontal and vertical distances between the closest points of all upper and lower reservoirs (this being a computationally expensive task in its own right). Reservoir volume and water-rock ratio are attributed to the reservoirs by the reservoir model. From there, the cost metric stated above will be calculated and placed as weights on arcs.

An intermediate network will be a bipartite network with all upper reservoirs on one layer and all lower reservoirs on the other. It will implement the transformation from an unbalanced to a balanced network if necessary.

For implementation with most common min-cost flow algorithms, artificial source node and sink nodes will be added to the network where arcs emanate from the source node to all nodes in  $U$  and from all nodes in  $L$  to the sink node.

## Solution methods

Not only do network flow algorithms provide efficient ways to solve the Maximum-Weight Bipartite Assignment problem, but they offer time complexity speedups due to their specialized structure.

“Simple” algorithms give straightforward approaches to the problem:

1. Successive Shortest Paths runs in  $O(n_1 S(n, m, C))$  time.
2. Hungarian algorithm runs in  $O(n_1 S(n, m, C))$  time.

More complex algorithms can give actual polynomial time bounds on the algorithm, being useful for worst-case algorithmic planning:

3. Network Simplex Algorithm  $O(V^2 m \log(nC))$  time.
4. Cost Scaling runs in  $O(\sqrt{n_1} m \log(nC))$  time.

The implementation of any of these methods could provide researchers with the final leg of a comprehensive search for PHS systems. A thorough comparison of these pitfalls including their experimental complexities should be performed.

# Conclusion

It is shown here that an automated framework for finding PHS sites given the actual outputs of a previous researcher's model is possible. Once each of the two parts of the process (reservoir finding & reservoir matching) are analyzed for their theoretical and empirical complexities, researchers will have a powerful tool for assessing this technology's potential for utility-scale power.

## Link to GitHub page

Assignment of Pumped Hydro Storage Reservoirs Presentation (<https://github.com/qjoel6398/Assignment-of-Pumped-Hydro-Storage-Reservoirs/blob/master/ProjectPresentation.pdf>)

## References

1. <sup>↑</sup> , Bin Lu, Matthew Stocks, Andrew Blakers, Kirsten Anderson. Geographic information system algorithms to locate prospective sites for pumped hydro energy storage. 2018.
  2. <sup>↑</sup> , Ravinda K. Ahuja, Thomas L. Magnanti, James B. Orlin. Network Flows. 1993.
  3. <sup>↑</sup> , Zvi Galil. Efficient Algorithms for Finding Maximum Matchings in Graphs. 1986.
  4. <sup>↑</sup> , Lyle Ramshaw, Robert E. Tarjan. On Minimum-Cost Assignments in Unbalanced Bipartite Graphs. 2012.
- Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Assignment\\_of\\_Reservoirs\\_for\\_Pumped\\_Hydro\\_Storage\\_Systems&oldid=2905](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Assignment_of_Reservoirs_for_Pumped_Hydro_Storage_Systems&oldid=2905)"

- This page was last modified on 9 May 2020, at 15:39.
- This page has been accessed 2,021 times.

# Burnside's Lemma

From CU Denver Optimization Student Wiki

Welcome to Grace Truong's project page on Burnside's Lemma!

## Abstract/Brief Summary

How would one count the number of colorings a graph would have? Some would consider just saying that there are  $k^n$  ways to color a graph where  $k$  is the number of colors and  $n$  is the number of objects being colored (whether they are edges or vertices). While this may be true, how do we factor in the fact that some of these graph colorings could be equivalent to each other? How would we count the number of colorings a graph has when considering equivalent classes?

To clearly answer this problem, we would first need to define a few things. Firstly, a group is a set  $G$  together with a compositions law satisfying the four properties: identity, inverse, closure, and associativity. We are going to have all permutations of the graph be a group. Next, we will define that two colorings are equivalent to each other when there exists a permutation,  $\pi$ , from our group  $G$  such that  $\pi$  turns one coloring into another. In other words, one coloring assigns to  $\pi(x)$  the same color that the other coloring assigns to  $x$ . This means that to count the number of colorings a graph has when considering permutations would mean we would have to count the equivalence classes.

A good way to do that is to introduce another concept, orbits. Considering  $G$  as a group of permutations on a set  $C$  now, the orbit of an object  $u$  is  $\pi(u) : \pi \in G$ . This implies that the orbits partition  $C$  and therefore any coloring that is in the same orbit is an equivalence relation. So all we need to do now is count the number of orbits to get the number of distinct colorings.

We can count the orbits by summing the number taking  $f$  to each element of its orbit but this would count the orbits  $|G|$  times so we would need to divide by  $|G|$  to count it once. This can be tedious so we introduce a lemma to make the computation simpler.

Lemma: If  $G$  is a group of permutations of  $C$ , and  $u, v \in C$  are in the same orbit, then  $|\{\pi \in G : \pi(u) = v\}| = |\{\pi \in G : \pi(v) = v\}|$

Using this lemma we can get Burnside's lemma Under the action of a group  $G$  of permutations of  $C$ , the number of orbits of  $C$  is  $\frac{1}{|G|} \sum \psi(\pi)$ , where  $\psi$  is the number of elements of  $C$  left fixed by the action of  $\pi$  on  $C$ .

## GitHub Link

[1] (<https://github.com/Grace-Truong/Burnside-s-Lemma>) Can find my presentation slides in the GitHub.

## Bibliography

Combinatorial Mathematics. 1st ed., Cambridge University Press, 2020. (Section 4.2, pages 179-180)

- This page was last modified on 5 May 2025, at 06:53.
- This page has been accessed 33 times.

# Catalan Numbers

From CU Denver Optimization Student Wiki

## Contents

- 1 Contributions
- 2 Introduction
- 3 Applications
  - 3.1 Sequences
  - 3.2 Dyck Paths
  - 3.3 Catalan Number Proof
  - 3.4 Rooted Binary Trees
  - 3.5 Triangulating Polygons
- 4 Github
- 5 Bibliography

## Contributions

The Catalan Numbers are Rachel Snyder's final project for the spring 2025 Graph Theory class. I, Rachel, am the only one who worked on this project.

## Introduction

The Catalan Numbers are a sequence of numbers defined by the formula  $C_n = \frac{1}{n+1} \binom{2n}{n}$ . This means that the first 10 Catalan Numbers are

$$\begin{aligned}
C_0 &= 1 \\
C_1 &= 1 \\
C_2 &= 2 \\
C_3 &= 5 \\
C_4 &= 14 \\
C_5 &= 42 \\
C_6 &= 132 \\
C_7 &= 429 \\
C_8 &= 1430 \\
C_9 &= 4862
\end{aligned}$$

As can be seen, these numbers grow increasingly quickly. The Catalan numbers can also be represented through the recursion

$$C_{n+1} = C_0 C_n + C_1 C_{n-1} + \dots + C_n C_0 = \sum_{i=0}^n C_i C_{n-i}$$

## Applications

The major reason why we like to use the Catalan Numbers is because they are able to count the objects of over 200 different patterns. This allows us to create bijections between these patterns and apply properties that belong to one to the other. The following are a few of my favorite applications

### Sequences

One group of objects that the Catalan numbers count is the number of sequences of length  $2n$  containing only 1s and -1s in which for all  $p \leq 2n$  the sum of the first  $p$  entries is nonnegative and the sum of all  $2n$  elements is 0. Many people often define the Catalan Numbers using this specific example.

This sequence is also sometimes represented using parentheses instead of numbers. In this case, a 1 is equivalent to an open parenthesis, and a -1 is equivalent to a closed parenthesis. Since we are exploring parentheses rather than numbers, instead of requiring nonnegative sums of the first elements, in this version of the sequence, when examining the elements from the beginning, we always must have at least as many open parentheses as we have closed parentheses. By the end of the sequence, there must be the same number of open parentheses as closed ones. We can see that these are equivalent as the sum of our sequence of numbers is equal to 0 if and only if we have the same number of 1s and -1s in our sequence. Further, for the sum of the first elements to be nonnegative, we must always have at least as many 1s as we had -1s.

### Dyck Paths

Counting Dyck Paths is another application of the Catalan Numbers. Dyck paths are the set of paths that travel between integer Cartesian coordinates from point  $(0, 0)$  to point  $(n, n)$ , that do not cross under the line  $y = x$ . Once again, a simple bijection can be made between Dyck paths and the previously discussed sequences. Every time there is a 1 in the sequence of -1s and 1s, have the path travel from point  $(i, j)$  to point  $(i, j + 1)$  and every time there is a -1 in the sequence of -1s and 1s, have the path



travel from point  $(i, j)$  to point  $(i + 1, j)$ . We know that our sequence of 1s and -1s is of length  $2n$  and sums to zero, hence the path travels from  $(0, 0)$  to  $(n, n)$ . Further, since for all  $p \leq 2n$ , the sum of the first  $p$  elements is non-negative, our path never goes further right than it has gone up, therefore never crossing under the line  $y = x$ .

### Catalan Number Proof

Using this interpretation, we will show that these three representations are truly counted by the Catalan number formula, using the proof from the thesis *Enumerations of  $n$ -Permutations Avoiding a Given  $k$ -Permutation* by Rachel Snyder. Given a non-Dyck path,  $D$ , construct a new path  $D'$  such that  $D$  and  $D'$  are identical until the first time that  $D$  crosses the line  $x = y$ . Once  $D$  crosses under the line  $y = x$  for first time, every decision made for the path  $D'$  is the opposite of the decision made for the path  $D$ . To specify, if the line for  $D$  goes to the right, the line for  $D'$  goes up, and if the line for  $D$  goes up, the line for  $D'$  goes to the right. This would therefore be a flip of the second part of  $D$  over the line  $y = x - 1$ .  $D'$  would therefore be a path that goes from  $(0, 0)$  to  $(n + 1, n - 1)$ .

It is clear by construction that every path from  $(0, 0)$  to  $(n + 1, n - 1)$ , must cross over the line  $y = x$ . By reversing the way we constructed our  $D'$  graph from  $D$ , we can create a non-Dyck path transversing from  $(0, 0)$  to  $(n, n)$  given any path from  $(0, 0)$  to  $(n + 1, n - 1)$ . Thus, there is a bijection between paths along a graph from  $(0, 0)$  to  $(n + 1, n - 1)$  and non-Dyck paths from  $(0, 0)$  to  $(n, n)$ .

We can see that there are  $\binom{2n}{n+1} = \frac{2n!}{(n+1)!(n-1)!} = \frac{n}{n+1} \binom{2n}{n}$  such paths since each path has  $2n$  steps,  $n + 1$  of which must be to the right. Thus, there are  $\binom{2n}{n} - \frac{n}{n+1} \binom{2n}{n} = \frac{1}{n+1} \binom{2n}{n}$  Dyck paths, proving that the Catalan numbers are  $\frac{1}{n+1} \binom{2n}{n}$ .

### Rooted Binary Trees

The Catalan Numbers also count the number rooted binary trees with  $n$  internal nodes. An internal node is a non-leaf node. This example is particularly interesting in comparison with the others as we can prove this relation using the recurrence form of the Catalan Numbers. As our base cases, the only tree with 0 internal nodes is a single vertex, and the tree with 1 internal node would look like an upside-down v. In building a tree with  $n + 1$  internal nodes, the root node would be internal and for all  $k$ , we can create unique subtrees with  $k$  internal nodes on the left side and  $n - k$  internal nodes on the right. Putting these three subtrees together, since we have already determined our root node and the sides of our other two subtrees, we have

$$R_{n+1} = \sum_{i=0}^n R_k R_{n-k}$$

options for rooted binary trees with  $n + 1$  internal nodes where  $R_n$  is the number of rooted binary trees with  $n$  internal nodes.

### Triangulating Polygons

Our final example is much more geometric than the others. In this example,  $C_n$  counts the number of ways you can turn an  $(n + 2)$ -gon into  $n$  triangles by adding  $n - 1$  straight lines between the vertices of the  $(n + 2)$ -gon without crossing them. For  $n = 1$ , the shape is already a triangle, so there is one way to triangulate it. There are 2 quadrilateral, as you simply have to choose one of the pairs of nonadjacent vertices. For larger polygons, you can start by drawing a single line between

two of the vertices. If these vertices have  $k$  vertices between them, this splits the polygon into a  $(k + 2)$ -gon and an  $(n - k + 2)$ -gon. Thus, there are

$$T_{n+1} = \sum_{i=0}^n T_k T_{n-k}$$

ways to triangulate an  $(n + 2)$ -gon.

## Github

The link for the github for this project is <https://github.com/snyrache/Catalan-Numbers>. It has my slides.

## Bibliography

“Catalan Numbers - Numberphile.” *www.youtube.com*, 29 Jan. 2024, [www.youtube.com/watch?v=fczN0BCx0xs](https://www.youtube.com/watch?v=fczN0BCx0xs).

“Catalan Numbers | Brilliant Math & Science Wiki.” *Brilliant.org*, [brilliant.org/wiki/catalan-numbers/](https://brilliant.org/wiki/catalan-numbers/). Accessed 2 May 2025.

Weisstein, Eric W. “Catalan Number.” *Mathworld.wolfram.com*, [mathworld.wolfram.com/CatalanNumber.html](https://mathworld.wolfram.com/CatalanNumber.html). Accessed 2 May 2025.  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Catalan\\_Numbers&oldid=5019](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Catalan_Numbers&oldid=5019)"

- 
- This page was last modified on 5 May 2025, at 08:28.
  - This page has been accessed 28 times.

# Chase Viss

From CU Denver Optimization Student Wiki

I am currently a second-year graduate student at the University of Colorado Denver. My mathematical interests include Graph Theory, Combinatorics, and Optimization.

Denver is my hometown; I lived here my entire childhood and graduated from Denver Christian High School in 2011. I then attended Dordt College in Sioux Center, IA, where I started out as an Engineering Major but quickly discovered that my primary interest was Mathematics. Critical thinking and problem solving are two of my strongest passions. While studying Mathematics, I also developed an interest in programming and Computer Science. I aim to continue learning about the many ways computers can be used to advance our mathematical abilities.

In 2015, I moved back to Denver to join Applied Mathematics graduate program at the University of Colorado Denver. I have learned a great amount as a member of this program and hope to continue learning much more. Currently, I am taking an Integer Programming course in which my final project is to research various algorithms that can be used to solve or approximate a classic combinatorial optimization problem known as the Knapsack Problem. More information about this project can be found on the Knapsack Problem Algorithms page.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Chase\\_Viss&oldid=371](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Chase_Viss&oldid=371)"

Category: Contributors

- 
- This page was last modified on 20 April 2017, at 19:21.
  - This page has been accessed 4,228 times.

# Christina Ebben

From CU Denver Optimization Student Wiki

I'm working on my master's degree at the University of Colorado Denver. Studying math is a passion of mine, not purely for math's sake but also as a chance to help people, a chance to bring together the community and, yes, because math is awesome. My ongoing studies in Optimization incorporate a diversity of disciplines and is an enterprise I am most excited for.

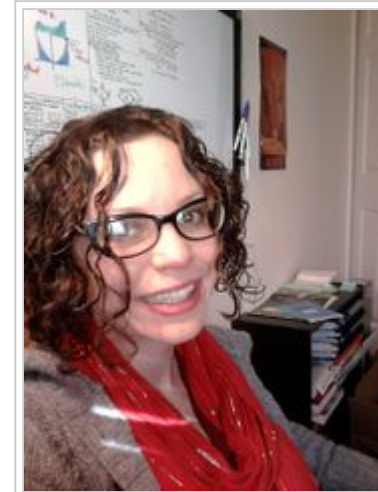
During the weekends I manage a large liquor store and it's here that I get to explore another passion of mine: beer! Craft beer to be exact. I relish the opportunity of being a semi-professional beer geek in a culture as rich and extensive in craft beer as Colorado's. When I'm not working or in class, I'm volunteering at my local library for a program called the Homework Center where I tutor kids in grades 3-12. I also enjoy reading and writing fiction, as well as studying the beautiful language of Portuguese.

Here are my contributions to the Optimization Wiki:

- Lagrangian Duality
- Lagrange Multipliers
- Linear Programming Methods for Radiation Therapy Treatment Planning
- Coordinating Response to Fatal Accidents

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Christina\\_Ebben&oldid=1969](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Christina_Ebben&oldid=1969)"

Category: Contributors



- This page was last modified on 18 April 2019, at 21:34.
- This page has been accessed 7,567 times.

# Circuit Walks and Stable Sets

From CU Denver Optimization Student Wiki

## Contents

- 1 Abstract
- 2 The Project
  - 2.1 The Problem - Maximum Stable Set
  - 2.2 The Greedy Algorithm - Ballard 2019
  - 2.3 The Polytope - The Convex Hull of the Feasible Space
  - 2.4 The Result - Circuit Walks are Used
  - 2.5 The Speculation - A Complete Characterization of Circuits
- 3 Project Author
- 4 Github Repository

## Abstract

### Circuit Walks and Stable Sets

This project considered the Maximum Stable Set Problem (maximum size of a set of vertices in a graph such that no two are adjacent) and the improvement upon the Greedy Algorithm that Ballard made in 2019. This algorithm finds all of the maximal stable sets, then selects a largest one.

The goal of the project is to prove whether or not the algorithm performs a circuit walk within the polyhedron. The feasible space is a clearly defined polyhedron and considers a sequence of feasible solutions. These conditions would suggest such a project can be completed successfully.

The maximal stable sets are found by considering all permutations of vertices, then greedily creating a maximal stable set by adding vertices, in order, when they are not adjacent to any other vertex in the current set. For a graph with  $n$  vertices, the feasible space is  $\{0, 1\}^n$  which contains the incidence vectors corresponding to the power set of the vertex set. It is clear that this algorithm considers a sequence of feasible solutions.

## The Project

We have chosen the Maximum Stable Set Problem. Currently, no known efficient algorithm exists to find the optimal solution. This algorithm is a complete enumeration of the feasible space.

The maximal sets constructed by the algorithm are done using circuit walks. More specifically, they are sign-compatible edge walks. A complete description of all circuits is only speculated at this time.

## The Problem - Maximum Stable Set

Input: A graph  $G(V,E)$

Output: The number of vertices in a maximum stable set. A stable set of vertices is a set of vertices where no two are adjacent.

## The Greedy Algorithm - Ballard 2019

- Select a vertex  $v_1$  and add it to a set  $S_1$ . Take the induced subgraph  $G - N_1$ , where  $N_1$  is the set of  $v_1$  and all vertices adjacent to  $v_1$
- Select a vertex  $v_2$  and add it to  $S_1$ . Take the induced subgraph  $G - N_1 - N_2$ , where  $N_2$  is the set of  $v_2$  and all vertices adjacent to  $v_2$
- Select a vertex  $v_3$  and add it to  $S_1$ . Take the induced subgraph  $G - N_1 - N_2 - N_3$ , where  $N_3$  is the set of  $v_3$  and all vertices adjacent to  $v_3$
- Continue until the induced subgraph is the null graph. Then  $S_1$  is a maximal stable set.
- Return to beginning and repeat for  $S_2, S_3, \dots, S_k$  until all permutations of vertices have been exhausted. Then  $\{S_1, S_2, \dots, S_k\}$  contains all maximal stable sets.
- It follows that the Maximum Stable Set of  $G$  has size  $\max(S_1, S_2, \dots, S_k)$

## The Polytope - The Convex Hull of the Feasible Space

$$\max \sum_{v \in V} x_v \text{ (binary variables)}$$

$$\text{s.t. } x_i + x_j \leq 1 \text{ for every } \{i, j\} \in E$$

## The Result - Circuit Walks are Used

We can see that circuit walks are used by this algorithm to construct the maximal sets. More specifically, we can show that they are edge walks using sign-compatible circuits.

First, since a single vertex is always a stable set, the  $x_i$ 's are always circuits for any arbitrary graph.

Next, since the maximal sets are constructed by adding single vertices at a time to a given stable set, the only steps used are these circuits.

Now, we can see that an empty graph on  $n$  vertices will correspond to a polytope that is an  $n$ -dimensional hypercube. Such a structure has the property that all the edges correspond to the addition or removal of a single vertex from a given stable set. A graph that has edges will correspond to a subset of the empty graph's feasible space, and thus its corresponding polytope is a subset of the  $n$ -dimensional hypercube. But, it still retains the property that the removal or addition of a single vertex (assuming both sets are feasible) will be an edge of the polytope. This is how we can see that not only circuit walks are used, but they are, more specifically, edge walks.

Next, since each of these steps consists of a vector with a single non-zero entry which is one, it is clear that all the steps consist of sign-compatible circuits.

Thus, we can see that the algorithm constructs these maximal stable sets by completing sign-compatible edge walks.

## The Speculation - A Complete Characterization of Circuits

At this time, we speculate that for any natural number  $n$ , there exists a graph that contains the non-zero circuit  $\sum_{v \in V} \alpha_v x_v$  where  $\alpha_v \in \{-1, 0, 1\}$ . The slides on the github repository show all circuits for graphs on three vertices.

## Project Author

Michael Burgher

## Github Repository

[https://github.com/MichaelBurgher/Circuit\\_Walks\\_and\\_Stable\\_Sets.git](https://github.com/MichaelBurgher/Circuit_Walks_and_Stable_Sets.git)

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Circuit\_Walks\_and\_Stable\_Sets&oldid=3586"

- 
- This page was last modified on 7 December 2021, at 19:08.
  - This page has been accessed 1,101 times.

# Circuits and Bloom's Algorithm

From CU Denver Optimization Student Wiki

## Contents

- 1 Abstract
- 2 Introduction
- 3 Problem Formulation
- 4 Constraint Matrices
- 5 Characterization of Circuits
- 6 Overview of Algorithm
- 7 Circuit Walks
- 8 References And Powerpoint

## Abstract

The matching problem is one of the most fundamental graph theory problems, whose application reach out of its discipline. Typically, the goal for a matching problem is to connect as many vertices as possible while ensuring that no vertex is incident to more than one edge. The minimum weight maximal matching is a matching that contains the maximal number of possible edges in any matching for a graph, while having the least total weight for the matching.

The purpose of this project is to study whether a classical algorithm can be interpreted as a circuit walk. We will be examining Bloom's algorithm, and the polytope of the minimum weight maximal matching problem and use these findings to determine if Bloom's algorithm performs a circuit walk over the polyhedron. The algorithm involves looking for vertices in your matching that start odd cycles in your graph. By contracting our odd cycles we will study different M-augmenting paths formed in our smaller graph.

## Introduction

In order to determine if Bloom's Algorithm performs a circuit walk, it is important to first understand what a circuit is, and what a circuit walk is. The formal definition for a circuit is:

**Definition:** The set of circuits  $\mathcal{C}(A, B)$  of a polyhedron  $P = \{x \in \mathbb{R}^n : Ax = b, Bx \leq d\}$  consists of all  $g \in \ker(A) \setminus \{0\}$  normalized to coprime integer components for which  $Bg$  is support-minimal over  $\{Bx : x \in \ker(A) \setminus \{0\}\}$ .



An algorithm can complete a circuit walk by traveling from one vertex of the polyhedron to another along a path which is parallel to an edge of the polyhedron in each step. One of the goals for this project is to determine whether or not Bloom's algorithm forms such a walk. The definition of this algorithm can be found in the Overview of the Algorithm section.

## Problem Formulation

Let  $G$  be a complete graph. Let  $x_{i,j}$  be an edge that connects vertex  $i$  to vertex  $j$  and let  $w_{i,j}$  be the cost of including the edge in our matching. Finally, let  $M$  be a maximal matching in  $G$ . Then, the problem formulation for the minimum weight maximal matching is:

$$\begin{aligned} & \text{minimize: } \sum_{i,j} x_{i,j} \cdot w_{i,j} \\ & \text{such that } \sum_j x_{i,j} \leq 1 \quad \forall i \in G \\ & \sum_{i,j} x_{i,j} = 2|M| \quad \forall i, j \in G \\ & x_{i,j} \in \{0, 1\} \quad \forall i, j \in G \end{aligned}$$

The goal for this problem is to return the matching that is both maximal, but whose total edge weights or minimal. The objective function for this problem returns the total cost of a maximal matching. The first set of constraints ensures that each vertex is incident to at most one edge in the matching. The second set of constraints ensures that there are enough edges in the matching so that it will be maximal. The last set of constraints ensures that each edge will be binary, or, each edge is either entirely included in the matching, or it is not included at all. These constraints prevent fractional amounts of edges from being included in the matching.

## Constraint Matrices

The matrix  $A$  is the constraint matrix that is associated with the set of equality constraints in the problem formulation. For our minimum weight maximal matching problem,  $A$  is a node-arc incidence matrix. The generalization for this matrix  $A$  on a complete graph of any size  $k$  can be seen in Figure 1. The other matrix in our polyhedron,  $B$ , is associated with the inequality constraints of the problem formulation. These constraints are the first set of constraints, which is the node-arc incidence matrix, and the last set of constraints, which is the identity matrix. For a given  $k$ , the matrix  $B_k$  can be seen in Figure 2.

We now show that our recursive definition of  $A_k$  is correct by using induction.

Base Case:  $n = 3$ . The node-arc incidence matrix for  $K_3$  is  $A_3$  which can be seen in Figure 3. The node-arc incidence matrix for  $K_2$  is  $A_2 = [11]^T$ . By inspection this

Induction step: Assume this is true for  $3 \leq k \leq n - 1$ . We show this is true for  $k = n$  by first proving a different claim.

Claim 1:  $K_{n-1}$  is an induced subgraph of  $K_n$ .

Proof of Claim: Let  $G$  be a complete graph with  $n$  vertices. Let  $G'$  be any induced subgraph. Let  $u, v$  be vertices of  $G'$ . Since  $G$  is a complete graph, there exist an edge between  $u$  and  $v$  in  $G$ . Thus there exist that same edge in the induced subgraph  $G'$ . Since  $u, v$  are arbitrary this works for any pair of vertices. Thus, there is an edge between any pair of vertices in  $G'$ , which implies that  $G'$  is a complete graph.

This claim also gives us the corollary that every incidence matrix of  $K_n$  will have the incidence matrix of  $K_{n-1}$  as a submatrix. This result can be found in the lower right block of our matrix. The 1 block and the 0 block of our matrix is produced by taking our  $K_{n-1}$  graph and adding a vertex  $v_1$ . In order to make the new graph  $K_n$  we add edges from all other vertices to  $v_1$ . We then relabel the edges in such a way that all edges connected to  $v_1$  are  $e_1 \dots e_{n-1}$ . This creates the upper row of our matrix.

The lower left identity block is exists since each new edge that is added to  $v_1$  for  $K_n$  has to be attached to  $v_1$  and some other vector  $v_i$ . If we label our vertices in a way such that  $e_i$  is connected to both  $v_i$  and  $v_{i+1}$ , then we will get the identity matrix for our lower left block. Thus, we have showed this formulation is true for  $K_n$  which completes the proof.

For any vector to be circuit, they must also for them to be support minimal over  $B_k$ . Fortunately, the  $B_k$  matrix for our constraints has an identity on the bottom  $k$  rows. Therefore, if we have a set of circuits that are linearly independent, they will be support minimal over  $B_k$ . In order to fully characterize the integral circuits, we only need to find a basis of the kernel of  $k$ .

We now show that the number of integral circuits, up to scalar multiplicity, is  $\binom{k}{2} - k$ . This argument is based on the rank nullity theorem. Since the circuits exist in the kernel of  $\binom{k}{2} - k$ , and we can show that the rank of our matrix is  $k$  then we know the number of vectors in our kernel and therefore the number of circuits. Since A is a non square matrix the  $rank(A) \leq min[k, \binom{k}{2}]$ . the number of vertices, k will always be our minimum value. So we know our rank will be at most k, and what we will argue is that it will be k exactly. The first k-1 column vectors are linearly independent because that is just a 1 vector on top of the identity matrix. The kth column will also be linearly independent. This is because it will always be of the form  $[011\dots]^T$ , but the only way to may the first coordinate a 0 and the second and third coordinate both be nonzero would be to subtract  $v_1$  and  $v_2$ . But that would make the second component a 1 and the third component a -1 which will not be of the form of  $[011\dots]^T$ . Thus that kth vector will be linearly independent. So our rank is exactly k which makes our nullspace have dimension  $\binom{k}{2} - k$ . Thus there are  $\binom{k}{2} - k$  circuits since for our formulation every vector in the kernel is a circuit.

## Characterization of Circuits

Since the constraint matrices for our problem have been established, we will use them to find the circuits for our problem. As previously mentioned, in order for a vector to be a circuit, the vector  $v$  must exist in the kernel of  $A_k$  and be support minimal over  $B_kv$ . Due to the structure of  $B_k$ , if we find a basis for the kernel of  $A_k$ , we will find the spanning set of circuits. For any complete graph of size  $k$ , there will be a basis with  $k - 3$  vectors of the form  $[0|I_{k-3}| - 1| - I_{k-3}|1|0]^T$  that exist in the kernel of  $A$ .

$$A_k = \left[ \begin{array}{c|c} 1 & \cdots & 1 & 0 \\ \hline & I_{k-1} & & A_{k-1} \end{array} \right]$$

**Figure 1:** Constr. Matrix A

$$B_k = \left[ \begin{array}{c} A_k \\ \hline I_{\binom{k}{2}-k} \end{array} \right]$$

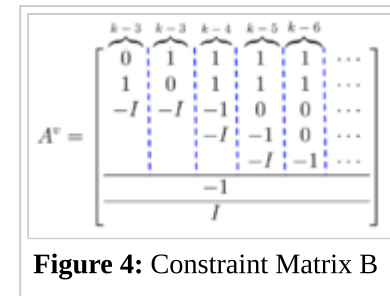
**Figure 2:** Constraint Matrix B

$$A_3 = \left[ \begin{array}{ccc} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{array} \right]$$

**Figure 3:** Constraint Matrix B

The remaining vectors in this basis follow a similar pattern. These remaining vectors can be seen in Figure 4.

What remains to be shown is that the characterization of circuits given in Figure 4 actually exists in the kernel of  $A_k$ . For the sake of brevity, we only show that the first  $k - 3$  vectors in this characterization exist in the kernel of  $A_k$ . The first  $k - 1$  entries for each of the  $k - 3$  vectors will contain exactly 2 non-zero entries, one that is +1, and a one that is -1. Thus, for these  $k - 3$  vectors, the product of each of these vectors and the first row of  $A_k$  will be zero. For the next row of the matrix  $A_k$ , the only non-zero entries that exist are in the next  $k - 2$  entries, and in the first entry of the row. The next  $k - 2$  entries for each of the first  $k - 3$  vectors  $v$  in our characterization will contain two non-zero values, one that is +1, and one that is -1. Additionally, the first entry of each of these vectors is zero, so, for these rows of  $A_k$ , the product of  $A_k$  and  $v$  will be zero. In the remaining rows of  $A_k$ , for any of the vectors  $v$  in the first  $k - 3$  vectors in our characterization,  $A_k v = [0 \cdots 0, 1 \cdot v_{k+i}, 0 \cdots 0, 1 \cdot v_i, 0 \cdots 0]$  where  $i$  is associated with the  $i - 1$  vector in our characterization. The reason that the entries  $1 \cdot v_{k+i}$  and  $1 \cdot v_i$  are the only non-zero values in  $A_k v$  for these rows of  $A_k v$  is that these are the only non-zero values that occur in the same index for both of these rows of  $A_k$  and  $v$ . In these cases,  $v_{k+i} = 1$  and  $v_i = -1$ . Thus, for the remaining rows, the sum of the entries for  $A_k$  will be zero. Therefore, each of the first  $k - 3$  vectors in our characterization exist in the kernel of  $A_k$ , and hence, are circuits of our problem.



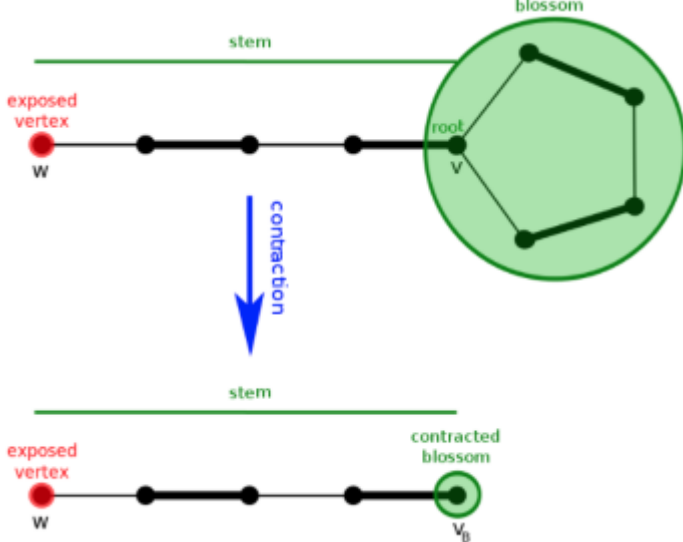
**Figure 4:** Constraint Matrix B

## Overview of Algorithm

We will be looking at two main algorithms. One is the Maximal Weight Matching Algorithm by Lovasz and Plummer. This will be the overarching algorithm that will find us feasible solutions to our problem. The other algorithm will be the Blossom Algorithm by Jack Edmonds. This algorithm is going to be a subroutine of our Maximal Weight Matching Algorithm. We will be focusing on determining if the Blossom Algorithm performs a circuit walk.

For Lovasz's and Plummer's algorithm we are going to start with a general graph with weighted edges. If that graph has an odd number of vertices we are going to add an isolated vertex so that our graph will have an even number of vertices. This will help us in making a perfect matching. The next step in our algorithm is to connect all non-adjacent vertices with an edge of weight zero. During this process it is important to not add any additional weight to our graph otherwise we would end up changing our initial problem. We then extend all of our matchings to perfect matchings. This is possible because we have an even number of vertices. We then replace all of our positive weights with there negative weight. This turns the problem into a minimum weight perfect matching problem. We then run our Blossom algorithm as a subroutine.

The Blossom Algorithm is an algorithm that involves contracting down cycles to find a M-augmenting path in our graph. We call a vertex exposed if no edge in our matching is adjacent to that vertex. The algorithm starts at an exposed vertex and then traverses along the graph. Let's label our exposed vertex as an outer vertex "o". The algorithm will then alternate between labeling each vertex as an outer vertex or an inner vertex. The algorithm continues along our graph until two vertices are adjacent and both labeled with as an outer vertex as can be seen in the figure below. If this happens, we have an odd cycle. Let's call this odd cycle a blossom. The algorithm will the contract our blossom and then repeat the above process. Once all blossoms have been contract, the algorithm check to see if there exist an M-augmenting path in the new contracted graph. If there exist no M-augmenting path in the new graph, then the algorithm has found a maximum matching.



## Circuit Walks

Now that we have established what the circuits for our formulation are, and how the algorithm which we are using on our problem actually works, we move into determining whether our algorithm performs a circuit walk on our polyhedra. Let's analyze what we know step by step. The vertices of our polyhedra are maximal matchings by our formulation of the matching polytope. Our algorithm takes a matching and preforms M-alternating paths to get rid of any M-augmenting path. Once we have no M-augmenting path in our graph, we have found a maximal matching. Since we are working on a complete graph our algorithm will take us from a maximal matching of size M to another maximal matching of size M. If we apply our circuit to a specific vertex we will go from one maximal matching (vertex) to another maximal matching (vertex). Thus, our algorithm will have completed a circuit walk.

Let's look at a specific example. If we take  $K_4$ , then our matrix  $A_4$  can be seen in Figure 5. The circuits are of our matrix are seen in Figure 6. Let's analyze  $v_1$ . We start with a matching corresponded to the -1's in our circuits. We then remove those edges from our maximal matching and add the edges corresponding to a 1. This gives us a new maximal matching. (Figures 7 and 8) Notice that in our circuits the number of 1's and -1's are exactly the same. This is because for every edge we remove in our matching we must add in the same number of edges back if we are to move from a maximal matching of size M to another maximal matching of size M. So, we have moved in a circuit direction from one maximal matching (vertex) to another maximal matching (vertex). This is the definition of a circuit walk. It is important to note that the -1's in our circuit won't necessarily be a maximal matching, but they are equivalent to the edges that are removed during any step in a circuit walk.

$$A_4 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

**Figure 5:** Matching after applying Circuit

$$v_1 = [0 \quad 1 \quad -1 \quad -1 \quad 1 \quad 0]^T$$

$$v_2 = [1 \quad 0 \quad -1 \quad -1 \quad 0 \quad 1]^T$$

**Figure 6:** Matching after applying Circuit

## References And Powerpoint

You can access the powerpoint of our presentation here:

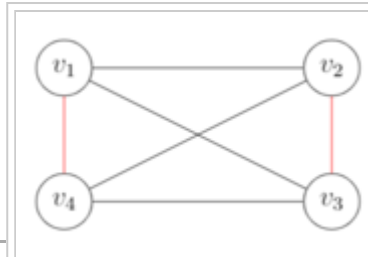
[https://github.com/toadtoad44/Matching-Powerpoint/blob/main/The%20Circuits%20of%20Matchings%20\(2\).pptx](https://github.com/toadtoad44/Matching-Powerpoint/blob/main/The%20Circuits%20of%20Matchings%20(2).pptx)

Lovász, László, and Michael D. Plummer. Matching Theory, North-Holland, Amsterdam, 1986, pp. 357–382.

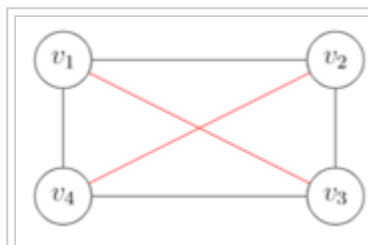
West, Douglas B. Graph Theory: Introduction to, Second Edition. Prentice Hall, 2001.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Circuits\\_and\\_Bloom%27s\\_Algorithm&oldid=3605](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Circuits_and_Bloom%27s_Algorithm&oldid=3605)"

- This page was last modified on 10 December 2021, at 20:02.
- This page has been accessed 1,847 times.



**Figure 7:** Original Matching



**Figure 8:** Matching after applying Circuit

# Circuit Analysis of the Maximum Clique Problem

From CU Denver Optimization Student Wiki

This Project is by Alyssa Newman

## Contents

- 1 Abstract
- 2 Background
  - 2.1 Maximum Cliques
  - 2.2 Polytope of the maximum clique problem
  - 2.3 An iterative algorithm for the maximum clique problem
- 3 The greedy algorithm over the polyhedron
- 4 Other circuits of the polytope
  - 4.1 Justification of other circuits existing
  - 4.2 Proposal of an additional circuit characterization
  - 4.3 Further work
- 5 Github Repository
- 6 references

## Abstract

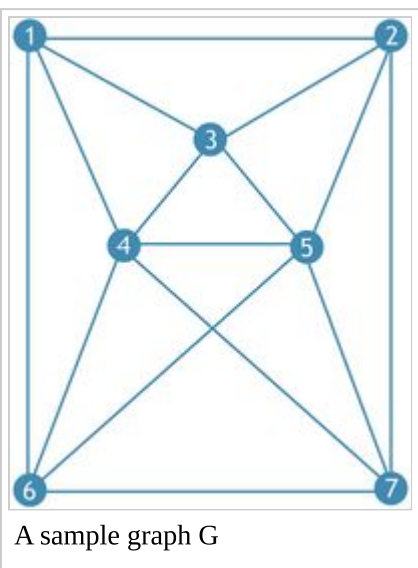
Circuit Analysis of the Maximum Clique Problem. I am analyzing the greedy algorithm on the polytope of the maximum clique problem. The maximum clique problem is the problem of trying to find the maximum clique, or complete subgraph, of any given graph. In the greedy algorithm, we move from a set of vertices of the graph that form a clique and add any other vertex that maintains the set forming a clique. It is well known that the polytope for the maximal clique problem is defined by it's vertices which are the incidence vectors of all cliques that appear in the graph. Combining what we know about the greedy algorithm and the V representation of the polytope it is clear that the greedy algorithm walks between vertices in the polytope, what this project aims to answer is if this walk amongst vertices is a circuit walk of the polytope.

## Background

### Maximum Cliques

In graph theory, a clique for a graph  $G$  is a subset of vertices of the graph that are all connected, or otherwise that that collection of vertices form a complete graph. It is standard to assume that a singleton point is a clique. For example, the graph below has many cliques including the set of vertices:  $456, 12, 345, 7$  and more. For a graph  $G$  represented as an incidence or  $0, 1$  vector that indicates if each vector is in the clique. For example the clique  $456$  will be represented as the incidence vector

$[0, 0, 0, 1, 1, 1, 0]$ . A maximum clique is the largest possible clique that is in any given graph, the size of the maximum clique of a graph  $G$  is often represented as  $\omega(G)$ . For the graph above the maximum clique can be seen to be 4, 5, 6, 7 or the incidence vector  $[0, 0, 0, 1, 1, 1, 1]$ , so the graph has  $\omega(G) = 4$ . The maximum clique problem is the process of finding the maximum clique in a graph. However this problem is know to be NP-complete. Just the process of checking is a graph has a clique larger than some given size is NP-complete.



## Polytope of the maximum clique problem

The problem of finding the maximum clique is a linear optimization problem and therefore the feasible set forms a polytope. The LP representation of the maximum clique problem is

$$\begin{aligned} \max \quad & \sum_{v \in V} v_i \\ \text{subject to} \quad & v_i + v_j \leq 1, \forall (i, j) \notin E \\ & v_i \in \{0, 1\} \forall i \end{aligned}$$

Which gives the H representation. The set of incidence vectors for all cliques will form the V representation of the polytope of the maximum clique problem.

## An iterative algorithm for the maximum clique problem

This project looks at the greedy algorithm on the graph. This algorithm does not guarantee the maximum clique but it will find a maximal clique. The greedy algorithm starts with any vertex of the graph. And then adds the first vertex it can while maintaining the set forming a clique. This is repeated until no new vertices can be added. Using the same graph as above we can start the greedy algorithm with vertex 1 or at the incidence vector  $[1, 0, 0, 0, 0, 0, 0]$ . We then look at vertex 2 which is connected to everything currently collected so it is added. We now have the set 1, 2 or the incidence vector  $[1, 1, 0, 0, 0, 0, 0]$ . We then look at vertex 3 which is connected to everything currently collected so it is added. We now have the set 1, 2, 3 or the incidence vector  $[1, 1, 1, 0, 0, 0, 0]$ . However now when we look at the rest of the vertices of the graph none of them are connected to all of the collected vertices and the algorithm then ends. In this example, the algorithm finds a maximal clique of 1, 2, 3 but recall

# The greedy algorithm over the polyhedron

We have established that at the general  $i^{\text{th}}$  step in the greedy algorithm results in a clique in the graph, and that the vertices of the polyhedron are the cliques in the graph and thus at the  $i^{\text{th}}$  step in the greedy algorithm we are at a vertex of the polyhedron. We now want to show that these vertices of the polyhedron are adjacent in the polyhedron. Before we show this we establish the following notation. Assume we have a graph  $G$  with  $n$  vertices. Assume the greedy algorithm returns a maximal clique of size  $m$ , label this first  $m$  vertices in the order they were chosen and the remainder randomly to get the vertices  $v_1, \dots, v_m, v_{m+1}, \dots, v_n$ . We will be calling an incidence vector  $w_k$  where it has the first  $k$  components being a 1 and the rest of the components being a 0. In general the  $i^{\text{th}}$  step of the greedy algorithm moves from the collection  $\{v_1, \dots, v_i\}$  which has incidence vector  $w_i$  to  $\{v_1, \dots, v_i, v_{i+1}\}$  which has incidence vector  $w_{i+1}$ . To show that  $w_i$  and  $w_{i+1}$  are adjacent in the polytope we will use the fact that two vertices are adjacent if and only if their midpoint can be uniquely represented as a convex combination of vertices in the polytope. Note that the midpoint of  $w_i$  and  $w_{i+1}$  is  $\frac{1}{2}(w_1 + w_{i+1})$  which will give the incidence vector with the first  $i$  components being 1, the  $i + 1$  component being  $\frac{1}{2}$  and the rest being 0. This can be represented at the convex combination of the vertices as  $0w_1 + \dots + 0w_{i-1} + \frac{1}{2}w_i + \frac{1}{2}w_{i+1} + 0w_{i+2} + \dots$ . Assume for the sake of contradiction that there is some other convex combination  $a_1w_1 + a_2w_2 + \dots = \frac{1}{2}(w_1 + w_{i+1})$ . Since this is a convex combination we know that  $a_j \geq 0 \forall j$  and  $\sum_j a_j = 1$ . We also know that  $w_j$ 's are incidence vectors and therefore all of their components are all 0 or 1. Since we know that  $\sum_j a_j w_j$  must be equal to 1 for all of its first  $i$  components and that means for each  $j$  such that  $a_j \neq 0$  the  $i^{\text{th}}$  component of  $w_j$  must be equal to 1. Similarly we know that the  $i + 2, \dots, n$  components of the sum must come out to be 0 thus for each  $j$  such that  $a_j \neq 0$  the  $i + 2, \dots, n$  component of  $w_j$  must be equal to 0. Thus of all vertices with non-zero coefficients only the  $i + 1$  component can change. Therefore the only two possible vertices with non-zero coefficients are  $w_i$  and  $w_{i+1}$  therefore the combination is the same.  $\Rightarrow \Leftarrow$  This now gives us the conclusion that  $w_i$  and  $w_{i+1}$  are adjacent vertices in the polytope and therefore  $w_{i+1} - w_i$  is a circuit direction in the polytope. Since this was done for an arbitrary step in the greedy algorithm this conclusion holds for each step in the greedy algorithm. Therefore the greedy algorithm a circuit walk in the polytope, and more specifically an edge walk. Note that the circuit directions for each of these steps will have the form  $(0, \dots, 0, 1, 0, \dots, 0)$  where the 1 is in the  $i + 1$  component. Therefore the circuits directions walked are sign compatible.

## Other circuits of the polytope

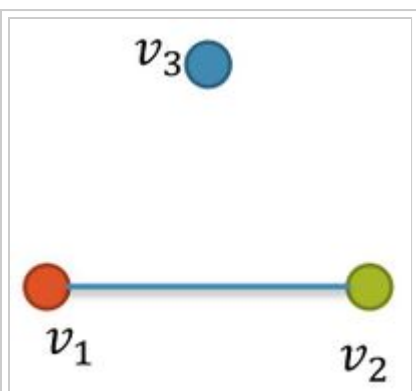
While this does give us many circuits in the polytope for the maximal clique problem this is not enough to characterize all circuits in the polytope. To do this we will be using the LP formulation of the maximal clique problem which is listed below.

$$\begin{aligned} & \max \sum_{v \in V} v_i \\ & \text{subject to} \\ & v_i + v_j \leq 1, \forall (i, j) \notin E \\ & v_i \in \{0, 1\} \forall i \end{aligned}$$

before finding other support minimal circuit directs we first note that there is no  $A$  matrix and therefore  $\ker A = \mathbb{R}^n$  and therefore we only need circuit directions to be support minimal with respect to  $B$



## Justification of other circuits existing



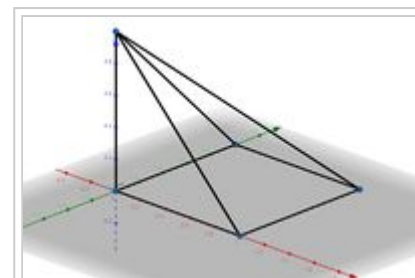
The graph of a small example to justify other circuits existing

Before attempting to find and other circuit directions we want to confirm that there are definitively other circuit directions. Take for example the graph on three vertices  $\{v_1, v_2, v_3\}$  with an edge between  $v_1$  and  $v_2$  and no other edges as drawn to the left. The 3 dimensional image of the polytope can be seen in the image to the right. Then the  $B$  matrix is

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

And the circuit directions are

$\pm\{(1, 0, 0), (0, 1, 0), (0, 0, 1), (1, -1, 0), (0, 1, -1), (1, 1, -1)\}$ . This clearly has other circuit directions that would not be traversed by the greedy algorithm.



The polytope of the example graph

## Proposal of an additional circuit characterization

For an underlying graph  $G$  with vertices  $v_1, \dots, v_n$  with corresponding incidence vectors  $u_i \forall i = 1, \dots, n$  where the  $i^{th}$  component is 1 and the rest are 0. Note  $u_i$  is a vertex in the polytope since all singleton sets are cliques. Further  $\pm u_i$  is a circuit direction that can be traversed through the greedy algorithm. We propose that there is also a circuit  $u_i - u_j \forall (v_i, v_j) \notin E(G)$ .

To justify this claim we first look at the support of  $u_i - u_j$  with respect to  $B$ . For any arbitrary row of  $B$  we will look at four cases. For each of these cases this arbitrary row of  $B$  will be called  $B_k$ .

Case 1: The  $i^{th}$  and  $j^{th}$  indices of  $B_k$  are both 1. In this case we would have that  $B_k(u_i - u_j)$  will come out to be  $0 + \dots + 0 + 1 + 0 + \dots + 0 + -1 + 0 + \dots + 0 = 0$  where the 1 is in the  $i^{th}$  sum and the -1 is in the  $j^{th}$  sum.

Case 2: The  $i^{th}$  and  $j^{th}$  indices of  $B_k$  are both 0. In this case we would have that  $B_k(u_i - u_j)$  will come out to be  $0 + \dots + 0 + 0 + 0 + \dots + 0 + 0 + 0 + \dots + 0 = 0$ .

Case 3: The  $i^{th}$  index is 1 and the  $j^{th}$  index is 0. In this case we would have that  $B_k(u_i - u_j)$  will come out to be  $0 + \dots + 0 + 1 + 0 + \dots + 0 + 0 + 0 + \dots + 0 = 1$  where the 1 is in the  $i^{th}$  sum.

Case 4: The  $i^{th}$  index is 0 and the  $j^{th}$  index is 1. In this case we would have that  $B_k(u_i - u_j)$  will come out to be  $0 + \dots + 0 + 0 + 0 + \dots + 0 + -1 + 0 + \dots + 0 = 0$  where the -1 is in the  $j^{th}$  sum.

Therefor the support of  $u_i - u_j$  is all rows of  $B$  that are the  $i^{th}$  or  $j^{th}$  row of the identity submatrix or that indicate a non-edge  $(v_i, v_\ell)$  or  $(v_j, v_\ell)$

To show that the support is inclusion minimal we will try and remove a support and find that this would mean adding a separate support. If we try to remove a support that comes from a non-edge  $(v_j, v_\ell)$ . The first way we could remove this support is by removing  $u_j$ , if we do that then when we would be looking at the support of  $B(w_i)$  we now have to add the row of  $B$  corresponding to the non-edge  $(v_i, v_j)$  which was not previously a support. The other way we could remove this support is by adding  $u_\ell$  and then we would be looking at the support of  $B(w_i - w_j + w_\ell)$  however this would add the  $\ell^{th}$  row of the identity submatrix of  $B$  to the support. The other support we

could try and remove would be the support from the  $j^{th}$  row of the identity submatrix of  $B$ . To would have to again remove  $u_j$ , if we do that then when we would be looking at the support of  $B(w_i)$  we now have to add the row of  $B$  corresponding to the non-edge  $(v_i, v_j)$  which was not previously a support. From this, it looks seems as though this is in fact support minimal and thus it would be a circuit direction.

## Further work

The next step in this project would be to further characterize more circuits and work on verifying our next proposal that  $\sum_{v_i \in \omega} u_i - u_j$  where  $\omega$  is a clique in  $G$  and  $v_j$  is disjoint from that clique. Which is a generalization of the previous proposal. It is further hypothesized that there is a further generalization of what is subtracted to give a full characterization of the circuits but what that generalization might be has not yet been determined.

## Github Repository

<https://github.com/ANewman94/Circuit-Analysis-of-the-Maximum-Clique-Problem>

## references

Korte, Bernhard, and Jens Vygen. Combinatorial Optimization: Theory and Algorithms. Springer, 2018.

De Simone, Caterina, and Mosca, Raffaele. Stable set and clique polytopes of (P5,gem)-free graphs. Discrete Mathematics, Volume 307, Issue 22,2007,Pages 2661-2670,ISSN 0012-365X, <https://doi.org/10.1016/j.disc.2007.01.010>.

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Circuit\_Analysis\_of\_the\_Maximum\_Clique\_Problem&oldid=3621"

- This page was last modified on 22 March 2022, at 17:03.
- This page has been accessed 518 times.

# Class Scheduling

From CU Denver Optimization Student Wiki

## Contents

- 1 Class Scheduling through Linear Programming
- 2 Abstract
- 3 AMPL Code for Model
- 4 Solution
- 5 Extensions

## Class Scheduling through Linear Programming

by Amber Rosacker

I worked on trying to create a very basic model for class scheduling using linear programming for teachers as my final project for Linear Programming, Fall 2019. The final presentation slides can be found in the GitHub repository below.

### Abstract

Class scheduling is a process that every school goes through every year. They have to determine which teacher, teaches what classes, and at what time with multiple constraints to take into account. This is a huge undertaking for one individual, and it would be beneficial to have a linear program that could quickly do this process. Currently, in my school, one administrator sends out a form that asks us to rank our top 5 classes we would like to teach from 1 to 5, 1 being the highest. It also asks us to state which plan-periods we would like to have, and any other additional things we would like her to take into account when building the schedule. She then takes all this information and determines the number of sections needed for each class based upon students' schedules. She then builds a huge board using magnets that assign a teacher to all of their classes and the corresponding time periods. This process takes her about a semester and is very time-consuming. The goal of my project is to create a very simple teacher scheduling model that assigns 5 teachers to their 5 classes that must be taught during different times periods. A future goal, with a lot more work and knowledge, would be to make this project something that could actually be utilized in the schedule-making process in schools.

### AMPL Code for Model

Please see my code and data for this problem in the following GitHub repository.

<https://github.com/ARosacker/Class-Scheduling->



[McIlroy,*,*]								[Rosacker,*,*]								[Smith,*,*]										
:	1st	2nd	3rd	4th	5th	6th	7th	:=	:	1st	2nd	3rd	4th	5th	6th	7th	:=	:	1st	2nd	3rd	4th	5th	6th	7th	:=
Algebra1	1	0	0	0	0	0	0		Algebra1	0	0	0	0	0	0	0		Algebra1	0	0	0	0	0	0	0	
Algebra1a	0	0	0	0	0	0	1		Algebra1a	0	0	0	0	0	0	0		Algebra1a	0	0	0	0	0	0	0	
Algebra1b	0	0	0	0	0	0	0		Algebra1b	0	0	0	0	0	0	0		Algebra1b	0	0	0	0	0	0	0	
Algebra1c	0	0	0	0	0	0	0		Algebra1c	0	0	0	0	0	0	0		Algebra1c	0	0	0	0	0	0	0	
Algebra1d	0	0	0	0	0	0	0		Algebra1d	0	0	0	0	0	0	0		Algebra1d	0	0	0	0	0	0	0	
Algebra1e	0	0	0	0	0	0	0		Algebra1e	0	0	0	0	0	0	0		Algebra1e	0	0	0	0	0	0	0	
Algebra2	0	0	0	0	0	0	0		Algebra2	0	0	1	0	0	0	0		Algebra2	0	0	0	0	0	0	0	
Algebra2a	0	0	0	0	0	0	0		Algebra2a	0	1	0	0	0	0	0		Algebra2a	0	0	0	0	0	0	0	
Algebra2b	0	0	0	0	0	0	0		Algebra2b	0	0	0	0	0	0	0		Algebra2b	0	0	0	0	0	0	0	
Algebra2c	0	0	0	0	0	0	0		Algebra2c	0	0	0	0	0	0	0		Algebra2c	0	0	0	0	0	0	0	
Algebra2d	0	0	0	0	0	0	0		Algebra2d	0	0	0	0	0	0	0		Algebra2d	0	0	0	0	0	0	0	
CollegeAlg	0	0	0	0	0	0	0		CollegeAlg	0	0	0	0	0	0	0		CollegeAlg	0	0	0	0	0	0	0	
CollegeAlga	0	0	0	0	0	0	0		CollegeAlga	0	0	0	0	0	0	0		CollegeAlga	0	1	0	0	0	0	0	
CollegeAlgb	0	0	0	0	0	0	0		CollegeAlgb	0	0	0	0	0	0	0		CollegeAlgb	0	0	0	0	0	0	1	
CollegeAlgc	0	0	0	0	0	0	0		CollegeAlgc	0	0	0	0	0	0	0		CollegeAlgc	0	0	1	0	0	0	0	
Geometry	0	0	0	0	0	0	0		Geometry	1	0	0	0	0	0	0		Geometry	0	0	0	0	0	0	0	
Geometrya	0	0	0	0	0	0	0		Geometrya	0	0	0	0	0	0	1		Geometrya	0	0	0	0	0	0	0	
Geometryb	0	0	0	0	0	0	0		Geometryb	0	0	0	0	1	0	0		Geometryb	0	0	0	0	0	0	0	
Geometryc	0	0	0	0	0	0	0		Geometryc	0	0	0	0	0	0	0		Geometryc	1	0	0	0	0	0	0	
Geometryd	0	0	0	0	0	0	0		Geometryd	0	0	0	0	0	0	0		Geometryd	0	0	0	0	1	0	0	
Geometrye	0	0	0	0	0	0	0		Geometrye	0	0	0	0	0	0	0		Geometrye	0	0	0	0	0	0	0	
Trig	0	0	0	0	0	0	0		Trig	0	0	0	0	0	0	0		Trig	0	0	0	0	0	0	0	
Triga	0	0	0	0	1	0	0		Triga	0	0	0	0	0	0	0		Triga	0	0	0	0	0	0	0	
Trigb	0	1	0	0	0	0	0		Trigb	0	0	0	0	0	0	0		Trigb	0	0	0	0	0	0	0	
Trigc	0	0	0	0	0	1	0		Trigc	0	0	0	0	0	0	0		Trigc	0	0	0	0	0	0	0	

I then ran this program as a linear program and not a binary program, and this still assigned the same teacher to the same classes to teach but assigned them to different times throughout the day. These results can be seen in my presentation slides below.

## Extensions

A future goal of this project is to make it an integer program that can actually be utilized in the class scheduling process in schools. In order to make this happen there are many more constraints that would need to be added to this model. Things to be added:

- Create a number of sections for each course (not list out Algebra1a, Algebra1b, etc.)
- Extend this to 15 teachers so one department can get a full schedule with the program
- Only 3 different types of classes per teacher maximum
- Only 4 maximum teachers per type of class (keeps PLCs small)
- There needs to be one of each section for each course during different time periods (all Algebra 1's can't be 1st period)
- Need to have one plan-period (no class) on an even class period, and the second plan-period on an odd class period.
- And so many more...

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Class\\_Scheduling&oldid=2475](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Class_Scheduling&oldid=2475)"

- This page was last modified on 3 December 2019, at 20:28.
- This page has been accessed 2,203 times.

# Claws and Effect: Finding Strategies to Increase Pet Adoption

From CU Denver Optimization Student Wiki



## Abstract

According to the American Society for the Prevention of Cruelty to Animals (ASPCA) around 6.3 million animals enter shelters each year in the United States, and of those around 920,000 are euthanized. This number has decreased drastically from 2.6 million per year in 2011, and this decline in euthanization can be partially explained by an increase in the percentage of animals adopted.

At the beginning of the ongoing covid-19 pandemic, shelters were seeing an immense increase in pet adoption, so much so that some shelters and rescues were empty. While the pandemic has been disastrous for numerous reasons, it has revealed valuable information on adoption rates, such as which areas have the most demand for pets. We utilize this information to help us distribute pets between facilities in such a way to better satisfy this demand and ensure we are maximizing the amount of pets being placed in permanent homes. We recommend policy makers consider incorporating this strategy for increasing adoption through redistribution of pets into the The Pet Animal Care Facilities Act (PACFA) Program, which is dedicated to protecting the well-being of pet animals in facilities throughout Colorado.

## Links to Code and Slides

The python code, data files, and final slide deck, can be accessed through GitHub here: [https://github.com/drewhort/claws\\_and\\_effect](https://github.com/drewhort/claws_and_effect)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Claws\\_and\\_Effect:\\_Finding\\_Strategies\\_to\\_Increase\\_Pet\\_Adoption&oldid=3535](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Claws_and_Effect:_Finding_Strategies_to_Increase_Pet_Adoption&oldid=3535)"

- This page was last modified on 7 December 2021, at 12:05.
- This page has been accessed 352 times.

# Cleaning Parks for a Safer Future

From CU Denver Optimization Student Wiki

The Traveling Salesman Problem (TSP) is one of the most famous integer programming algorithm. The concept is simple, find the quickest/cheapest/shortest path to visit a certain amount of locations. The solutions become far from simple, especially when the amount of locations increases to larger numbers. We plan to find that solution for a very specific case. The city of Denver has a large amount of parks in which we could apply TSP algorithms to. Our goal is to set up routes for anyone who wants to, or has to, clean the parks of Denver.

## Contents

- 1 Abstract
- 2 Data
- 3 The Integer Program and its Implementation
- 4 Christofides' Algorithm
- 5 Results
- 6 Poster and Powerpoint Presentation
- 7 Possible Future Improvements
- 8 References

## Abstract

Parks are a place for communities to come together. In fact, several studies <sup>[1]</sup> have shown that a clean park can help to reduce the crime of the area in and around the park. Our goal is to develop routes to clean parks in Denver. We will create these routes by solving a famous integer program called “The Traveling Salesmen Problem.” We plan to implement Christofides’ algorithm to find a route for city workers and volunteers to visit and clean our parks in a quick, efficient manner. Cleaner parks will lead to safer neighborhoods around the parks. Park locations and size are pulled from data sets provided on the Data to Policy website. Distances are calculated by using the driving distances between individual parks.

## Data

One of the challenges that our project faced was how to properly gather data. Our entire process revolved around that fact that the distance between parks had to be gathered. The simple strategy of “as the crow flies” measured the geographical distance between two locations. Although there are several pieces of software that can find these values, in reality, people use roads to travel from one location to another.



Distances between two locations as it pertains to traveling by car is called the “taxi cab metric”. Since this is more applicable than the previous strategy of “as the crow flies” this is the method that we built around our algorithm. The issue with choosing this method is that expensive software is required to find the distance between multiple locations. Instead, we decided to use a brute force method to find these distances. We decided to use distances rather than driving time because of the variability of Denver traffic.

There is data available on the Data to Policy website that has the locations of various parks in Colorado. The data was sorted to focus on the 50 largest parks in Denver. Then, google maps was used iteratively to find the distance between each park in relation to the other 49 parks. This data was then entered into Python in order to solve the Traveling Salesman Problem. A copy of said data can be found here: <https://github.com/toadhkjl/CleanParks/blob/master/Park%20Distance%20IP.xlsm>

## The Integer Program and its Implementation

We are solving the travelling salesman problem to most efficiently travel between the parks. The integer program of this is shown below:

$$\min \sum_{e \in E} c_e x_e \text{ s.t.}$$

$$\sum_{i \in \delta(j)} x_{i,j} = 2, \forall j \in V$$

$$\sum_{e \in \delta(S)} x_e \geq 2, \forall \emptyset \subset S \subset V$$

$$x_e \in \{0, 1\}, \forall e \in E$$

In this formulation,  $c_e$  represents the driving distance between the two parks in the edge,  $x_e$  represents whether or not we pick the edge, and  $\delta(S)$  are all edges going out of  $S \subseteq V$ . This first constraint guarantees that we enter and exit every park. The second constraint guarantees that we don't have any subtours, that is multiple disjoint cycles rather than a single cycle hitting every vertex. The third constraint is what makes this a binary program, we must either pick an edge between two parks or not.

For exact solutions we used a program called Sagemath. This allowed us to create a complete graph on a specified number of nodes, where the edge weight between two nodes was the distance between the corresponding parks. Sagemath then has a specific command to solve travelling salesman problems (G.traveling\_salesman\_problem(use\_edge\_labels = True)). This uses Sagemath's built in mixed integer linear programming solver CPLEX to find the solution. We give an example of the code below for 5 parks, and include the code for more parks at [https://cocalc.com/share/7364b121-f9f1-4d5d-b82b-f837523a8171/finalCode.sagews?](https://cocalc.com/share/7364b121-f9f1-4d5d-b82b-f837523a8171/finalCode.sagews?viewer=share) viewer=share in Sagemath's online compiler or at [https://github.com/toadhkjl/CleanParks/blob/master/finalCode\(1\).sagews](https://github.com/toadhkjl/CleanParks/blob/master/finalCode(1).sagews) as a file on Github.

```
G = graphs.CompleteGraph(5)

G.set_edge_label(0,1,5.7)
G.set_edge_label(0,2,4)
G.set_edge_label(0,3,7.5)
G.set_edge_label(0,4,8)
```

```
G.set_edge_label(1,2,7)
G.set_edge_label(1,3,3.2)
G.set_edge_label(1,4,6.4)
G.set_edge_label(2,3,9.5)
G.set_edge_label(2,5,3.1)
G.set_edge_label(3,4,8.8)

TSP = G.traveling_salesman_problem(use_edge_labels = True)

TSP.show()
```

We also did approximations using Christofides' algorithm. This C++ code was taken from Github <https://github.com/sth144/christofides-algorithm-cpp>. The code only ran for integer coordinates, so we used these rather than driving times for the approximations. In particular we used the latitudes and longitudes multiplied by the appropriate power of 10. This however caused overflow when calculating distances, so the code had to be changed to take long integers, rather than regular integers. We include our slightly modified code on Github at [https://github.com/toadhkjl/CleanParks/blob/master/christofides-algorithm-cpp-master\(1\).zip](https://github.com/toadhkjl/CleanParks/blob/master/christofides-algorithm-cpp-master(1).zip) which also includes the data files needed to run the code with the coordinates of each park. We give an overview of how Christofides' algorithm works in the following section.

## Christofides' Algorithm

Christofides' algorithm is a way of find a good tour of a graph when we are in a metric space. In particular it guarantees that the tour it generates is within 1.5 of the best tour created by the travelling salesman problem. Note that our formulation, with driving distances, having a metric is an good assumption. There's a non-zero driving distance as long as two parks aren't the same, driving distances are symmetric, and the triangle inequality should hold, that is if we wish to get from park a to c this is just as long as going from a to park b and b to c. Now, assume that we have a complete graph  $G$  with weighted edges. The algorithm then takes the following steps

- 1.) Calculate a minimum spanning tree,  $T$ , of  $G$ .
- 2.) Find a minimum perfect matching on the odd degree vertices in  $T$ .
- 3.) Find an Eulerian tour of the resulting (possibly non-simple) graph with the edges of  $T$  and the perfect matching.
- 4.) If we repeat a vertex, say the tour goes from  $u$  to  $v$  to  $w$ , but  $v$  has already been reached in the Eulerian tour, simply skip  $v$  and go from  $u$  to  $w$ .

We know that there are an even number of odd degree vertices in  $T$  as a degree sum of any graph is even (two times the number of edges), which means that a perfect matching actually exists. After this perfect matching has been found and combined with  $T$ , we have added one to the degree of each odd degree vertex, and changed no degrees of even vertices. This means that every vertex in the graph now has even degree so an Eulerian tour can be found. Step 4 is where the fact that we have a metric comes into play. In particular, the fact that the triangle inequality holds makes sure that this skipping of vertices can only decrease the total weight.

This algorithm can run in polynomial times. Assume that we have  $n$  vertices. Step 1 can run in  $\mathcal{O}(n^2 \log n)$ , using Prim's algorithm <sup>[2]</sup>. Step 2 can run in  $\mathcal{O}(n^{\frac{5}{2}})$  using an algorithm by Micali and Vazirani <sup>[3]</sup>. Step 3 can be done in  $\mathcal{O}(n^2)$  using Hierholzer's algorithm <sup>[4]</sup>. Finally step 4 can be done in  $\mathcal{O}(n)$  as there are at most a linear number of repetitions. Thus the entire algorithm can be done in  $\mathcal{O}(n^{\frac{5}{2}})$  time.

We now provide a proof that this algorithm produces a tour within  $3/2$  the length of the best tour, call this best value  $c$ . To do this, note that each iteration of step 4 does not increase the total weight because of the triangle inequality. Therefore we prove that all Eulerian circuits created have weight at most  $3/2 c$ . Now, if we remove a single edge from the best tour, we get a tree, say  $T'$ . Now as  $T$  is minimum we know that the cost of  $T'$  is at least the cost of  $T$ . Say that the vertices of the shortest path are (in the order they appear in the path)  $v_1, v_2, \dots, v_n$ . Now assume that we wish to find some perfect matching on the vertices  $v_{i_1}, v_{i_2}, \dots, v_{i_n}$  where  $i_1 < i_2 < \dots < i_{2k}$ . Then consider two matchings,  $i_1$  and  $i_2$ ,  $i_3$  and  $i_4$ , and so on, as well as  $i_{2k}$  and  $i_1$ ,  $i_2$  and  $i_3$  and so on. Then by the triangle inequality we know that if we sum these two matchings it is less than or equal to  $c$ . Therefore we know that one matching is at most size  $c/2$ . Therefore we have an upper bound of the Eulerian circuit of  $3c/2$  and therefore on the tour created.

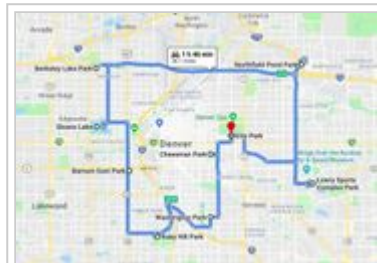
## Results

We ran the exact algorithm up to 30 parks. Due to the exponential nature of the problem, and how long the program took to run on 30 parks, we used this as our cut off for the number of parks for the exact algorithm. We were using a relatively slow server for Sagemath, so this could in theory be expanded to possibly to even 50 parks if we were to run the code on a better computer. We did however run Christofides' algorithm for all 50 parks. This is a relatively fast algorithm, so the 50 parks ran almost instantly. It seems like any reasonable number of parks could be run through this algorithm to get an approximation for the travelling salesman problem, due to short running time for 50 parks. Christofides' algorithm could also be automated significantly more easily than actually solving the travelling salesman problem. This is because our implementation uses the latitudes and longitudes which are included in the data set. This way we don't even have to find the driving times for the parks, which was time consuming and would require expensive software. If the software was purchased to automatically find the distances, this could still be used in Christofides' algorithm, and could provide an even better estimation of the best solution.

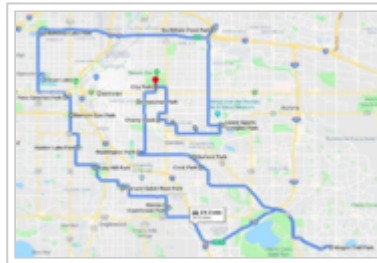
We summarize our results in the following table for multiples of 5 parks using both Christofides' algorithm and Sagemath for the true solution, as well as the result from Christofides' algorithm for 50 parks.

Number of Parks	True Solution (Miles)	Christofides' Approximation (Miles)	Ratio
5	24.3	24.3	1.00
10	38.3	43.4	1.13
15	60.9	69.3	1.14
20	65.1	76.7	1.18
25	71.8	72.8	1.01
30	77.4	86.7	1.12
50	N/A	103.4	N/A

We were at most 1.14 of the best tour, far below the theoretical limit of 1.5. Interestingly many of these tours were off from the best tour by a simple switch in the tour of some parks, or a permutation of a few sets of a small number of parks. It would be interesting to use a heuristic on the tours created by Christofides' algorithm to test some permutations of the parks to see if that could improve our bound. This would, in theory, improve our results, getting us closer to the exact value.



**Figure 1:** Shortest Path 9 Parks



**Figure 2:** Shortest Path 19 Parks

# Poster and Powerpoint Presentation

The poster that was presented at the Data 2 Policy symposium can be found here:

[https://github.com/toadhkjl/CleanParks/blob/master/Park%20Poster%20Rev\(1\).pdf](https://github.com/toadhkjl/CleanParks/blob/master/Park%20Poster%20Rev(1).pdf). A copy of the powerpoint used for our final presentation can be found here: <https://github.com/toadhkjl/CleanParks/blob/master/Final%20Integer%20Program%20PPTN.pptx>

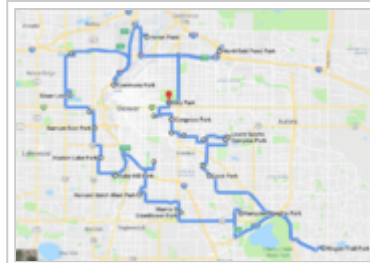
## Possible Future Improvements

One possible improvement would be to allow subtours within the program. With the length of the tours for even 10 parks, it does not make sense for a singular crew to visit all of these parks. With subtours we could break up the parks into different sections so that different crews, or the same crew over multiple days, could visit a reasonable number of park, to give them a proper cleaning. This could particularly come into play with volunteer groups. It would allow us to split up the groups to most efficiently clean the parks.

## References

1. ↑ “Community Clean Up” <https://www.ncjrs.gov/pdffiles1/171690.pdf>. July 1999
2. ↑ Prim, R. C. (November 1957), "Shortest connection networks And some generalizations", Bell System Technical Journal, 36 (6): 1389–1401, doi:10.1002/j.1538-7305.1957.tb01515.x.
3. ↑ Micali, S.; Vazirani, V. V. (1980), "An  $O(\sqrt{|V|} \cdot |E|)$  algorithm for finding maximum matching in general graphs", Proc. 21st IEEE Symp. Foundations of Computer Science, pp. 17–27, doi:10.1109/SFCS.1980.12
4. ↑ N. L. Biggs, E. K. Lloyd and R. J. Wilson, Graph Theory 1736–1936, Clarendon Press, Oxford, 1976, 8–9, ISBN 0-19-853901-0

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Cleaning\\_Parks\\_for\\_a\\_Safer\\_Future&oldid=2065](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Cleaning_Parks_for_a_Safer_Future&oldid=2065)"



**Figure 3:** Shortest Path 24 Parks

- This page was last modified on 7 May 2019, at 17:39.
- This page has been accessed 6,821 times.

# Clustering Neighborhoods in Order to Analyze Policy Needs

From CU Denver Optimization Student Wiki

## Contents

- 1 Project Contributors
- 2 Abstract
- 3 GitHub
- 4 A Brief Discussion of the Constrained K Means Algorithm
- 5 Results

## Project Contributors

Alex Semyonov ([http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Alex\\_Semyonov](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Alex_Semyonov))

Orlando Gonzalez ([http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Orlando\\_Gonzalez](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Orlando_Gonzalez))

Jacob Dunham ([http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Jacob\\_Dunham](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Jacob_Dunham))

This project is a collaborative effort between Alexander Semyonov, Orlando Gonzalez and Jacob Dunham. The project was to be completed as an assignment for Steffen Borgwardt's Linear Programming class at the University of Colorado Denver. Our project was presented at The University of Colorado Denver's Fall 2023 Data to Policy symposium, winning best in data. Each contributor's page is linked above.

## Abstract

In this project we will attempt to find an optimal clustering of the neighborhoods of Denver into a given number of groups based on various features of the neighborhoods. Specific features of interest include: average rent, average income, acres of park per 100 residents, percentage of population that spend more than 30% on rent, total jobs, and total housing units. Each neighborhood will be represented as a vector of these neighborhood features and we will utilize the K-Means algorithm to find an approximate optimal clustering. Additionally, we will formulate a linear program to solve this clustering problem and compare the clusterings generated by each method. Once we have identified a satisfactory clustering of neighborhoods, we will analyze the characteristics and policies of neighborhoods in the same clusters in order to find commonalities. We can then determine the needs of certain groups of neighborhoods relative to other groups. If there are large discrepancies between groups in certain categories we can make policy recommendations targeted to those specific neighborhoods which are disproportionately in need of resources. We will also do an analysis of policies in place for clusters of neighborhoods which are considered to have fair access to many desirable resources, and determine if similar policies may work for clusters of neighborhoods which have less access to these resources. With this clustering method, we can more easily compare and contrast policies that do and do not work for groups of neighborhoods, and in turn make more targeted policy suggestions.

## GitHub

Link to project GitHub: <https://github.com/asems99/Clustering-Neighborhoods-to-Identify-Policy-Needs-Policy-Needs/tree/main>

The GitHub contains the full Python code for the constrained K Means mode (this file is called LP Neighborhood Project Code), there is also a PDF of this file which displays all of the code and plots. Additionally, the GitHub contains the AMPL model formulation code (the file is named Model Formulation. mod) as well as all of the data used in the model, the presentation slides for the linear programming course, and the results of the model. If anyone should have any questions about the project, please contact any one of the collaborators, whose contact information can be found on their personal wiki pages.

## A Brief Discussion of the Constrained K Means Algorithm

The algorithm we developed to try and find an optimal constrained clustering functions very similarly to the standard K Means algorithm (sometimes called Lloyd's algorithm) in the sense that it alternates between an assignment step and a cluster center update step. In order to use our algorithm, the specific number of clusters must be known beforehand and the cluster centers are initialized via the standard K Means algorithm (the hope is that this would help prevent particularly problematic initializations). During the assignment step of the algorithm, we use our current cluster centers (or our initialized clusters for first iteration) to assign the neighborhoods to the cluster whose center it is closest to by solving the following Linear Program (note:  $c_{ij}$  is given by the 2-Norm distance between neighborhood  $i$  and cluster center  $j$ ):

$$\text{Minimize } \sum_{j=1}^5 \sum_{i=1}^{65} c_{i,j} Y_{i,j}$$

$$\text{Subject to: } \sum_{i=1}^{65} Y_{i,j} = 13 \quad \forall j = 1, 2, 3, 4, 5$$

$$\sum_{j=1}^5 Y_{i,j} = 1 \quad \forall i = 1, 2, \dots, 65$$

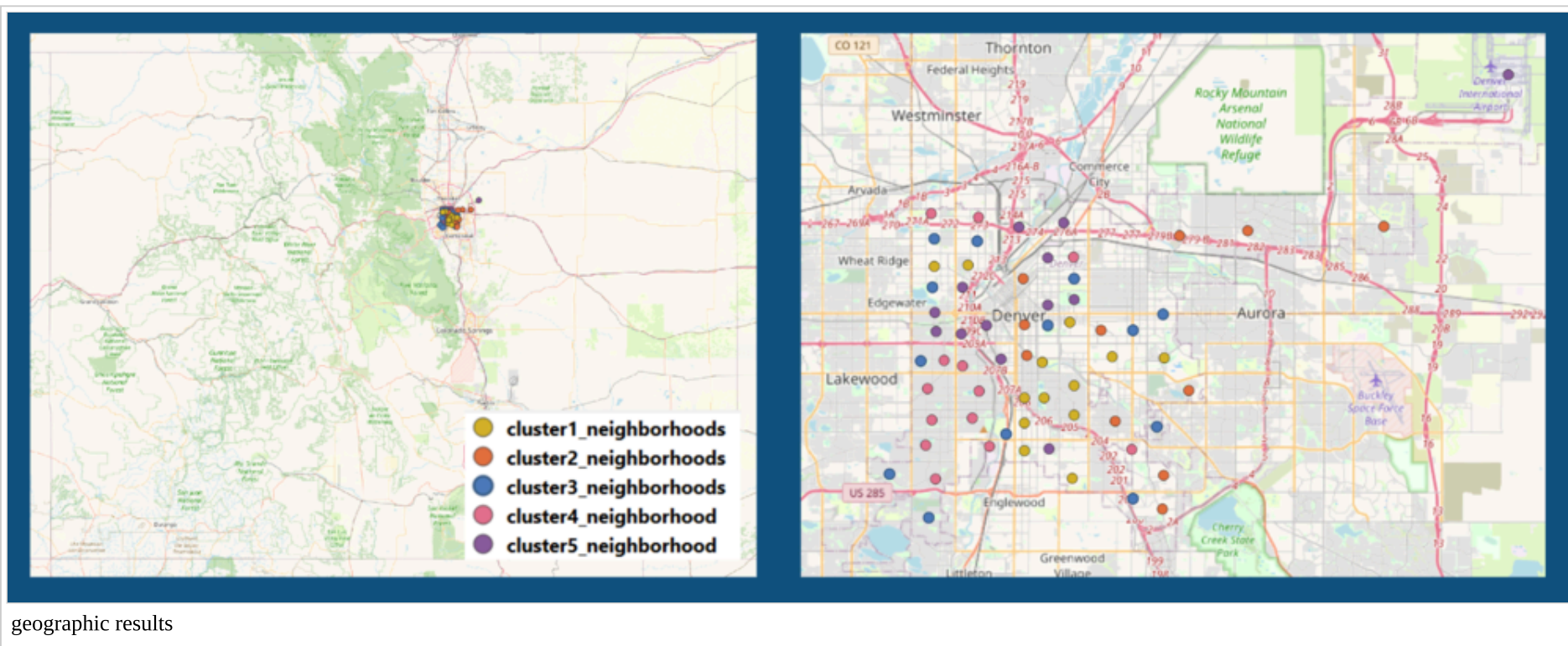
$$Y_{i,j} \in [0, 1]$$

Upon solving this LP, we find our clustering of neighborhoods and now want to update these centers, we do this by computing the centroid of each cluster which is simply the arithmetic mean of all points in the clusters. These centroids become our new cluster centers and we now recompute our costs (2-Norm distance between neighborhoods and new cluster centers) and return back to the assignment step. We alternate between these two steps until our objective function in the LP formulation above stops decreasing (which typically occurs after 3 or 4 iterations for this specific data).

This algorithm was created using Jupyter and AMPLPy and the .ipynb code (alongside a PDF preview of this code )file can be found using the GitHub link provided above.

## Results

After running our constrained Kmeans algorithm we were able to develop clusters of alike neighborhoods within Denver. We then mapped the neighborhoods on GIS software in order to develop a geographic representation of our final results. From these results we can determine geographically targeted policies to improve neighborhood clusters.



Retrieved from

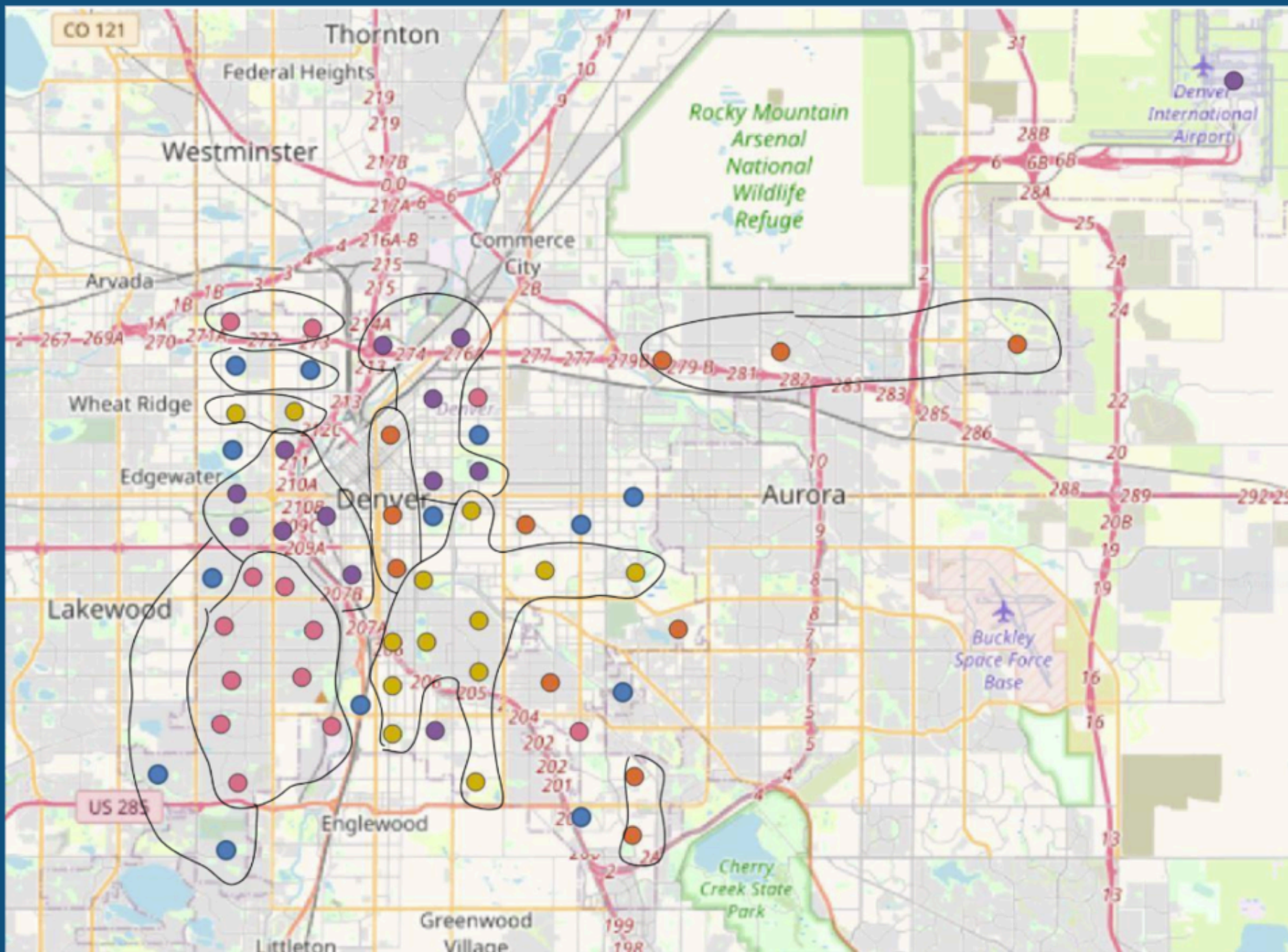
"[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Clustering\\_Neighborhoods\\_in\\_Order\\_to\\_Analyze\\_Policy\\_Needs&oldid=4614](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Clustering_Neighborhoods_in_Order_to_Analyze_Policy_Needs&oldid=4614)"

- This page was last modified on 6 December 2023, at 13:10.
- This page has been accessed 85 times.



-  **cluster1\_neighborhoods**
-  **cluster2\_neighborhoods**
-  **cluster3\_neighborhoods**
-  **cluster4\_neighborhood**
-  **cluster5\_neighborhood**

legend



geographic clusters

Our algorithm was able to successfully group our neighborhoods and also find our cluster centers. I've provided our final clustering below:

**Cluster 1:** 'Belcaro', 'Congress Park', 'Cory - Merrill', 'Country Club', 'Highland', 'Hilltop', 'Lowry Field', 'Platt Park', 'Rosedale', 'Washington Park', 'Washington Park West', 'Wellshire Denver', 'West Highland'

**Cluster 2:** 'Capitol Hill', 'Cherry Creek', 'Five Points', 'Green Valley Ranch Denver', 'Hale', 'Hampden', 'Hampden South', 'Montbello', 'Speer', 'Stapleton Denver', 'Virginia Village', 'Washington Virginia Vale', 'Windsor'

**Cluster 3:** 'Barnum West', 'Bear Valley', 'Berkeley', 'Cheesman Park', 'East Colfax', 'Fort Logan', 'Indian Creek', 'Montclair', 'Overland', 'Skyland', 'Sloan Lake', 'Southmoor Park', 'Sunnyside'

**Cluster 4:** 'Athmar Park', 'Barnum', 'Chaffee Park', 'Clayton', 'College View - South Platte', 'Goldsmith', 'Harvey Park', 'Harvey Park South', 'Mar Lee', 'Regis', 'Ruby Hill', 'Valverde', 'Westwood'

**Cluster 5:** 'Baker', 'City Park', 'City Park West', 'Cole', 'Denver International Airport', 'Elyria Swansea', 'Globeville', 'Jefferson Park', 'Lincoln Park', 'Sun Valley', 'University', 'Villa Park', 'West Colfax'

final clusters

# Code for Knapsack Problem Algorithms

From CU Denver Optimization Student Wiki

This page contains a Java implementation of the dynamic programming algorithm used to solve an instance of the Knapsack Problem, an implementation of the Fully Polynomial Time Approximation Scheme for the Knapsack Problem, and programs to generate or read in instances of the Knapsack Problem. To learn more, see Knapsack Problem Algorithms.

## Contents

- 1 KnapsackProblem.java
- 2 DynamicKnapsack.java
- 3 KnapsackApproximation.java
- 4 KnapsackGenerator.java
- 5 ParseKnapsackCSV.java
- 6 KnapsackTest.java

## KnapsackProblem.java

The following class is used to represent an instance of the Knapsack Problem.

```
1. import java.io.BufferedWriter;
2. import java.io.File;
3. import java.io.FileWriter;
4. import java.io.PrintWriter;
5.
6. public class KnapsackProblem {
7.
8.     private int n;           //number of objects
9.     private int[] p;         //array of profits
10.    private int[] s;          //array of sizes
11.    private int B;            //capacity of knapsack
12.
13.    private boolean[] solution; //optimal solution given by boolean array
14.    private int Z;              //optimal objective value
15.
16.
17.    //accessor methods
18.    public int getN(){
19.        return this.n;
20.    }
```

```

21. public int[] getProfits(){
22.     return this.p;
23. }
24. public int[] getSizes(){
25.     return this.s;
26. }
27. public int getCapacity(){
28.     return this.B;
29. }
30. public int getZ(){
31.     return this.Z;
32. }
33. public boolean[] getSolution(){
34.     return this.solution;
35. }
36.
37. //mutator methods
38. public void setZ(int Z){
39.     this.Z = Z;
40. }
41. public void setSolution(boolean[] solution){
42.     this.solution = solution;
43. }
44.
45. //constructor override; assumes p and s have length n
46. public KnapsackProblem(int n, int p[], int s[], int B){
47.     this.n = n;
48.     this.p = p;
49.     this.s = s;
50.     this.B = B;
51. }
52.
53. //solve problem using Dynamic Programming for Knapsack
54. public void solve(){
55.     DynamicKnapsack.solve(this);
56. }
57.
58. //write an lp file of the associated IP for CPLEX to solve
59. public void writeLP(){
60.     PrintWriter out = null;
61.     try{
62.         File lp = new File("knapsack" + n + ".lp");
63.
64.         out = new PrintWriter(new BufferedWriter(new FileWriter(lp, false)));
65.     } catch(Exception e){
66.         e.getMessage();
67.         System.exit(0);
68.     }
69.     out.println("Maximize");
70.
71.     //print objective
72.     out.print(" obj: ");
73.     out.print(p[0] + " x0");
74.     for(int i=1; i<n; i++)
75.         out.print(" + " + p[i] + " x" + i );
76.     out.println();
77.

```



```

78.         //print constraint
79.         out.println("Subject To");
80.         out.print(" c1: " + s[0] + " x0");
81.         for(int i=1; i<n; i++)
82.             out.print(" + " + s[i] + " x" + i);
83.         out.println(" <= " + B);
84.
85.         //domain restriction
86.         out.println("Binaries");
87.         for(int i=0; i<n; i++)
88.             out.println(" x" + i);
89.
90.         out.println("End");
91.         out.close();
92.         System.out.println("LP file written. \n");
93.     } //end method writeLP()
94.
95. } //end class KnapsackProblem

```

## DynamicKnapsack.java

Contains a static method used to solve an instance of KnapsackProblem using the dynamic programming algorithm.

```

1. //algorithm for solving KnapsackProblem in pseudo-polynomial time using Dynamic Programming for Knapsack
2. public class DynamicKnapsack {
3.
4.     public static void solve(KnapsackProblem kp){
5.
6.         //obtain problem information
7.         int n = kp.getN();
8.         int[] p = kp.getProfits();
9.         int[] s = kp.getSizes();
10.        int B = kp.getCapacity();
11.
12.        //calculate maximum possible profit nP
13.        int P = 0;
14.        for (int i = 0; i < n; i++){
15.            if(p[i] >= P){
16.                P = p[i];
17.            }
18.        }
19.        int nP = n * P;
20.
21.        //initialize profit array (use only two arrays instead of n)
22.        int[][] A = new int[2][nP + 1];
23.
24.        //initialize solution array
25.        boolean[][] S = new boolean[2][nP+1][n];
26.
27.        //empty solution
28.        boolean[] zeros = new boolean[n];
29.        for(int i=0; i<n; i++) zeros[i] = false;

```

```

30. //initialize first columns of A and S
31. A[0][0] = 0;
32. S[0][0] = zeros.clone();
33. for (int j = 1; j <= nP; j++){
34.     if(j == p[0]){
35.         A[0][j] = s[0];
36.         S[0][j] = zeros.clone();
37.         S[0][j][0] = true;
38.     } else{
39.         A[0][j] = 1000000000; //assumes sizes are not too large
40.         S[0][j] = null;
41.     }
42. }
43.
44.
45. A[1][0] = 0;
46. S[1][0] = zeros.clone();
47.
48. //recursively generate A[1] and S[1] for subsequent subproblems
49. for(int i = 1; i < n; i++){
50.     for (int j = 1; j <= nP; j++){
51.         if( p[i] <= j){
52.             if(s[i] + A[0][j - p[i]] < A[0][j]){
53.                 A[1][j] = s[i] + A[0][j - p[i]];
54.                 S[1][j] = S[0][j - p[i]].clone();
55.                 S[1][j][i] = true;
56.             } else{
57.                 A[1][j] = A[0][j];
58.                 if(S[0][j]==null) S[1][j] = null;
59.                 else S[1][j] = S[0][j].clone();
60.             }
61.         } else{
62.             A[1][j] = A[0][j];
63.             if(S[0][j]==null) S[1][j] = null;
64.             else S[1][j] = S[0][j].clone();
65.         }
66.     } //end for
67.
68.     //save A[1] and S[1] in A[0] and S[0]
69.     A[0] = A[1].clone();
70.     S[0] = S[1].clone();
71. }
72.
73. //find optimal solution using final stage solutions
74. int Z = 0;
75. boolean[] solution = null;
76. for(int j=1; j <= nP; j++){
77.     if(A[0][j] <= B){
78.         Z = j;
79.         solution = S[0][j];
80.     }
81. } //end for
82.
83. //record solution
84. kp.setZ(Z);
85. kp.setSolution(solution);
86.

```

```
87.     }//end method solve
88.
89. }//end class DynamicKnapsack
```

# KnapsackApproximation.java

Approximates an instance of the Knapsack Problem using FPTAS for Knapsack.

```
1. public class KnapsackApproximation {
2.
3.     private int n;           //number of objects
4.     private int[] p;         //array of profits
5.     private int[] s;         //array of sizes
6.     private int B;           //capacity of knapsack
7.     private double eps; //accuracy parameter
8.
9.     private boolean[] solution = null; //optimal solution given by boolean array
10.    private int Z = 0;         //optimal objective value
11.
12.
13.    //accessor methods
14.    public int getN(){
15.        return this.n;
16.    }
17.    public int[] getProfits(){
18.        return this.p;
19.    }
20.    public int[] getSizes(){
21.        return this.s;
22.    }
23.    public int getCapacity(){
24.        return this.B;
25.    }
26.    public int getZ(){
27.        return this.Z;
28.    }
29.    public boolean[] getSolution(){
30.        return this.solution;
31.    }
32.    public double getEps(){
33.        return this.eps;
34.    }
35.
36.
37.    //constructor override; assumes p and s have length n
38.    public KnapsackApproximation(int n, int p[], int s[], int B, double eps){
39.        this.n = n;
40.        this.p = p;
41.        this.s = s;
42.        this.B = B;
43.        this.eps = eps;
44.    }
```



```

45. //otherwise, construct object from existing KnapsackProblem
46. public KnapsackApproximation(KnapsackProblem kp, double eps){
47.     this.n = kp.getN();
48.     this.p = kp.getProfits().clone();
49.     this.s = kp.getSizes().clone();
50.     this.B = kp.getCapacity();
51.     this.eps = eps;
52. }
53.
54. //approximate solution using dynamic programming with scaled profits
55. public void solve(){
56.     //calculate maximum profit P
57.     int P = 0;
58.     for (int i = 0; i < n; i++)
59.         if(p[i] >= P) P = p[i];
60.
61.     //calculate scaling factor K
62.     double K = (eps * P) / n;
63.     if(K < 1){
64.         System.out.println("Scaling factor K is too small: K=" + K + ". Need K > 1.");
65.         return;
66.     }else{
67.         System.out.println("Scaling factor: K=" + K );
68.
69.     }
70.
71.     //scale down profits
72.     int[] p_scaled = new int[n];
73.     for(int i=0; i<n; i++)
74.         p_scaled[i] = (int) (p[i] / K);
75.
76.     //solve scaled problem
77.     KnapsackProblem kp_scaled = new KnapsackProblem(n, p_scaled, s, B);
78.     kp_scaled.solve();
79.
80.     //obtain (1-eps)-approximate solution
81.     solution = kp_scaled.getSolution();
82.     Z = 0;
83.     for(int i=0; i<n; i++)
84.         if(solution[i]) Z = Z + p[i];
85.
86. } //end method solve()
87.
88. } //end class KnapsackApproximation

```

## KnapsackGenerator.java

Randomly generates an instance of KnapsackProblem for a given problem size and maximum profit.

```

1. public class KnapsackGenerator {
2.
3.     //generate a knapsack problem with size n and max profit at most P

```

```

4. public static KnapsackProblem generateProblem(int n, int P){
5.
6.     //knapsack capacity is a function of n
7.     int B = 80*n;
8.
9.     //sizes distributed between 20 and 400
10.    int[] s = new int[n];
11.    for(int i=0; i<n; i++)
12.        s[i] = (int) ((Math.random()*380) + 20);
13.
14.    //profits will be somewhat correlated with sizes
15.    int[] p = new int[n];
16.    for(int i=0; i<n; i++)
17.        p[i] = (int) (10 + ((Math.random()*P*s[i])/400));
18.
19.    return (new KnapsackProblem(n,p,s,B));
20. } //end method
21.
22. } //end class KnapsackGenerator

```

## ParseKnapsackCSV.java

Read in an instance of the Knapsack Problem from files found on <http://www.diku.dk/~pisinger/codes.html>.

```

1. import java.io.File;
2. import java.util.LinkedList;
3. import java.util.Scanner;
4.
5. public class ParseKnapsackCSV{
6.
7.     //construct a KnapsackProblem object given a csv file from http://www.diku.dk/~pisinger/codes.html
8.     public static KnapsackProblem buildKapnsack(String filename){
9.
10.        System.out.println("Loading knapsack...");
11.
12.        //temporarily store data from file in linked lists
13.        LinkedList<Integer> profits = new LinkedList<Integer>();
14.        LinkedList<Integer> sizes = new LinkedList<Integer>();
15.
16.        //problem parameters
17.        int n=0;
18.        int B=0;
19.
20.        //line counter
21.        int i = 0;
22.
23.        //initialize scanner object for file
24.        Scanner in = null;
25.        try {
26.            in = new Scanner(new File(filename));
27.        } catch (java.io.FileNotFoundException e) {
28.            System.out.println(e.getMessage());

```

```

29.         System.exit(0);
30.     }
31.
32.     //parse each line until first problem is finished (will encounter "-----")
33.     while (in.hasNext()){
34.         i++;
35.         String line = in.nextLine();
36.         if(i==1 || i==5) continue;
37.         if(line.equals("-----")) break;
38.
39.         if(i==2){
40.             String[] values = line.split("\\s+");
41.             n = Integer.parseInt(values[1]);
42.         }else if(i==3){
43.             String[] values = line.split("\\s+");
44.             B = Integer.parseInt(values[1]);
45.         }else if(i==4){
46.             String[] values = line.split("\\s+");
47.             System.out.println("Optimal objective: " + values[1]);
48.         }else{
49.             String[] values = line.split(",");
50.             profits.add(Integer.parseInt(values[1]));
51.             sizes.add(Integer.parseInt(values[2]));
52.         } //end else
53.     } //end while
54.     in.close();
55.
56.     //create profit and size arrays
57.     int[] p = new int[n];
58.     int[] s = new int[n];
59.     int j = 0;
60.
61.     for(Integer l : profits){
62.         p[j] = l;
63.         j++;
64.     }
65.     j=0;
66.     for(Integer l : sizes){
67.         s[j] = l;
68.         j++;
69.     }
70.
71.     //create knapsack problem
72.     KnapsackProblem kp = new KnapsackProblem(n,p,s,B);
73.
74.     System.out.println("Knapsack problem " + filename + " has been successfully uploaded.\n");
75.     return kp;
76. } //end method buildNetwork
77.
78. } //end class ParseKnapsackCSV

```

## KnapsackTest.java

Test randomly generated KnapsackProblem or one generated from .csv file.

```

1. public class KnapsackTest {
2.
3.     public static void main(String[] args){
4.
5.         //use knapsack problem from csv file found on http://www.diku.dk/~pisinger/codes.html
6.         // KnapsackProblem kp = ParseKnapsackCSV.buildKnapsack("knapPI_11_20_1000.csv");
7.         // KnapsackProblem kp = ParseKnapsackCSV.buildKnapsack("knapPI_13_100_1000.csv");
8.         // KnapsackProblem kp = ParseKnapsackCSV.buildKnapsack("knapPI_16_50_1000.csv");
9.         // KnapsackProblem kp = ParseKnapsackCSV.buildKnapsack("knapPI_16_500_1000.csv");
10.        // kp.writeLP();
11.
12.        // or randomly generate a problem of size n and max profit P using generateProblem(n, P);
13.        KnapsackProblem kp = KnapsackGenerator.generateProblem(20, 10000);
14.
15.        int n = kp.getN();
16.        // kp.writeLP(); //if desired, write lp file for CPLEX to solve associated integer program
17.
18.        //print problem details
19.        System.out.println("Number of objects: " + n);
20.        System.out.print("Profits: ");
21.        for (int i=0; i<n; i++)
22.            System.out.print(kp.getProfits()[i] + " ");
23.        System.out.println();
24.        System.out.print("Sizes: ");
25.        for (int i=0; i<n; i++)
26.            System.out.print(kp.getSizes()[i] + " ");
27.        System.out.println("\n");
28.
29.        //dynamic knapsack algorithm
30.        System.out.println("Dynamic knapsack:");
31.        kp.solve();
32.        System.out.println("Optimal objective: " + kp.getZ());
33.        System.out.println("Optimal solution: ");
34.        for(int i=0; i<n; i++)
35.            if( kp.getSolution()[i] )System.out.print(i + " ");
36.        System.out.println("\n ");
37.
38.        //polynomial approximation algorithm
39.        System.out.println("Approximation algorithm with epsilon=" + eps);
40.        KnapsackApproximation ka = new KnapsackApproximation(kp, eps);
41.        ka.solve();
42.        System.out.println("Objective: " + ka.getZ());
43.        System.out.println("Solution:");
44.        for(int i=0; i<n; i++)
45.            if(ka.getSolution() != null) if( ka.getSolution()[i] ) System.out.print(i + " ");
46.        System.out.println("\n ");
47.
48.    } //end main
49.
50. } //end class KnapsackTest

```

- This page was last modified on 2 May 2017, at 11:33.
- This page has been accessed 11,349 times.

# Colin Furey

From CU Denver Optimization Student Wiki

## Contents

- 1 About me
- 2 Education
- 3 Projects
- 4 Github

## About me

Hello, my name is Colin Furey. I'm a student at CU Denver interested in mathematics and the application of linear programming to real world problems.

## Education

I have a Bachelor of Arts in mathematics and history from CU Denver.

## Projects

Fall 2023 - Optimizing car availability in the Denver metro area with Alana Saragosa and Paul Guidas. Our project is called Emissions and Equality: Colorado Car Share Optimization

Spring 2024 - Investigating different implementations of Dijkstra's shortest path algorithm DijkstraHeapImplementation

## Github

Github Containing Code Used for Dijkstra Heap Implementation Project (<https://github.com/fureyc/Dijkstra-Project>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Colin\\_Furey&oldid=4704](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Colin_Furey&oldid=4704)"

Category: Contributors

- This page was last modified on 1 May 2024, at 11:17.
- This page has been accessed 50 times.

# Collin Powell

From CU Denver Optimization Student Wiki

I a Graduate student at CU Denver, I have a Bachelor's Degree in Mathematics and Physics from Oberlin College, and a Masters Degree in Applied Mathematics from CU Boulder.

When not working on Mathematics, I enjoy spending time with my wife and two daughters as well as gaming.

Current Project: Vaccine Distribution to optimize herd immunity with Sandra Robles and Michael Burgher (Spring 2021).

Previous Project: Optimizing Highschool Graduation Rates with Zane Showalter-Castorena and Alyssa Newman

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Collin\\_Powell&oldid=3119](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Collin_Powell&oldid=3119)"

Category: Contributors

- This page was last modified on 13 April 2021, at 10:46.
- This page has been accessed 954 times.





# Column Generation

From CU Denver Optimization Student Wiki

Column generation involves solving linear programs which contain too many variables to consider all possible combinations of solutions. The master problem is therefore broken into subproblems solving for the reduced cost. The dual of the subproblem is then solved. As long as the reduced cost is negative, the pattern continues to iterate adding new columns to our constraint matrix. Once our reduced cost is no longer negative, we have our optimal solution.

## Contents

- 1 Deriving Reduced Cost
- 2 Cutting Stock Example
  - 2.1 Dictionary of Variables
  - 2.2 Problem
  - 2.3 Solution
- 3 References

## Deriving Reduced Cost

Primal

$$\min c^T x$$

$$s.t. Ax = b$$

$$x \geq 0$$

Dual

$$\max b^T y$$

$$s.t. A^T y \leq c$$

Decompose

$$A = (B|N)$$

$$x = (x_B \ x_N)^T$$

$$c = (c_B^T \ c_N^T)^T$$

Then  $Ax = (B|N)(x_B \ x_N)^T$

$$= Bx_B + Nx_N = b$$

$$x_B = B^{-1}b - B^{-1}Nx_N$$

Thus, we wish to  $\min c^T x = c^T(x_B + x_N)$

$$= c_B^T x_B + c_N^T x_N$$

$$= c_B^T (B^{-1}b - B^{-1}Nx_N) + c_N^T x_N$$

$$= c_B^T B^{-1}b + (c_N^T - c_B^T B^{-1}N)x_N$$

$c_N^T - c_B^T B^{-1}N$  is our reduced cost.

## Cutting Stock Example

Example by Kedric Daly, Northwestern University<sup>[1]</sup>

### Dictionary of Variables

$l_i$     *length of demand*

$b_i$     *demand for each piece of length  $l_i$*

$P$     *set of all cutting patterns*

$a_{ip}$     *number of pieces of length  $l_i$  cut in pattern  $p$*

$x_p$     *number of times pattern  $p$  is cut*

$r$     *reduced cost*

## Problem

We are given rolls of length  $L = 16m$  that are to be cut into lengths  $l = (3 \ 6 \ 7)^T$  with demand  $b = (25 \ 20 \ 18)^T$ . We wish to

$$\begin{aligned} \min z &= \sum_p x_p \\ \text{s.t.} \quad &\sum Ax \geq b \\ &x_i \geq 0 \end{aligned}$$

The dual is

$$\begin{aligned} \max y &= \sum b^T \pi \\ \text{s.t.} \quad &A^T \pi \leq (1 \ \dots \ 1)^T \\ &\pi_i \geq 0 \end{aligned}$$

## Solution

Let

$$A = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

So  $x = (5 \ 10 \ 9)^T$

$$z = \sum x_i = 5 + 10 + 9 = 24 \text{ rolls needed.}$$

Solving the dual yields  $\pi = (1/5 \ 1/2 \ 1/2)^T$

Our reduced cost is now  $r = \min(1 - \sum \pi_i a_{ip})$

$$\text{s.t. } l^T a_{ip} \leq L$$

$$r = 1 - \max((1/5)a_{1p} + (1/2)a_{2p} + (1/2)a_{3p})$$

$$\text{s.t. } 3a_{1p} + 6a_{2p} + 7a_{3p} \leq 16$$

Solving the linear program, we find  $a_p = (1 \ 2 \ 0)^T$

So  $r = 1 - ((1/5)(1) + (1/2)(2) + (1/2)(0)) = -1/5 < 0$

The reduced cost is negative. This column is therefore added to  $A$ .

Now  $5 \ 0 \ 0 \ 1$

$$A = 0 \ 2 \ 0 \ 2$$

$$0 \ 0 \ 2 \ 0$$

and  $x = (4 \ 0 \ 9 \ 10)^T$

$z = \sum x_i = 4 + 9 + 10 = 23$  rolls needed.

Solving the dual yields  $\pi = (1/5 \ 2/5 \ 1/2)^T$

$$r = 1 - \max((1/5)a_{1p} + (2/5)a_{2p} + (1/2)a_{3p})$$

$$s.t. \ 3a_{1p} + 6a_{2p} + 7a_{3p} \leq 16$$

Solving the linear program, we find  $a_p = (1 \ 1 \ 1)^T$

So  $r = 1 - ((1/5)(1) + (2/5)(1) + (1/2)(1)) = -1/10 < 0$

Our reduced cost is still negative, so this column is added to  $A$ .

Now  $5 \ 0 \ 0 \ 1 \ 1$

$$A = 0 \ 2 \ 0 \ 2 \ 1$$

$$0 \ 0 \ 2 \ 0 \ 1$$

$x = (6/5 \ 0 \ 0 \ 1 \ 18)^T$

$z = \sum x_i = 6/5 + 1 + 18 = 20 + 1/5$  or 21 rolls needed.

Solving the dual yields  $\pi = (1/5 \ 2/5 \ 2/5)^T$

Thus  $a_p = (5 \ 0 \ 0)^T$

So  $r = 1 - ((1/5)(5) + (2/5)(0) + (2/5)(0)) = 0$

Our reduced cost is no longer negative, so our iterations stop. This column is not added to  $A$ .

Our optimal solution is **21** rolls!

## References

1. ↑ "Column Generation Algorithms", Kedric Daly. [https://optimization.mccormick.northwestern.edu/index.php/Column\\_generation\\_algorithms](https://optimization.mccormick.northwestern.edu/index.php/Column_generation_algorithms). Accessed 7 Dec. 2017. Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Column\\_Generation&oldid=957](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Column_Generation&oldid=957)"

- This page was last modified on 7 December 2017, at 10:20.
- This page has been accessed 10,049 times.

# Complementary slackness

From CU Denver Optimization Student Wiki

It is possible to find a solution to the dual problem when only the optimal solution to the primal is known. This is the theorem of complementary slackness. Formally it states:

Suppose that  $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$  is primal feasible and that  $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_m)$  is dual feasible. Let  $(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m)$  denote the corresponding primal slack variables, and let  $(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n)$  denote the corresponding dual slack variables. Then  $\mathbf{x}$  and  $\mathbf{y}$  are optimal for their respective problems if and only if

- $\mathbf{x}_j \mathbf{z}_j = 0$ , for  $j = 1, 2, \dots, n$ , and
- $\mathbf{w}_i \mathbf{y}_i = 0$ , for  $i = 1, 2, \dots, m$ .

This means that whenever the primal variable is active at a constraint, the dual variable is equal to 0 and vice versa.

For an in depth look into the theory behind complementary slackness, please see the Linear Programming section included in the overview of Lagrangian Duality.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Complementary\\_slackness&oldid=279](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Complementary_slackness&oldid=279)"

- 
- This page was last modified on 30 March 2017, at 15:52.
  - This page has been accessed 29,397 times.

# Computing Using Mathematical Programs

From CU Denver Optimization Student Wiki

Many mathematical programs are far too large and complex to solve by hand, therefore computers are used to do the calculations. However, it is necessary to know how to tell the computer what to do. There are some common standard solvers and file types that provide a template for optimizing. Python will be the standard language discussed, as most mathematicians have worked with it. The solver for Python, lpsolve55, was written for C++ and modified to work for python, which makes some of the implementation difficult in Python. However, because Python is a higher level language and therefore further away from machine code, it is easier for humans to read and consequently understand.

## Contents

- 1 File Types
  - 1.1 .lp Files
  - 1.2 .mps Files
  - 1.3 Dualizing a File
- 2 IDLEs
- 3 Solver Issues
- 4 References

## File Types

The most common programming file types are .lp and .mps files. These are both file types that can be read by a wide variety of solvers. Many languages have solvers for these built in. For example, Python has lpsolve55 as its default solver for .lp and .mps files.

### .lp Files

A .lp (linear program) file is the most intuitive linear program file, as it is written much like a linear program in standard form. It has an objective function, constraints, inequalities and equalities, and non-negativity constraints where necessary. In .lp files, non-negativity constraints are considered variable bounds, and that section of the file will contain all constraints on single variables. Consider the following linear program is the same program found in the shadow price section of the wiki:

$$\begin{array}{llllll} \max & 40x_1 & + & 50x_2 & & \\ s.t & x_1 & + & x_2 & \leq & 1,000 \\ & x_1 & & & \leq & 700 \\ & & & 2x_2 & \leq & 1,100 \\ & & & x_2 & \geq & 300 \end{array}$$

The program is written as a .lp file in source code:

```
max: 40 x1 50 x2 ;
/* Constraints */
x1 x2 <= 1000;
x1 <= 700;
2 x2 <= 1100;
x1 >= 300;
x2 >= 300;
/* Variable Bounds */
x1, x2 >= 0
```

This is how the file will look when opened in a simple text editor. Notice that there are no addition signs. This is because an .lp format can only handle linear programs.

However, if the variables were multiplied together the program would become non-linear. The solver that Python uses has a variety of algorithms that can be called to use for solving any .lp file. These range from the simplex algorithm, which is the standard algorithm for lpsolve, to primal-dual algorithms and dual solve methods. The .lp file can also be generated in Python, by writing a program in this form and then commanding "return = lpsolve('write\_lp', lp, filename)" instead of "return = lpsolve('solve\_lp', lp)". <sup>[1]</sup>

The interesting aprt of using lpsolve is that when a program is entered in this form, as soon ans the information in the file is stored to a variable, the variable takes the most simplified for of the program the computer can conceive. For example, after running the code:

```
lp = lpsolve('read_LP', 'My_Lp.lp')
lpsolve('solve', lp)
lp = lpsolve('solve', lp)
lpsolve('write_lp', lp, 'My_Lp2.lp')
```

The following program will be written:

```
max: 40 x1 50 x2
/* Constraints */
x1 x2 <=1000
/* Variable Bounds */
300 <= x1 <= 700
300 <= x2 <= 550
```

This is the same program, but with some of the constraints being read as variable bounds. This type makes the primal problem easier to solve, but may confuse the machine when generating a new dual program.

While Python is a very easy language to do this in, there are some IDLEs that have interfaces that are tuned more for linear program writing, such as Lp\_Solve. This program can be used to write .lp files and solve them, and can be imported as a package into Python, Matlab, and many other languages to be solved in whatever language the user prefers to code in. Writing these programs in an IDLE such as this is is usually simpler, as often times the IDLE will ask for inputs that are then directly written to a text editor. For comparison, writing this in Python requires specification on each line that the solver is writing that line into the program and then ensuring that constraints are aligned properly, which can be difficult and more time consuming for large programs.



## .mps Files

.mps (mathematical programming system) files contain the same information that .lp files contain, but are written differently. While a .lp file is written vary similar to the conventional way a linear program is written, a .mps file is written in column format. For comparison, the previous program written in .mps form looks like:

name		friction				
n		cost				
l		lim1				
l		lim2				
l		lim3				
g		dem1				
g		dem2				
columns						
xone		cost	40	lim1	1	
xone		lim2	1	dem1	1	
xtwo		cost	50	lim1	1	
xtwo		lim3	2	dem2	1	
rhs						
rhs1		lim1	1000	lim2	700	
rhs1		lim3	1100	dem1	300	
rhs1		dem2	300			
enddata						

Notice how the program individually lists all of variables and their coefficients with the particular constraint. It is more difficult for a human to read and write, but .mps files are able to solve mixed integer programs, where .lp files can only work with linear programs.

## Dualizing a File

In Python, generating the dual program is rather simple. Instead of creating a whole new program, one can take the known .lp file, import it after importing the lpsolve package, and run the following code:

```
[obj, x, duals, return] = lpsolve('get_solution', lp)
```

This code will then print the objective function value, the dual objective function value, and the duals.

In order to get a file of the dual program, access the directory in which the .lp or .mps file is contained, then import the lpsolve library and run the following code.

```
from lpsolve55 import *
lp = lpsolve('read_LP', 'My_Lp.lp')
lpsolve('solve', lp)
dlp = lpsolve('copy_lp', lp)
lpsolve('dualize_lp', dlp)
lpsolve('write_lp', dlp, 'My_DLp.lp')
```

This will return a file into the same directory as the file that My\_Lp.lp is under the name My\_DLp.lp. Because lpsolve55 was originally written for C++ and then messed with to be workable with Python, the dualize function does not always work properly. In dualizing, sometimes the dual objective function will contain a very large constant, a remainder of the (unnecessary) punishing term that the solver uses to dualize the program. When creating the dual program be sure to check for this constant in the objective function of the new dual program file before attempting to use it.

## IDLEs

Because most people working with .lp and .mps files are generally well versed in computing, the solvers for these file types are usually written in a language closer to machine code. Instead of creating new solvers, a few developers have created IDLEs (integrated development environments) that are far more user friendly. A common freeware IDLE for .lp files is LP\_Solve. The interface of the IDLE is similar to that of the Terminal, but has a command bar that makes creating and solving these files much easier for people that have little to no programming experience. In this IDLE and others like it, a user can write a program to a .lp or .mps file just the same as in a text editor. The difference is that instead of needing to know all the code to call the program and solve it and copy it and dualize it, the IDLE has a command bar that the user clicks on to call commands and view outputs. The downside to this is that these IDLEs are restrictive. They only are written with a limited number of commands and are not full programs that are meant to run scripts. This means that while they are great for solving the programs and getting some of the information about the program, the more in depth information can only be reached by knowing commands for the files in some other programming languages. However, writing the program to a .lp or .mps file is far simpler in many of the IDLEs than it is in Python or C. This is because the IDLE is specific to linear programs so it knows that all of the input is going into the program. On the other hand, writing a program in Python requires telling the machine that the code being written is for the program on each line, and specifying what part of the program the code is going to.

## Solver Issues

Because the majority of students using this site will not have access to the best commercial solvers, some issues with freeware solvers will be discussed. Freeware solvers are generally effective but not optimized and made to work under all circumstances. For example, a popular freeware solver, Lp\_solve, is written in C and then forced into other languages like Python and Matlab. While this means they are cheap to develop and therefore free to the public, they don't necessarily work right in these other languages. This is normally not an issue, as developers make the most important parts of the solver work. However, some of the commands in the base language don't work properly in the secondary language. For example, in C, the command

```
lpsolve(dualize_LP, My_Lp.lp)
```

will create a dual program of the .lp file, and save the .lp as the new dual program. However, in Python the following script is necessary:

```
from lpsolve55 import *

lp = lpsolve('read_LP', 'My_Lp.lp')
lpsolve('solve', lp)
dlp = lpsolve('copy_lp', lp)
lpsolve('dualize_lp', dlp)

lpsolve('write_lp', dlp, 'My_DLp.lp')
```

Even when this is all typed in, the way the commands run in Python can create a dual program with a punishing term in the objective function that makes the program infeasible or unbounded.



# Connor Mattes

From CU Denver Optimization Student Wiki

I am currently a second year graduate student at CU Denver, in the discrete group. I received his undergraduate degree at Colorado School of Mines in computational and applied mathematics, and while there spent a semester studying abroad in the Budapest Semesters in Mathematics program. At Mines I did research on topological Ramsey theory, and while at an REU at Rochester Institute of Technology I did research on a field of graph coloring called  $L(h,k)$  labelling. I currently work with Flo Pfender doing research in flag algebras. When not doing math I like to rock climb, ski, and hike/backpack.

In Fall 2018 for Linear Programming I worked on a problem involving the placement of after school programs with Zachary Sorenson, the link is: [Using After School Activities To Reduce Crime](#).

In Spring 2019 for Integer Programming I worked on a problem on finding the most efficient routes to clean parks with Zachary Sorenson, the link is: [Cleaning Parks for a Safer Future](#)

In Spring 2020 for Network Flows, I worked with Zachary Sorenson (3 for 3!) on: [Min-mean Cycle Cancelling Algorithm and It's Applications](#)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Connor\\_Mattes&oldid=2525](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Connor_Mattes&oldid=2525)"

Category: Contributors

- 
- This page was last modified on 20 April 2020, at 16:38.
  - This page has been accessed 2,130 times.

# Contributors

From CU Denver Optimization Student Wiki

Jordan\_Perr-Sauer

Steffen\_Borgwardt

Christina\_Ebben

Keegan\_Schmitt

Retrieved from "<https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Contributors&oldid=552>"

- 
- This page was last modified on 9 November 2017, at 16:34.
  - This page has been accessed 1,257 times.

# Converting Disused Lots to Housing: Equitable locations for new housing

From CU Denver Optimization Student Wiki

Author: Michael Schmidt

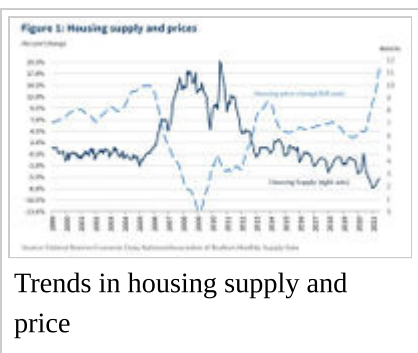
## Contents

- 1 Abstract
- 2 Motivation
- 3 Goals
- 4 Data
  - 4.1 Employment Density
  - 4.2 Parcels
  - 4.3 Walking Network
- 5 Analysis
  - 5.1 Walk Disks
  - 5.2 Job Densities
  - 5.3 Parcels
  - 5.4 Areas of Interest
  - 5.5 Optimization Problem
    - 5.5.1 Problem A
    - 5.5.2 Problem B
- 6 Results
- 7 Code
- 8 Notes

## Abstract

A housing shortage in Denver has hit the city’s most vulnerable hardest. In the greater Denver metro area, an estimated 14-60 thousand homes shortage persists. As a result, candidates for Denver Mayor have suggested using vacant or underutilized land to develop affordable housing. This project seeks to determine ideal locations for conversion based on centers of employment and walkability to grocery stores, schools, and public transit using equitable measures and methods from integer optimization.

# Motivation



During the financial crisis of 2008, the number of new homes being built dropped to the lowest point on record<sup>[1]</sup>. As the population has continued to grow since 2008, the housing supply has not kept pace, resulting in a housing shortage. In the Denver Metro Area, there is an estimated shortage of 14-60 thousand homes<sup>[2]</sup>.

Candidates for Mayor have suggested building affordable housing on city-owned vacant lots<sup>[3]</sup>. However, determining which lots qualify for such a designation and whether such land can be sold is an unsolved problem.

Instead of looking at the city and RTD land, as the candidates suggested, the following analysis looks at vacant private land around the city. Ideally, incentives could encourage land owners to convert the land into housing. The only remaining question is what land to encourage development on.

## Goals

This analysis aims to determine which vacant lots within Denver are suitable for housing development with equity in mind. Access is the foremost concern for housing equity. Transportation is a means to increase access; however, direct access is more advantageous. Therefore, this model assumes all core amenities should be accessible within a 20-minute walk.

Here, a simplifying assumption is made that the transit network cannot be used. Transit networks are difficult to source information for, and connecting it to Open Street Maps is non-trivial (specifically OSMNX, which doesn't encode relations between nodes). Given adequate time, allowing for walk times, including transit, would create a more representative model. Instead of such an analysis, this analysis requires access to the light rail network within a 20-minute walk.

Since the policy suggestion is to encourage the construction of affordable housing through grants or tax breaks, ideally, the total cost of such grants is proportional to the property value. Therefore, the total value of the land on which to encourage construction should be cost minimal.

Therefore, the relevant requirements are:

- Equitable access to amenities, i.e., within a 20-minute walk:
  - Schools
  - Supermarkets
  - Light Rail Stations
- Access to the maximum number of employment opportunities
- Lowest cost

## Data

### Employment Density

The US EPA keeps a dataset of employment densities across census blocks for the US. This can be used to determine the locations of jobs in the Denver metro area.

## Parcels

Source: <https://denvergov.org/opendata/dataset/city-and-county-of-denver-parcels>

Each parcel in Denver is labeled with a category (single family, condo, tower, warehouse, vacant land, etc.), a polygon defining its boundary, and tax information like assessment. This data can be used to extract location data, filter for potential plots (vacant land), and determine the land value (assessment information).

## Walking Network

Open Street Maps has information on walking paths available through the `OSMNx` Python package. Given an average walking speed of 4.8 km/hr, the time between nodes can be determined, and each node within a 20-minute radius can be selected.

## Analysis

### Walk Disks

A 20-minute walk disk can be generated from the OSM walk network to determine if a parcel location satisfies the conditions above.



20-Minute  
walking disk  
from Auraria  
Library

### Job Densities

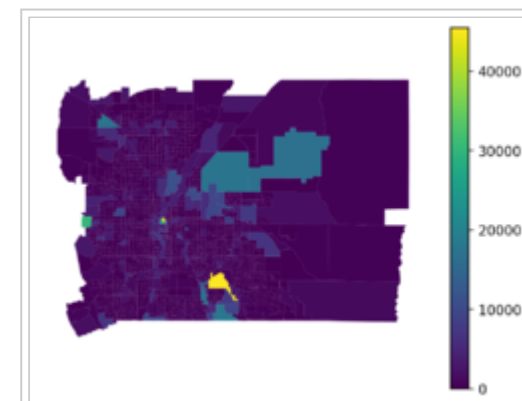
The total jobs in the Denver Metro Area by census tract. Each potential parcel's 20-minute walking disk can be compared to the employment density to determine the number of jobs nearby. While not all jobs are available, it approximately measures job availability in the region.

### Parcels

Denver's parcels are labeled with a `PROP\_CLASS` attribute, where each value lower than 600 is some variant of a vacant parcel. Some of these parcels are vacant but clearly used for some purpose; to further filter these parcels, parcels with plots less than \$300 total assessed value are ignored. It appears public parcels, including several runways, are listed as vacant with a value of \$30.

### Areas of Interest

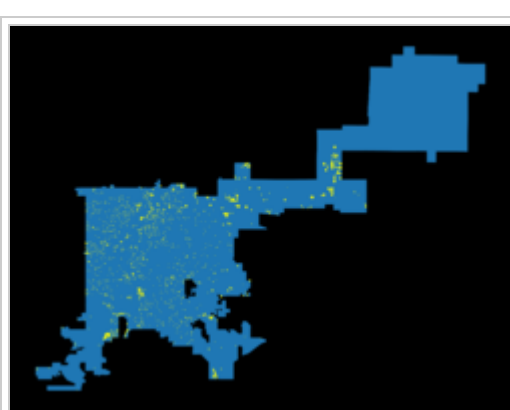
As the only parcels under consideration are those within a 20-minute walk of a Light Rail station, supermarket, and school, the whole of Denver may be reduced to a subset area of interest. Doing this per parcel would be too cumbersome; it is far easier to select each school, light rail station, and supermarket and, compute the 20-minute walk disk, take each union, then take the intersection of all three.



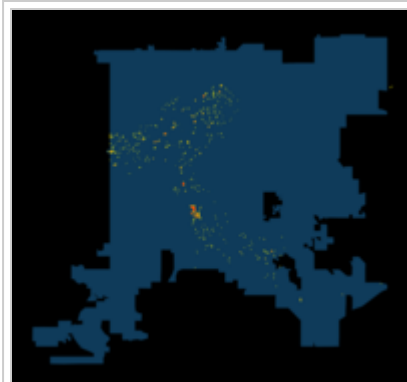
Total jobs in census tracts for Denver  
Metro

The final set of the points are therefore:





Vacant Parcels in Denver



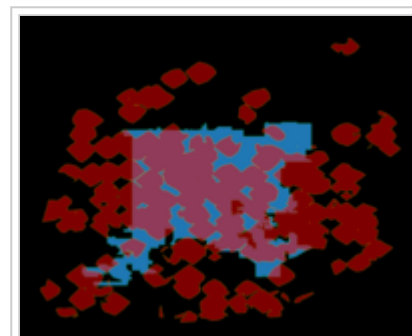
Vacant parcels near Schools,  
LightRail, and Supermarkets.



Areas near LightRail,  
supermarkets, and schools

## Optimization Problem

There are two programs one could use to find which parcels to select in this problem. One is to select some number of parcels to meet some criterion (count or cost, for instance) and solve with an implementation of the knapsack problem. This can lead to many parcels being chosen in the same neighborhood. While this outcome could be ideal in many situations, it could also create adverse competition for local jobs. To address this, another program could be to find an optimal covering of the employment density (i.e. cover the maximal number of jobs) while minimizing the amount of overlap between walk disks of each parcel.



Areas near supermarkets and  
schools

## Problem A

$P$  : Set of parcels

$x_p$  : Indicator variables for parcel  $p$ . Selects whether the parcel is selected or not.

$c_p$  : Cost of parcel  $p$

$v_p$  : Capacity, in units, of a parcel  $p$

$e_p$  : Employment Coverage of parcel  $p$

$D$  : Total demand in terms of units

$\alpha, \beta > 0$  : Weights for cost vs employment coverage

Here the problem has the form

$$\begin{array}{ll} \min & \sum_{p \in P} \alpha x_p \cdot c_p - \beta x_p \cdot e_p \\ \text{with the program} & \\ \text{s.t.} & \sum_{p \in P} x_p \geq D \end{array}$$

### Problem B

Here the problem has the form

- $P$  : Set of parcels
- $x_p$  : Indicator variables for parcel  $p$ . Selects whether the parcel is selected or not.
- $c_p$  : Cost of parcel  $p$
- $v_p$  : Capacity, in units, of a parcel  $p$
- $\theta_{jk}$  : Overlap of walking disks for parcels  $j$  and  $k$
- $\alpha$  : Allowable overlap between two different parcel's disks
- $e_p$  : Employment Coverage of parcel  $p$
- $D$  : Total demand in terms of units
- $\alpha, \beta > 0$  : Weights for cost vs employment coverage

with the program

$$\begin{aligned} & \min \sum_{p \in P} \alpha x_p \cdot c_p - \beta x_p \cdot e_p \\ & \text{s.t.} \sum_{p \in P} x_p \geq D \\ & \quad \forall j, k \in P : j \neq k : x_j x_k \theta_{jk} < \alpha \end{aligned}$$

Unfortunately, computing the relative overlaps between all pairs of walking disks for each vacant lot in the candidate area is computationally infeasible on the resources at my disposal. (I plan to revisit this with some better methods).

## Results

The solution to problem A for 1000 units can be seen in the following map.



## Code

The code can be found here: <https://github.com/schmidmt/D2P-Disused-to-Housing>

## Notes

1. ↑ U.S. Census Bureau and U.S. Department of Housing and Urban Development, New Privately-Owned Housing Units Started: Total Units [HOUST], retrieved from FRED, Federal Reserve Bank of St. Louis; <https://fred.stlouisfed.org/series/HOUST>, May 1, 2023.
  2. ↑ <https://commonsenseinstitute.co/colorado-housing-affordability-update-november-2022/>
  3. ↑ [https://denvergazette.com/politics/elections/denver-mayoral-candidates-leslie-herod-tom-wolf-agree-homeless-must-be-housed-but-disagree-how/article\\_938af810-b30e-11ed-a7a3-fbe8e4ec47a1.html](https://denvergazette.com/politics/elections/denver-mayoral-candidates-leslie-herod-tom-wolf-agree-homeless-must-be-housed-but-disagree-how/article_938af810-b30e-11ed-a7a3-fbe8e4ec47a1.html)
- Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Converting\\_Disused\\_Lots\\_to\\_Housing:\\_Equitable\\_locations\\_for\\_new\\_housing&oldid=4366](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Converting_Disused_Lots_to_Housing:_Equitable_locations_for_new_housing&oldid=4366)"

- This page was last modified on 4 May 2023, at 09:33.
- This page has been accessed 94 times.

# Convex Hull Finding Algorithms

From CU Denver Optimization Student Wiki

Given a finite set  $S \subset \mathbb{R}^d$ , a **convex hull finding algorithm** is a procedure that generates a description of the convex hull of  $S$ . Finding quick ways of generating descriptions for the convex hull of a set is useful applications such as Geographical Information Systems (GIS), robotics, visual pattern matching, and finding integer hulls. Depending on the algorithm and dimension of the set  $S$ , the convex hull finding algorithm could give an half plane description and/or vertex description of the convex hull. There are many convex hull finding algorithms such as Gift Wrapping, Graham's Scan, and Chan's algorithm which are specialized for  $\mathbb{R}^2$  and  $\mathbb{R}^3$ . Other algorithms such as Quickhull work well for higher dimensions. This article provides summary descriptions for some planar convex hull finding algorithms. All descriptions ignore degenerate cases where the input is too small.

## Contents

- 1 Gift Wrapping
- 2 Graham's Scan
- 3 Chan's Algorithm
- 4 Quickhull
- 5 References

## Gift Wrapping

**Gift Wrapping** (Jarvis 1973) is an iterative convex hull finding algorithm for  $\mathbb{R}^2$  which produces both a  $V$ -description and  $H$ -description of  $\text{conv}(S)$ . It runs in  $O(nh)$  where  $n$  is the number of points and  $h$  is the number of vertices in  $\text{conv}(S)$ .

Assume no 3 points co-linear. A modification to the algorithm can account for colinear points. Begin by finding  $P_0$ , the left most point of  $S$ , which is a vertex of  $\text{conv}(S)$ . At a given step, find point  $P_{i+1}$  such that all elements of  $S$  are to the right of the line defined by  $P_i, P_{i+1}$ . Add  $P_{i+1}$  to the description and repeat this process until  $P_{i+1} = P_0$ . The **orientation** of a three points  $a, b, c$  is the direction of travel from  $a$  to  $b$  to  $c$  is either a clockwise, counterclockwise, or colinear. The direction is determined by the sign of  $\vec{ab} \cdot \vec{bc} = ||\vec{ab}|| \cdot ||\vec{bc}|| \cos(\theta)$ . If the sign is positive, then the three points are oriented in a clockwise direction, counterclockwise if the sign is negative, colinear if the dot product is 0. To test if a point  $s$  is to the right of a line defined by  $P_i, P_{i+1}$ , the **orientation** is typically used. To find  $P_{i+1}$  such that all elements of  $S$  are to the right of the line  $P_i, P_{i+1}$  is to search for  $P_{i+1}$  such that the triple  $P_i, s, P_{i+1}$  is oriented counterclockwise for all  $s$  that are not  $P_i, P_{i+1}$ . This can be done by iterating over all of  $S$  and updating a guess for  $P_{i+1}$  when the triple  $P_i, s, P_{i+1}$  is oriented clockwise. The complexity for Gift Wrapping is  $O(nh)$  because for each vertex there is an  $O(n)$  method to find the vertex as described above.

# Graham's Scan

Graham's Scan (Graham 1972) is an iterative algorithm for  $\mathbb{R}^2$  which finds both a vertex description and half plane description  $conv(S)$  and runs in  $O(n \log(n))$  where  $n$  is the size of the input.

Assume that no 3 points are colinear. Simple modifications to the algorithm can be made to account for colinear points. Begin by selecting the bottom left most point,  $P_0$ , a vertex, of  $conv(S)$ . Sort the points in  $S$  according to the angle the form from  $P_0$  relative to the  $x$ -axis from least to greatest. Starting from the inital point  $P_0$ , iterate over this order. Let  $P_i$  be the most recent point found on the convex hull. Let  $s_{next}$  and  $s_{future}$  be the next two points in the order. If  $P_i, s_{next}, s_{future}$  are oriented counterclockwise then  $s_{next}$  is apart of the convex hull. Add  $s_{next}$  to the convex hull. Otherwise, remove  $s_{next}$  from the order and repeat until  $s_{next} = P_0$ .

To find the angle of each point  $s$  to  $P_0$  relative to the  $x$ -axis, use a dot product to compute  $\cos(\theta)$ . Since each angle is relative to the  $x$  axis and  $P_0$  is the bottom left most point,  $\theta \in [0, \pi]$  and thus  $\cos(\theta) \in [-1, 1]$  and can be ordered from greatest to least which corresponds to ordering from the least to greatest angle. The complexity of Graham's Scan is  $O(n \log(n))$  because any sorting algorithm which runs in  $O(n \log(n))$  can be used to sort the angles. All other steps can be accomplished with a method that is  $O(n)$  time.

## Chan's Algorithm

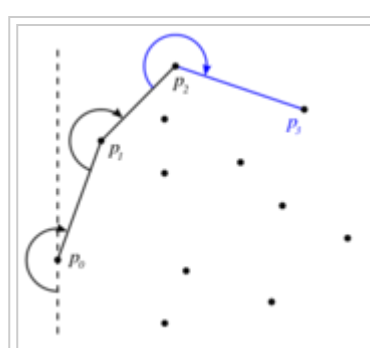
Chan's algorithm (Chan 1996) is a divide and conquer approach that combines Graham's Scan and Gift Wrapping. Chan's algorithm has two phases. The first phase divides  $S$  into equally sized subsets and computes the convex hull of each one. The second phase uses the computed convex hulls to find  $conv(S)$ . The algorithm is output sensitive and runs in  $O(n \log(h))$ .

**Phase 1:** Let  $H$  be given. Partition  $S$  into  $S_1, S_2, \dots, S_j$  such that  $|S_k| \leq H$  and  $j = \lceil \frac{n}{H} \rceil$ . Then for each partition compute  $conv(S_k)$  using Graham's Scan.

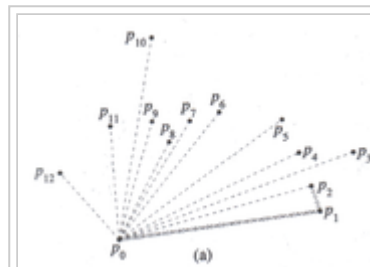
**Phase 2:** Use a modification of Gift Wrapping and the precomputed convex hulls to compute the first  $H$  elements of the  $conv(S)$ . If the Gift Wrapping fails to complete, then increase  $H$ .

The Gift Wrapping modification is as follows. Instead of searching all of  $S$ , we search for  $P_{k,i}$ , a vertex of such  $conv(S_k)$ , that  $P_{i-1}, P_i, P_{k,i}$  forms the greatest angle. Since the vertices of  $S_k$  are ordered, a binary search and testing orientation can be used to find  $P_{k,i}$ .

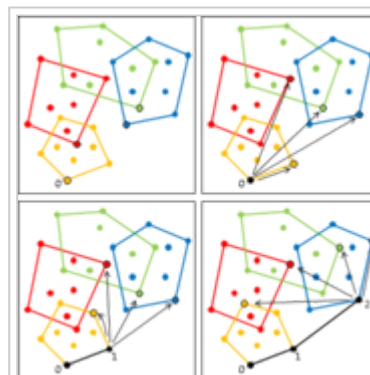
Phase 1 runs in  $O(n \log(H))$  as does Phase 2. Ideally, we want to pick  $H = h$  but we do not know the number of vertices of the  $conv(S)$ . If we increase  $H$  too slowly, the algorithm becomes cumbersome and too quickly we fail to harness the advantage of the algorithm. By picking  $H = \min\{n, 2^{2^t}\}$  for  $t = 0, 1, 2, \dots$  we can ensure  $O(n \log(h))$ . The algorithm terminates for some  $t$  when  $2^{2^{t-1}} < h \leq 2^{2^t}$  thus



A visualization of 2D Gift Wrapping Algorithm.



The sorting phase in Graham's Scan.



A demonstration of Phase 2 in Chan's Algorithm.

$2^{t-1} < \log(h)$ . Since Chan's algorithm occurs for  $t$  instances, the total running time is

$$\sum_{k=0}^i O(n \log(2^{2^k})) = \sum_{k=0}^i O(n \cdot 2^k) = O(n(2^{i+1} - 1)) = O(n(2^{i+1})) < O(4n \log(h)) = O(n \log(h)).$$

## Quickhull

Quickhull (Barber, Dobkin, Huhdanpaa 1996) is a divide and conquer algorithm. It works well for any dimension  $d$ . For  $d = 2$ , it has worst case run time  $O(n^2)$  and average case  $O(n \log(n))$ . The algorithm as described will be for  $d = 2$ .

Find 3 points,  $p_1, p_2, p_3$ , that are known to be vertices of  $\text{conv}(S)$ . This partitions  $S$  into 4 subsets. The points inside the triangle, and points above (to the left of) the line  $p_i, p_j, i \neq j$ . Points inside the triangle can be ignored as they are not vertices of  $\text{conv}(S)$ . If there are points above the line  $p_i, p_j, i \neq j$  find the point furthest from the line. Add this to the description of the convex hull. This creates a new triangle with any points inside of it being apart of the current description of  $\text{conv}(S)$ . With the two new edges, if possible repeat the procedure of find a point above the line and furthest away. Once no points are above any edge, the algorithm terminates.

Note a triangle is a  $2 + 1$  simplex and a line formed by two points in the plane is a facet of this. To generalize this procedure, use a  $d + 1$  simplex and instead check if points are 'above' a facet of this simplex and find the point that is furthest from this facet.

## References

- B.C. Barber, D.P. Dobkin, H. Huhdanpaa(1996). "The quickhull algorithm for convex hulls" PDF (<http://dpd.cs.princeton.edu/Papers/BarberDobkinHuhdanpaa.pdf>)
- R. Binbin, C.Cheng, Y. Jie, J. Qi, "Applications" Web page ([http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg\\_ss\\_07w/Webprojects/Chen\\_hull/applications.htm](http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg_ss_07w/Webprojects/Chen_hull/applications.htm)).
- R. L. Graham (1972), "An effcient algorithm for determining the convex hull of a finite planar set."
- M. Homann (2009), "Chan's Algorithm - Lecture Notes" PDF (<https://www.ti.inf.ethz.ch/ew/lehre/CG09/materials/v2.pdf>)
- S. Ramaswami (1993), "Convex Hulls: Complexity and Applications (a Survey)" PDF ([http://repository.upenn.edu/cgi/viewcontent.cgi?article=1272&context=cis\\_reports](http://repository.upenn.edu/cgi/viewcontent.cgi?article=1272&context=cis_reports))
- A. C. Yao (1981), "A lower bound to finding convex hull". PDF (<http://infolab.stanford.edu/pub/cstr/reports/cs/tr/79/733/CS-TR-79-733.pdf>) Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Convex\\_Hull\\_Finding\\_Algorithms&oldid=536](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Convex_Hull_Finding_Algorithms&oldid=536)"

- This page was last modified on 3 May 2017, at 12:00.
- This page has been accessed 21,042 times.

# Coordinating Response to Fatal Accidents

From CU Denver Optimization Student Wiki

## Abstract

Denver Police and Fire Departments have separate dispatch systems that can only track their respective fleets. Because of this, dispatchers rarely have the opportunity to communicate to the other dispatch entity about their deployment decisions. Given that potentially fatal car accidents require a response from both police and fire units, we can better coordinate response to these incidents using a Many-to-One matching problem and the locations of previous fatal car accidents in Denver County. Our results offer a set of “suggested partnerships” of police and fire units given the location of a potentially fatal car crash. Deploying units with a set of “suggested partnerships” can improve the response to accidents; if the unit deployed from one entity is aware of the other’s activities, every responder will be better prepared to deal with life threatening emergencies.

## Model and Description

We use Integer Programming to construct a Many-to-One matching between police stations and fire stations. We model this by assigning a binary variable to each pair that can be possibly matched. If a pair is matched, the variable has value 1. If the pair is not matched, the variable has value 0. We sum the variables of the pairs involving a certain fire station, and set that sum to 1, in order to ensure that every fire station is matched to exactly one police station. Similarly, we can set the sum of the variables of the pairs involving a certain police station to be at least 1 to ensure that every police station is matched to at least one fire station. We additionally apply a cost to each matching based on how synchronized those two stations would be responding to accidents based on past accident data. We then look for a matching satisfying these conditions which minimizes that cost.

$$\begin{aligned} &\text{Minimize} && \sum_{i \in P, j \in F} c_{ij} x_{ij} \\ &\text{Subject to} && \sum_{i \in P} x_{ij} = 1 \quad \forall j \in F \\ &&& \sum_{j \in F} x_{ij} \geq 1 \quad \forall i \in P \\ &&& x_{ij} \in \{0, 1\} \quad \forall i \in P, j \in F \end{aligned}$$

Where  $c_{ij} = \sum_{c \in C} |dist(c, i) - dist(c, j)|$

$P$  is the set of police stations

$F$  is the set of fire station

$C$  is the set of all crashes we are considering.

## Code, Poster and Presentation Slides

The AMPL code used, a PDF of the poster, and the PowerPoint presentation slides are in a Github repository here ([https://github.com/eric-d-culver/D2P\\_Sp\\_2019](https://github.com/eric-d-culver/D2P_Sp_2019)).

Project by Eric Culver and Christina Ebben

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Coordinating\\_Response\\_to\\_Fatal\\_Accidents&oldid=2012](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Coordinating_Response_to_Fatal_Accidents&oldid=2012)"

- 
- This page was last modified on 3 May 2019, at 11:38.
  - This page has been accessed 1,707 times.



# Counting Eulerian Cycles in Graphs

From CU Denver Optimization Student Wiki

## Project Contributors

Alex Semyonov ([http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Alex\\_Semyonov](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Alex_Semyonov))

Michael Boyce ([http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Michael\\_Boyce](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Michael_Boyce))

This project is a collaborative effort between Alexander Semyonov and Michael Boyce. This project was completed as an assignment for Steffen Borgwardt's Applied Graph Theory course at the University of Colorado Denver. Each contributors page is linked above.

## Abstract

In this project, we will investigate how to count the number of Eulerian circuits in a graph. We begin by introducing some terminology to explore necessary conditions for an Eulerian circuit to exist in a graph, or digraph. The most important of which is that in-degree equals out-degree for every vertex in a digraph. We shift our attention to Eulerian circuits in digraphs. This shift leads us the B.E.S.T theorem (van Aardenne-Ehrenfest, de Bruijn, Tutte-Smith). The B.E.S.T theorem gives a formula for counting the number of Eulerian circuits in an Eulerian digraph. We take a look at the B.E.S.T. theorem and ultimately prove this theorem by partitioning the space of all Eulerian cycles into disjoint classes (using rooted in-trees to create this partition) and then establishing that each class must in fact contain the same number of circuits.

After proving the BEST theorem, we investigate the problem of counting circuits in undirected graphs. This is a much more difficult problem, as it is a #P problem. We will briefly discuss some difficulties associated with undirected graphs and highlight why the B.E.S.T theorem does not generalize to undirected graphs.

The Github below contains our slide deck.

Github link: <https://github.com/asems99/Counting-Eulerian-Cycles-in-Graphs>

## References

1. Douglas B. West. Combinatorial Mathematics, 2021. (In particular, section 15.2)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Counting\\_Eulerian\\_Cycles\\_in\\_Graphs&oldid=5030](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Counting_Eulerian_Cycles_in_Graphs&oldid=5030)"

- 
- This page was last modified on 12 May 2025, at 16:22.
  - This page has been accessed 39 times.

# Courtney Franzen

From CU Denver Optimization Student Wiki

## Contents

- 1 Contact Links
  - 1.1 Education
- 2 Professional Life
  - 2.1 Career
  - 2.2 Programming Languages/Experience
  - 2.3 Projects

## Contact Links

- Email (<mailto:courtney.franzen@ucdenver.edu>)

## Education

1. University of Central Arkansas, B.S., General Science with Math Minor
2. University of Central Arkansas, M.A., Mathematics Education
3. Air Force Institute of Technology, M.S., Operations Research

## Professional Life

### Career

- Pet Store Associate - The Fish Bowl
- Graduate Teaching Assistant - University of Central Arkansas
- Operations Research Analyst - United States Air Force

## Programming Languages/Experience

- MATLAB
- Python

## Projects

In the Fall of 2023 for Linear Programming, she worked on Housing Assistance Program Allocation. This project utilized a knapsack integer program to overcome challenges in Denver regarding housing assistance program awareness and distribution of resources.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Courtney\\_Franzen&oldid=4568](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Courtney_Franzen&oldid=4568)"

Category: Contributors

- 
- This page was last modified on 5 December 2023, at 09:05.
  - This page has been accessed 25 times.

# Craig Friehauf

From CU Denver Optimization Student Wiki

I am currently a graduate student at the University of Colorado Denver studying Computer Science.

I enjoy taking on challenges that have a high likelihood of failure. When I am not working on a project, I enjoy golf and have recently started competing in Triathlons.

Minimum Planar Crossing Number

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Craig\\_Friehauf&oldid=393](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Craig_Friehauf&oldid=393)"

Category: Contributors

- 
- This page was last modified on 25 April 2017, at 16:05.
  - This page has been accessed 1,631 times.

# Creating Fair Voting Districts

From CU Denver Optimization Student Wiki

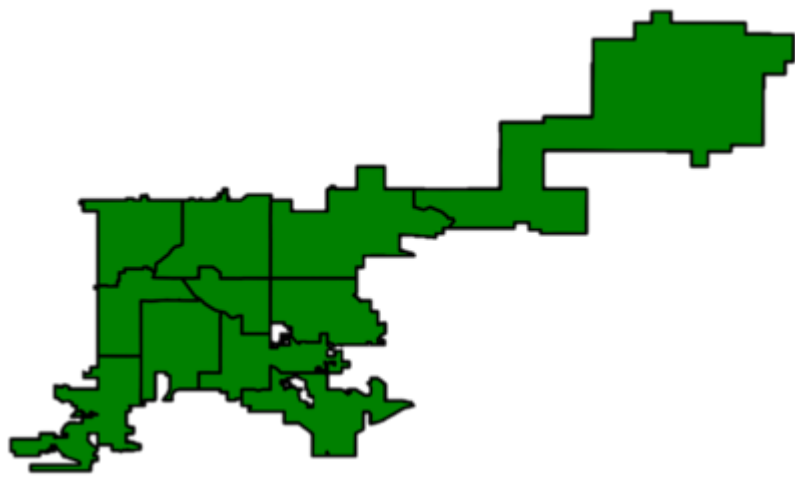
This project is by Angela Morrison and Weston Grewe

## Contents

- 1 Overview
- 2 Mathematics
  - 2.1 Constrained Clustering
  - 2.2 Power Diagram
- 3 Implementation
  - 3.1 Census Blocks
  - 3.2 Voting Locations
- 4 Results
- 5 Moving Forward
- 6 GitHub
- 7 Presentation Materials
- 8 References
  - 8.1 Papers:
  - 8.2 Census Data:
  - 8.3 Census Data Key:
  - 8.4 City Council Map:

## Overview

Electoral districts are a feature of democratic governments. Districts must be redesigned often as demographics change. In the U.S. districts are redrawn every ten years with the arrival of a new census. The process of redrawing districts can be highly contentious and politically motivated. Standards for fair districts have been discussed at length in political science and a few key concepts of what constitutes fair. First and foremost, every district should have roughly the same number of people to ensure equal representation for each elected official. Second, a district should be simply connected. In most cases, two people who live close to one another should also be in the same district, of course, there are exceptions for the boundary. Additionally, districts should be sufficiently regular in the sense that they look like convex polygons and the maximum and minimum diameter of the set is similar. In our project, we implement a method, using linear programming, to design city council districts for the city of Denver that contain the same number of people and are convex polygons with the exception of the boundary of the city. We begin by using  $k$  — constrained clustering to assign residents to voting districts. We then find the power diagram which yields the lines that define our voting districts.



## Mathematics

In this section, we present the two linear programs used to create the voting districts. The first linear program provides an assignment of the blocks. The second linear program produces values needed to create a power diagram for the districts.

### Constrained Clustering

To assign residents to voting districts we use fractional  $k$  — constrained clustering. In the context of our problem the data required for this clustering is the locations and populations of each census block and locations for a number of sites, in our case we used polling locations in Denver, and in particular chose 11 polling locations since there are 11 city council districts. Each site can be thought of as a base point for a district, then census blocks near a site are assigned to that site. Finally, all blocks assigned to a particular site are grouped together as a voting district. In this project, we go one step further and allow for fractional clustering, this allows each census block to be assigned to more than 1 site. In practice, few blocks are assigned to multiple sites and this is only necessary to ensure that every district has the exact same number of residents. Fractional clustering could be avoided by relaxing the requirement that every district must have the same number of residents and instead decide on an acceptable range. Now, let's dive into the mathematics. Assume you have a list  $B = (b_1, b_2, \dots, b_k)$  of census blocks where each  $b_i$  is the location of that census block, in our project this is a lat/long value. Additionally, assume you have a list  $W = (w_1, w_2, \dots, w_k)$  where  $w_i$  is the population of  $b_i$ , let  $w = w_1 + \dots + w_k$ , the total population of the city. Finally, assume you have a list  $S = (s_1, s_2, \dots, s_n)$  of site locations. To find electoral districts, we need only solve the following linear program:

$$\begin{aligned} \min \quad & \sum_{i,j} x_{i,j} w_j \|b_j - s_i\|^2 \\ \text{s.t.} \quad & \sum_i x_{i,j} = 1, x_{i,j} \in [0, 1] \\ & \sum_j x_{i,j} w_j = \frac{w}{n} \end{aligned}$$

The variables  $\boldsymbol{x}_{i,j}$  correspond the proportion of block  $b_j$  assigned to site  $\boldsymbol{s}_i$ . The squaring of the norm guarantees that the districts will be polygonal. If we do not care about polygonal districts, the squared-norm can be replaced with any distance-like function. Additionally, the squared distance also punishes districts being assigned to a voting location that is not their closest location. This helps ensure a reasonable looking district. The first constraint is what forces the  $\boldsymbol{x}_{i,j}$  to act like proportions, it tells us that the entirety of a census block is assigned to some number of districts and it is not over-assigned. The second constraint ensures that each district will have the same number of people. We have  $n$  districts and a total population of  $w$ , thus each district should have  $w/n$  number of people. Summing over  $j$  simply counts the number of people from each census block assigned to site  $i$ .

## Power Diagram

The power diagram for our clustering is illustrated below. Mathematically, the power diagram is formed from the separating hyperplanes between every pair of clusters. Suppose our census blocks have already been clustered, denote this clustering  $C = \{C_1, \dots, C_k\}$ . A separating hyperplane for the pair of clusters  $(C_i, C_j)$  can be specified by a pair  $(\boldsymbol{s}_{i,j}, \gamma_{i,j})$  where we have  $C_i \subseteq \{\boldsymbol{x} : \boldsymbol{s}_{i,j}^T \boldsymbol{x} \leq \gamma_{i,j}\}$  and  $C_j \subseteq \{\boldsymbol{x} : \boldsymbol{s}_{i,j}^T \boldsymbol{x} \geq \gamma_{i,j}\}$ . It is often the case that the pairing  $(\boldsymbol{s}_{i,j}, \gamma_{i,j})$  is not unique, however, that is alright. We can get closer to uniqueness by seeking a so-called maximum margin power diagram. In this case, we seek a set of pairings  $\{(\boldsymbol{s}_{i,j}, \gamma_{i,j})\}_{i,j}$  that is a solution to the linear program

$$\begin{aligned} & \max \epsilon \\ \text{s.t. } & \boldsymbol{s}_{i,j}^T \boldsymbol{x}_\ell + \epsilon \leq \gamma_{i,j}, \boldsymbol{x}_\ell \in C_i \\ & \gamma_{i,j} = -\gamma_{j,i} \end{aligned}$$

So far, what the  $\boldsymbol{s}_{i,j}$  and  $\gamma_{i,j}$  are has been skirted over. We will first discuss how the  $\boldsymbol{s}_{i,j}$  are formed, then discuss the  $\gamma_{i,j}$ . Recall, our collection of site locations:  $S = (\boldsymbol{s}_1, \boldsymbol{s}_2, \dots, \boldsymbol{s}_n)$ . The vector

$$\boldsymbol{s}_{i,j} = \frac{\boldsymbol{s}_j - \boldsymbol{s}_i}{\|\boldsymbol{s}_j - \boldsymbol{s}_i\|}.$$

Therefore,  $\boldsymbol{s}_{i,j}$  may be interpreted as the normalized direction from site  $\boldsymbol{s}_i$  to site  $\boldsymbol{s}_j$ . In the formulation of the hyperplane separating the cluster based at site  $\boldsymbol{s}_i$  from site  $\boldsymbol{s}_j$ , the vector  $\boldsymbol{s}_{i,j}$  implies that the hyperplane is orthogonal to the direction  $\boldsymbol{s}_{i,j}$ . With this in mind, the value  $\gamma_{i,j}$  is a term that adjusts the position of the hyperplane. It is calculated by ensuring that cluster  $C_i$  is on one side of the hyperplane defined by  $(\boldsymbol{s}_{i,j}, \gamma_{i,j})$  and  $C_j$  is on the other side. Furthermore, the linear program yields a power diagram of maximum margin  $\epsilon$ . This ensures that the hyperplane is not too close to either cluster, on either side it is separated by this  $\epsilon$ .

We can make one further simplification to the linear program. In our constraints we have  $\boldsymbol{s}_{i,j}^T \boldsymbol{x}_\ell \leq \gamma_{i,j}$  for all  $\boldsymbol{x}_\ell \in C_i$ . It will suffice to set  $\sigma_{i,j} = \max\{\boldsymbol{s}_{i,j}^T \boldsymbol{x}_\ell : \boldsymbol{x}_\ell \in C_i\}$  and then solve the following linear program to determine  $\{\gamma_{i,j}\}$ .

$$\max \epsilon$$

$$\begin{aligned} \text{s.t. } \sigma_{i,j} + \epsilon &\leq \gamma_{i,j} \\ \gamma_{i,j} &= -\gamma_{j,i} \end{aligned}$$

The presented linear program can be solved by specifying clusters and site locations easily with a machine. We used Python and SciPy.Optimize to find a solution. The program can be found in the GitHub link below for city council districts for the city of Denver.

## Implementation

For this project, we primarily used Python and Jupyter Notebooks. The library SciPy provides a linear programming tool, optimize.linprog, that can solve linear minimization programs given coefficients of optimization and sets of equality and inequality constraints. Our work can be found in the GitHub link below.

Before we could put our data into the linear program, we had to make some adjustments in order to solve the problem we had stated. In particular, the census blocks and in-person voting locations needed to be cleaned up a bit before being manipulated into the correct form for the linear program in Python.

### Census Blocks

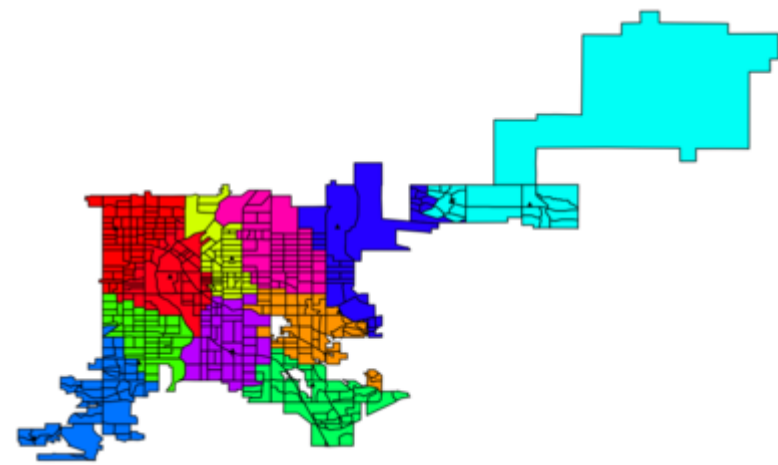
The way the census block data comes in gives the geometry of the borders of each census block. While this is useful for plotting this information, it can make it difficult when computing distances between census blocks and voting locations. Therefore, the center of each census block needed to be found in order to use that point when calculating distance. Doing this does not change the project much as locations within the block would not have to travel much farther or shorter than the center of each block. Another aspect of each census block that needed to be accounted for was the population. There were some blocks that, according to the dataset, had a population of zero people. While this is certainly just a case of bad data, these census blocks need to be removed from this process. We did this because having a population of zero meant that these census blocks would be assigned to each of the new districts being created, and would therefore throw off the calculations of the hyperplanes for the power diagram.

### Voting Locations

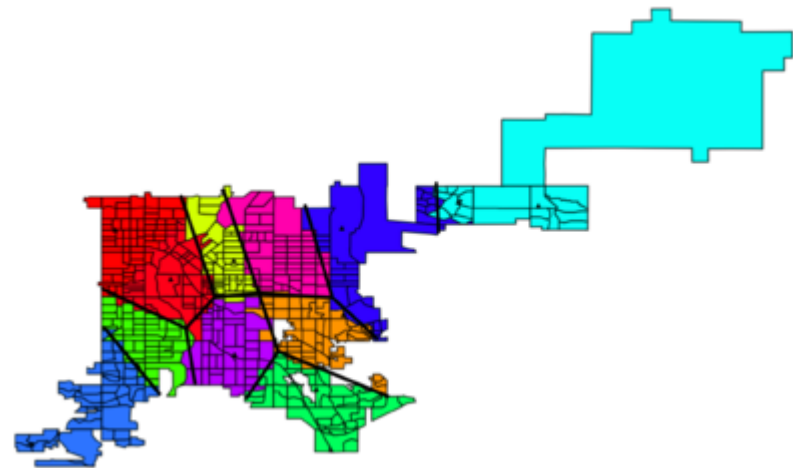
Due to the fact we were trying to construct 11 new districts, we needed to adjust the voting locations used for this project. If we had used our entire set of voting locations, we end up with the same number of districts as voting locations which is not as helpful for the overall goal of this project. To narrow down our choices of voting locations, we performed a K-means clustering of the voting locations to create the same number of clusters as districts. Now, we could not just pick a single voting location in each cluster as this might contain some bias and would mean that our results change depending on which location in the cluster we were to choose. To remove this concern, we decided to create a new "voting location" by using the center point of each cluster as the "sites" in our linear programs. This removed any bias we might have in choosing voting locations while still allowing us to have ach cluster represented as a possible site in our linear program.

With the data properly formatted, it is ready to be plugged into the linear program that determines the voting districts. The code is modular to allow for other data sets and will run regardless of the number of census blocks or site locations. The limitation is only your computers processing power. The first linear program outputs a fractional cluster of all census blocks. The clustering is returned through a matrix whose rows index site locations and columns index census blocks. The  $i, j$  entry is the proportion of the  $j^{th}$  census block assigned to the  $i^{th}$  site location.





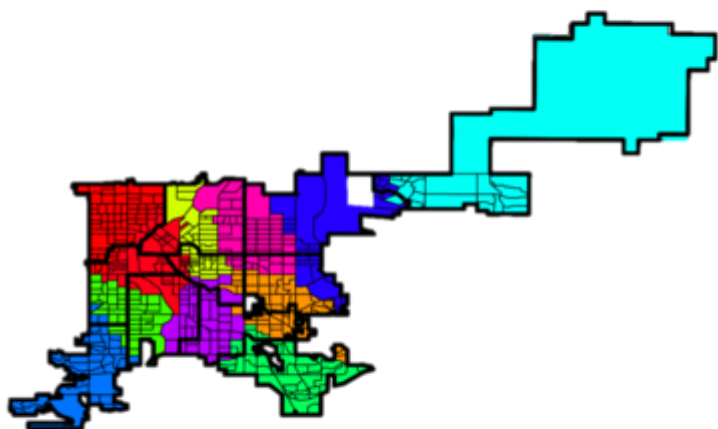
With this clustering, the power diagram can now be computed. Again, the code is modular so it will run for a different set of census blocks and site locations. The functions that run implement the simplified maximum margin separating power diagram found in page 12 of the paper linked below by Dr. Borgwardt. The program returns a collection of triples, one triple for each pair of site locations, that defines the separating hyperplane. With these triples, a hyperplane can be graphed in Python by setting a range for the  $x$  variable and then computing the corresponding  $y$  values that satisfy the hyperplane equation.



## Results

The current city council districts look like:

Our program pulled from 481 census blocks to create the following districts (Colored), note that the current districts are overlaid on the image:



## Moving Forward

Our work does not have to be limited to the city of Denver. It could be applied to any region so long as there is a population count and a number of specified districts. We used polling locations to determine our districts, however, any landmark could be used. This work could be applied to the U.S. House of Representatives by using electoral precincts and determining centers of districts by calculating arithmetic means of current districts.

Moreover, implementing a program that incentivizes a fair distribution of seats is another path to explore. In particular, if some U.S. state has 10 congressional seats and 60% of the population is Democratic and the other 40% is Republican than a fair distribution of seats may be 6 Democratic and 4 Republicans. Many states fail this for a number of reasons. Some states are gerrymandered, others it may just be hard to find any region of the state where the statewide minority is a local majority. California and Alabama are good examples of this phenomenon.

## GitHub

[https://github.com/DillWithIt77/D2P\\_Voting](https://github.com/DillWithIt77/D2P_Voting)

# Presentation Materials

Introductory Slides: <https://docs.google.com/presentation/d/173VWiBv43VMpyjKLuRN3FfMPVyyM0TyuaPJW0AkTK-Q/edit?usp=sharing>

Slides for Linear Programming Presentation/D2P: [https://docs.google.com/presentation/d/1DXEv6PCl5WqQ\\_z\\_nX\\_-pfgOnY0xQK5OjOS6DSEucSP0/edit?usp=sharing](https://docs.google.com/presentation/d/1DXEv6PCl5WqQ_z_nX_-pfgOnY0xQK5OjOS6DSEucSP0/edit?usp=sharing)

## References

### Papers:

Borgwardt, S. On Soft Power Diagrams. J Math Model Algor 14, 173–196 (2015).

Andreas Brieden, Peter Gritzmam, Fabian Klemm, Constrained clustering via diagrams: A unified theory and its application to electoral district design, European Journal of Operational Research, Volume 263, Issue 1, 2017

### Census Data:

<https://data.colorado.gov/Demographics/Census-Block-Groups-in-Colorado-2017/ty5m-9xub>

### Census Data Key:

<https://data.colorado.gov/Demographics/Census-Field-Descriptions/qten-sdpn/data>

### City Council Map:

<https://www.denvergov.org/maps/map/councildistricts>

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Creating\\_Fair\\_Voting\\_Districts&oldid=3084](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Creating_Fair_Voting_Districts&oldid=3084)"

- 
- This page was last modified on 7 December 2020, at 13:27.
  - This page has been accessed 1,134 times.

# Crime & Temperature: Scheduling Awareness Programs

From CU Denver Optimization Student Wiki

## Contents

- 1 Abstract
- 2 Questions
- 3 Correlations
- 4 Social Awareness Program
- 5 Optimizing the Date
- 6 Allocating Budgets
- 7 Future Work
- 8 Bibliography

## Abstract

Many patterns of crimes distribute unevenly throughout the year, and people’s emotions and behaviors are highly influenced by ambient temperatures. In this research project, we have studied the association between the number of incidences of six different types of crimes and the average temperature of the corresponding month, based on the local data of Denver. We have found that there is clearly a positive linear relationship between the two variables, especially in aggravated assault.

To best prepare citizens with the knowledge to minimize their risk of becoming victims of crimes, many schools and communities have initiated “crime prevention/awareness” programs. By introducing the concept of "social memory" of such programs and hypothesizing a formula to model the decay process of the "social memory", we have used AMPL to calculate the optimized budgets and months of the year to host the crime awareness programs based on the constraint-free-scenario and certain scenarios with constraints.

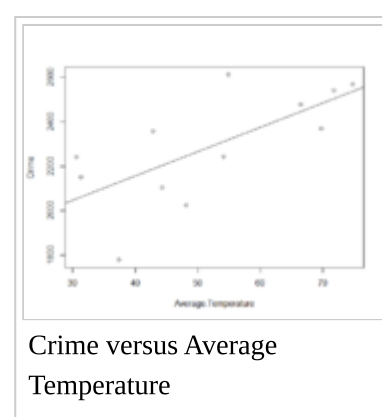
## Questions

Is there a positive correlation between crime and temperature? If yes, how strong the correlation is? Many schools and communities have initiated "crime prevention/awareness" programs. But how to determine the best time and budget allocation for hosting the awareness program?

## Correlations

The crime data from 2015, 2016, and 2017 will be used to analyze the relationship between crime rates and average temperature. The correlation between average temperature and different types of crimes were moderate. Correlation coefficients just measures the strength of the relationship between crime and average temperature. Thus, combining  
Typesetting math: 100% at the correlation between temperature and crime gave us a nearly strong Pearson correlation coefficient of 0.69.

However, it is important to emphasize that one does not necessarily cause the other. It is possible that in the summer time when the temperature is warmer, more people are in vacations out from the city. It is also possible that majority of people spent their summer time just inside their home watching television which made them less vulnerable to violent crimes. But, there are numerous standard researches done around the world studying crime rate and temperature. Most of the researches reached to a conclusion that crime rates are correlated with warmer weather. "The warmer the weather, the more certain types of crimes are committed", said Lauritsen, a researcher. The crime rates are not only correlated with temperature, crime can be correlated with various other factors such as socioeconomic statuses. However, for this project we are specifically focusing on temperature and crime.



Following the correlation, a linear regression model was executed in the data. We regressed crime rates onto average temperature and saw an interesting upward trend line. It seems that as the temperature gets warmer, the higher the crimes rates are. We also obtained a extremely small p-value and R-squared of 0.48. However, we must recall that simply having a distinct trend line and a statistically significant p-value, does not mean that one causes the other. But, crime is human behavior and practically it is very challenging to predict human behavior. Thus, humans are harder to predict than physical processes because humans can be easily distracted by many factors. Therefore, studying this crime data, we found out that average temperature has statistically significant impact on crime rates.

## Social Awareness Program

Social Memory:

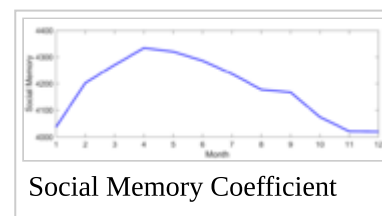
$$SM[k] = \sum_{i=k+1}^{k+12} c[i] \cdot e^{(i-k)/8}$$

where  $c[i]$  is the average crime incidents happening in the  $i^{th}$  month of recent years.  $k$  represents the particular month and  $SM[k]$  represents the effective social memory we can produce if we host awareness programs on that month.  $e^{(i-k)/8}$  models the decay of people's memory after participating the awareness program in the  $k^{th}$  month of the year. The figure on the right shows what the social memory coefficients are for each months. It shows us that the memory is at maximum on the month of April.

## Optimizing the Date

<1> If one and only one of month would be chosen, obviously we choose the  $k$  corresponding to the max value of  $SM[k]$ . That is April.

<2> School scenarios: For a school with regular spring/fall semester, we can only pick  $k$  among  $\{2,3,4,9,10,11\}$ . That are April and September.



<3> If we want to conduct awareness programs twice a year but with frequency no higher than once in every three months, maximizing the social memory is to maximizing the following objective function:

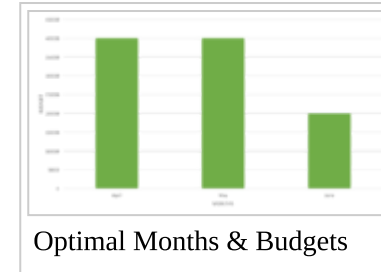
$$\max(SM[k_1] + SM[k_2]), s. t. |k_1 - k_2| \geq 3$$

That are April and July.

# Allocating Budgets

$x_k$  represents the budget allocated to each month.

$$\begin{aligned} &max(\sum_{k=1}^{12} x_k \cdot SM[k]) \\ &s.t. \sum_{k=1}^{12} x_k = 100000 \\ &0 \leq x_k \leq 40000, k = 1...12 \end{aligned}$$



Here is the link for the AMPL code for the linear program and the Project Poster: <https://github.com/kushmakarb/LP-code>

## Future Work

One of the goals of this research is to find the correlation between crime and temperature. However, in our current analysis, we used the average monthly temperature as the data for temperature which can be improved.

Future work to do:

- Classify each crime case into indoor-type or outdoor-type. For outdoor type, use the precise outdoor temperature when the incident happened.
- Divide the city into different regions and divide the crime into different sub-types. Then find how a particular type of crime would change along the change of the temperature in a particular region.
- Collect more information about how social awareness programs are organized so that we can design a more detailed plan for the budgeting of awareness programs.

## Bibliography

1. City of Denver Police Department/Data Analysis Unit Country of Denver Crime. 2018.

2. Lauritsen PhD, J.L. Seasonal Patterns in Criminal Victimization Trends. Bureau of Justice Statistics. 2014.

3. Jari Tiihoen, Pirjo Halonen. The Association of Ambient Temperature and Violent Crime. 2017.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Crime\\_%26\\_Temperature:\\_Scheduling\\_Awareness\\_Programs&oldid=1882](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Crime_%26_Temperature:_Scheduling_Awareness_Programs&oldid=1882)"

- This page was last modified on 3 December 2018, at 19:05.
- This page has been accessed 7,485 times.

# Crime response planning by linear programing

From CU Denver Optimization Student Wiki

**Crime response planning by linear programming** is a project giving the optimal police location knowing a set of crime positions. The objective of the project is to find the optimal solutions of transportation problem for several marginals. In order to do this, Discrete Wasserstein barycenters was used.

The Denver crime data for the years 2012 to 2017 is available in Open Data Catalog. The analysis in the project was focused on murder data but could be extended to other crimes.

The data was classified by months. The motivation to do this is that the number of police men in Denver for a year is fixed even when some months have more crime than others. We need the police to be able to react to the crime in all the months. Then, the question is where to locate the police to react faster to the crime? The idea is to minimize the distance between the crime location and the police location.

The main result of the project is a Python code that extracts relevant data from Denver Open Data Cataloge, computes discrete barycenters for crime data in AMPL, and visualizes results on Google maps.

## Contents

- 1 Data extraction and cleaning
- 2 Discrete Wasserstein Barycenters<sup>[2]</sup>
  - 2.1 Linear programming model
    - 2.1.1 Parameters
    - 2.1.2 Variables
  - 2.2 Experiments
  - 2.3 Optimization using group of months
- 3 Strongly Polynomial 2-Approximation<sup>[3]</sup>
  - 3.1 Optimization method
  - 3.2 Experiments
- 4 Conclusions
- 5 References

# Data extraction and cleaning

Denver Open Data Catalog<sup>[1]</sup> includes criminal offenses in the City and County of Denver for the previous five calendar years plus the current year to date. The data is based on the National Incident Based Reporting System (NIBRS) which includes all victims of person crimes and all crimes within an incident. The data is cleaned in the preprocessing stage of the Python program avoiding crime locations outside of Denver area.

## Discrete Wasserstein Barycenters<sup>[2]</sup>

The Discrete Wasserstein Barycenters model will be called exact model in future sections and it is defined in the next subsection.

### Linear programming model

It is required to solve the next linear programming model:

Objective function

$$\min \sum_{i=1}^N \lambda_i \sum_{j=1}^{|S_0|} \sum_{k=1}^{|P_i|} ||x_j - x_{ik}||^2 y_{ijk}$$

Subject to

$$\begin{aligned} \sum_{k=1}^{|P_i|} y_{ijk} &= z_j, \forall i = 1, \dots, N, \forall j = 1, \dots, |S_0|, \\ \sum_{j=1}^{|S_0|} y_{ijk} &= d_{ik}, \forall i = 1, \dots, N, \forall k = 1, \dots, |P_i|, \\ y_{ijk} &\geq 0, \forall i = 1, \dots, N, \forall j = 1, \dots, |S_0|, \forall k = 1, \dots, |P_i|, \\ z_j &\geq 0, \forall j = 1, \dots, |S_0|. \end{aligned}$$

Where the objective functions tries to minimize the Euclidian distance between the crime and possible police locations, the 2 first constrains balance the offer and demand of police in the possible police locations and the 2 last ones ensure all the weights to be positive.

In the two next subsections, there are explained all the parameters and variables of the model using an example of 3 month murder data of Denver from January to March of 2016.



## Parameters

The crime location  $k$  in the month  $i$  is represented by  $x_{ik}$ . The location can be represent by a latitude-longitude representation and it is possible to see in the figure on the right in a Google maps visualization. The 3 different marker colors: black, green and brown, represent respectively the different months January, February and March ( $N = 3$ ) and each month  $i$  has a particular number of crimes which are represented by  $|P_i|$  ( $|P_1| = 3$ ,  $|P_2| = 2$  and  $|P_3| = 3$ ).

$\lambda_i$  is giving a specific weight for each month  $i$ . The reason of this parameter is to set the different months by relevance. For instance, if it is known that in March Denver is more dangerous than in the other months, it is possible to define a bigger weight in that particular month. The only restriction defining  $\lambda_i$  is that:

$$\sum_{i=1}^N \lambda_i = 1$$

In all the experiments below,  $\lambda_i$  is defined to have the same weight for each month:

$$\lambda_i = \frac{|P_i|}{\sum_{i=1}^N |P_i|}$$

So, in the example:

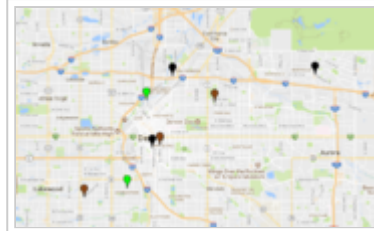
$$\lambda_1 = \frac{|P_1|}{\sum_{i=1}^N |P_i|} = \frac{3}{8} = 0.375$$

$$\lambda_2 = \frac{|P_2|}{\sum_{i=1}^N |P_i|} = \frac{2}{8} = 0.25$$

$$\lambda_3 = \frac{|P_3|}{\sum_{i=1}^N |P_i|} = \frac{3}{8} = 0.375$$

For each crime, it is necessary to know also a specific crime weight,  $d_{ik}$ , which represents the demand of each crime. If it was necessary to put more weight in a specific crime, it will be possible using this parameter. The only restriction of  $d_{ik}$  is:

$$\sum_{k=1}^{|P_i|} d_{ik} = 1, \forall i = 1, \dots, N$$



Example of a 3 months crimes data location ( $x_{ik}$ ) visualization of Denver from January to March of 2016.

Typesetting math: 100% below, it is used the same weight in all the crimes in the same month, so:

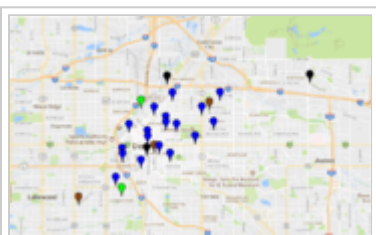
$$d_{ik} = \frac{1}{|P_i|}, \forall k = 1, \dots, |P_i|$$

So, in our example:

$$d_{1k} = \frac{1}{|P_1|} = \frac{1}{3}, \forall k = 1, \dots, 3$$

$$d_{2k} = \frac{1}{|P_2|} = \frac{1}{2}, \forall k = 1, \dots, 2$$

$$d_{3k} = \frac{1}{|P_3|} = \frac{1}{3}, \forall k = 1, \dots, 3$$



In blue, the possible locations of the police  $x_j$  using an example of 3 months crimes data  $x_{ik}$  of Denver from January to March of 2016.

The last parameter is the possible locations of the police, represented by  $x_j$ , which is computed using the crimes location data  $x_{ik}$ . It is defined as the mean of all the individual combinations of the crime locations of each month. So, the number of possible police locations  $|S_0|$  are:

$$|S_0| = \prod_{i=1}^N |P_i|$$

So, in the previous example:

$$|S_0| = \prod_{i=1}^3 |P_i| = |P_1| \cdot |P_2| \cdot |P_3| = 3 \cdot 2 \cdot 3 = 18$$

Therefore, in the left image, there are 18 blue markers representing the possible police locations.

## Variables

All the linear programming model is controlled by the transport variable  $(y_{ijk})$  which balances the weights between crimes and police.

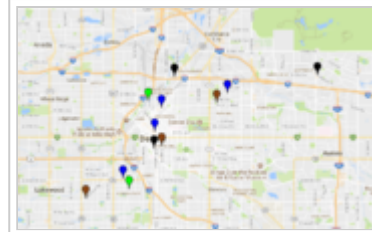
Finally, the police weights,  $z_j$ , gives the offer of the model. Each weight represents the relevance of each police location and there is also the same previous restriction:

$$\sum_{j=1}^{|S_0|} z_j = 1$$

At the end, there are taken all the police  $x_j$  satisfying:

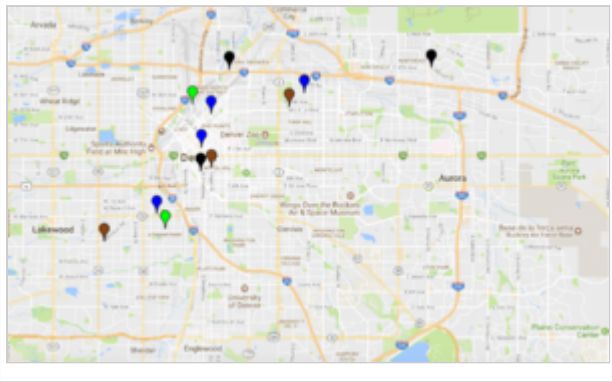
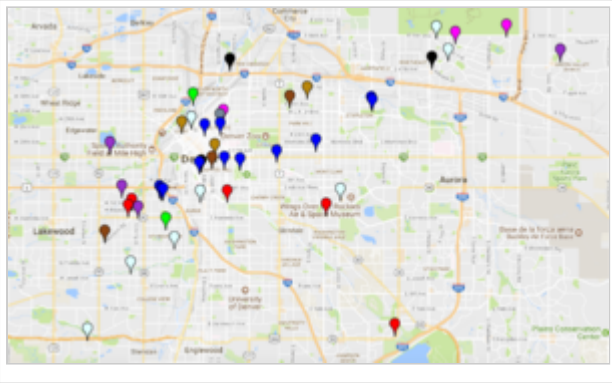
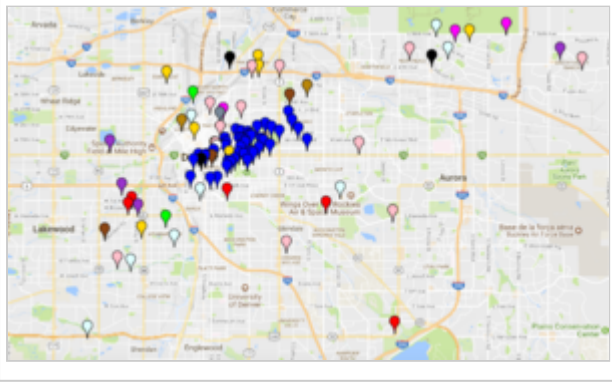
$$z_j > 0$$

So, in the example, there are 4 police locations with positive weight and the other 14 are removed because they have weight 0.

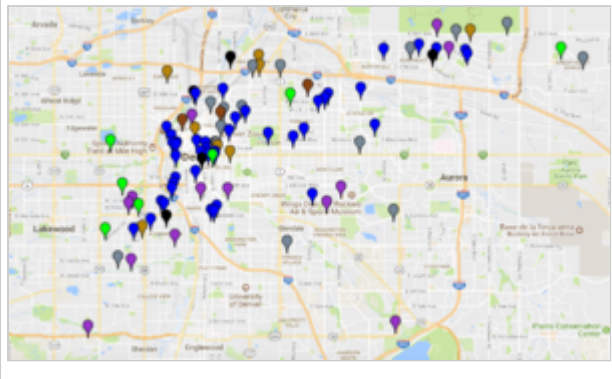
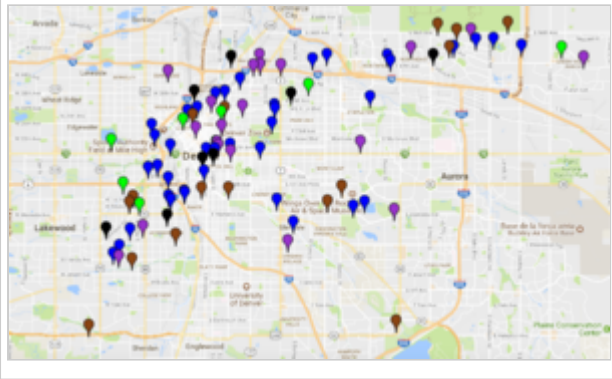


Final location of the police (in blue all  $x_j$  with  $z_j > 0$ ) of an example of 3 months data crimes of Denver from January to March of 2016.

# Experiments

Discrete Wasserstein Barycenters experiments		
Experiment name	Map result	Description
3 months experiment (previous example)		The execution time using the exact model for 3 months is 0.12 seconds. The value of the objective function is 0.00182.
9 months experiment		The execution time using the exact model for 9 months is 59 seconds. The value of the objective function is 0.00377.
12 months experiment		The execution time using the exact model for 12 months is 66 minutes. The computer ran out of memory before AMPL can give an optimal result.

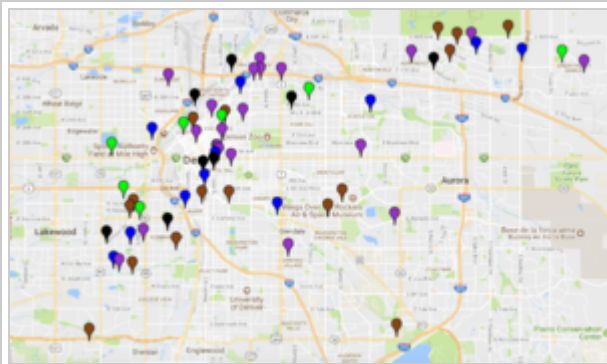
# Optimization using group of months

Discrete Wasserstein Barycenters experiments using groups of months		
Experiment name	Map result	Description
12 months experiment using 2-months groups (6 groups)		Now It is possible to run the exact model for a year. The execution time using the exact model for 12 months in groups of 2 months is 22 minutes. The value of the objective function is 0.00177.
12 months experiment using 3-months groups (4 groups)		The execution time using the exact model for 12 months in groups of 3 months is 70 seconds. The value of the objective function is 0.00107. There are a lot of police markers. This happens because we are increasing the number of murders per group (period of time, 3 months per group).

It is defined  $\epsilon$ , the tolerance to choose the police. Then, it is chosen  $x_j$  such that:

$$z_j > \epsilon$$

So, going back to the 12 months experiment using 3-months groups, the result using  $\epsilon = 0.03$  is:



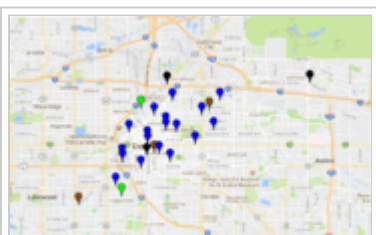
12 months experiment using 3-months groups (4 groups) and  $\epsilon = 0.03$

In the previous image, it is possible to see a significant reduction of the police markers.

## Strongly Polynomial 2-Approximation<sup>[3]</sup>

The Strongly Polynomial 2-Approximation algorithm trades a small error for a significant reduction in computational effort. The Strongly Polynomial 2-Approximation will be called 2-approximation in future sections and it is defined in the next subsection.

### Optimization method

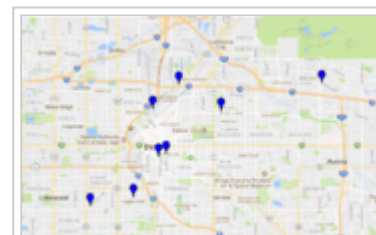


In blue, the possible locations of the police  $x_j$  using the exact model.

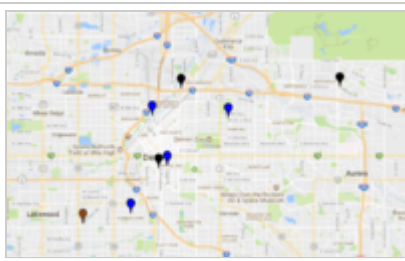
The main difference between the exact model and the 2-approximation method is computing the  $x_j$  parameter (possible police location). In the exact model, the possible police location is the mean of all the individual combinations of the crime locations of each month. In the 2-approximation, the possible police location is the exact location of the crimes. The figure on the left is showing the possible police location for the exact model and the 2-approximation technique is used on the right image.

Therefore, the reduction of possible police locations is significantly reduced and this gives a computational memory solution that implies also a significant reduction of the execution time.

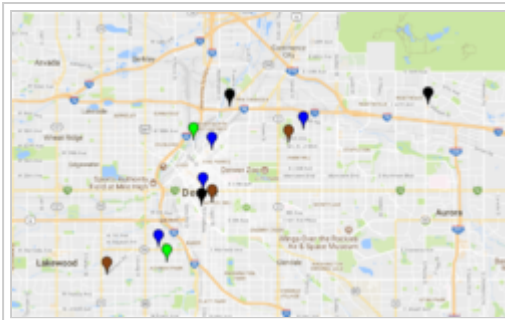
When the execution finishes, it gives the police locations over crime data (figure left below). Therefore, it is necessary a postprocessing stage in the Python code where the final location of the police will be the mean of the crime locations where the police have positive transport. In other words, the mean of the crime locations where a specific police marker have to react. The final result doing this is showing in the image below in the middle which can be compare to the below right figure (result for the same experiment using the exact model). It is possible to observe that the results are very similar, so the 2-approximation gives a very good approximation.



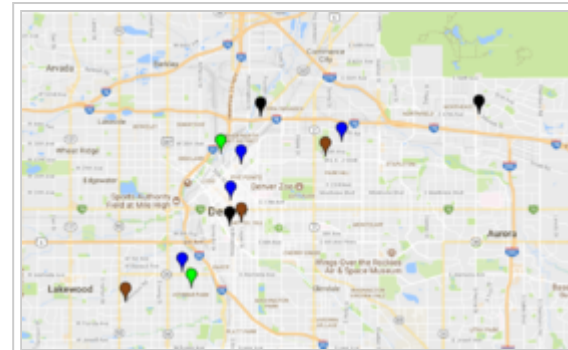
In blue, the possible locations of the police  $x_j$  using the 2-approximation method. The crime data is below the blue markers.



Result of the police location (in blue) using Strongly Polynomial 2-Approximation and postprocessing the results for the previous example.



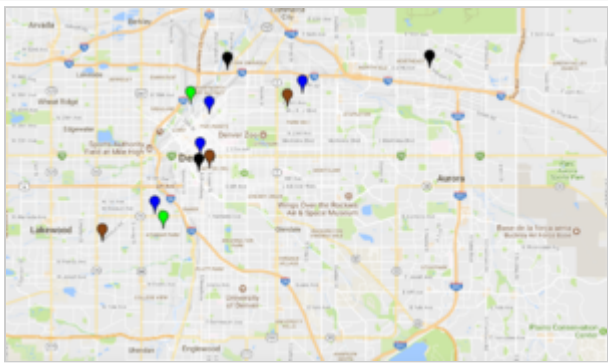
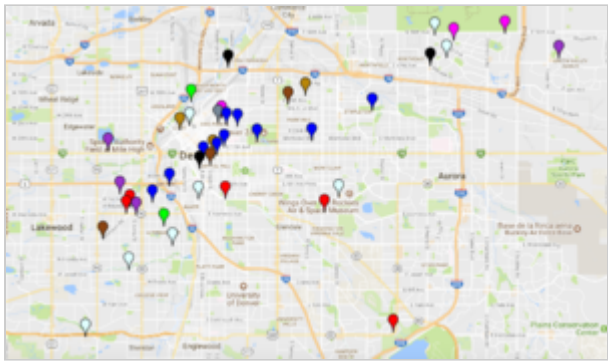
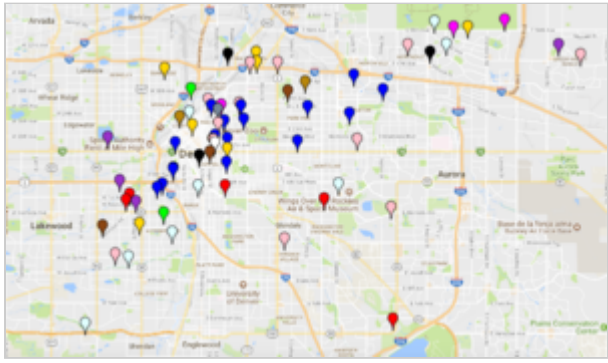
Result of the police location (in blue) using Strongly Polynomial 2-Approximation for the previous example.



Final location of the police (in blue) using the exact model in the previous experiment.



# Experiments

Strongly Polynomial 2-Approximation experiments		
Experiment name	Map result	Description
3 months experiment		The execution time using 2-aproximation for 3 months is 0.11 seconds. The value of the objective function is 0.00182.
9 months experiment		The execution time using 2-aproximation for 9 months is 0.17 seconds. The value of the objective function is 0.00379.
12 months experiment		The execution time using 2-aproximation for 12 months is 0.29 seconds. The value of the objective function is 0.00334. There are fewer police markers than the exact model with groups for the same number of murders.







# Cycle Cancelling Algorithm

From CU Denver Optimization Student Wiki

## Contents

- 1 Abstract
- 2 Introduction
- 3 Feasibility and Algorithm Setup
- 4 Solving and Optimality Condition
- 5 Complexity
- 6 Code Outline
- 7 Files and Presentation
- 8 Contributors
- 9 Sources

## Abstract

This was created as a final project for MATH 7825: Topics in Optimization. It is a basic overview of the Cycle Cancelling Algorithm with explanations of the pseudocode and an example run in the presentation. It also includes explanations of the optimality condition and the complexity.

## Introduction

To understand the cycle-cancelling algorithm we need to understand what a minimum cost flow problem is as this is the type of problem this algorithm is intended for. A min-cost flow problem finds the minimum cost necessary to send the maximum flow through a network.

Notation and Assumptions (heavily adapted from (Ahuja et al., 1993))

- Let:**
- $G = (N, A)$**  be a directed network with  $N$  nodes and  $A$  arcs.
- $c_{ij} \geq 0$**  represent the cost of arc  $(i, j) \in A$
- $u_{ij}$**  represent the capacity of arc  $(i, j) \in A$
- $s(i)$**  represent the supply or demand of node  $i$  depending on whether  $s(i) > 0$  or  $s(i) < 0$ ,  $\forall i \in N$
- $C$**  denote the largest magnitude of any arc cost.
- $U$**  denote the largest magnitude of any supply/demand or finite arc capacity.

So, the minimum cost problem is:

Minimize  $z(x) = \sum_{(i,j) \in A} c_{ij}x_{ij}$

subject to:

$$\sum_{\{f:(i,j) \in A\}} x_{ij} - \sum_{\{f:(j,i) \in A\}} x_{ji} = s(i) \quad \forall i \in N,$$
$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in A.$$

We will also assume: The network is directed. (This is necessary for us to have feasible flow.) The data is all integral. (Data being defined as arc capacities/costs and node supplies/demands)

Note: If we have integral values for all our data, our solution will also be integral which is important to note as many of these problems will not allow for a non-integral solution. The validity of this claim is discussed in Theorem 9.10 (Integrality Property) (Ahuja et al., 1993)

## Feasibility and Algorithm Setup

First we need to verify that the problem in question is feasible. To do this we first need to check if the sum of all the supplies (positive values) and demands (negative values) equals zero.

$$\sum_{\{i \in \mathcal{N}\}} s(i) = 0$$

Next, we need to verify that a feasible flow exists. To do this we need to convert the problem into a max flow problem by adding an 's' node and a 't' node with arcs going to each supply and demand node respectively. We will set the capacities of these arcs to the supply/demand values of those nodes.

Now, we will verify we can find a valid max flow using a max flow algorithm like Edmonds-Karp (an implementation of the Ford-Fulkerson augmenting path algorithm). If we find a valid max flow, we will use that as our initial feasible solution to start the Cycle Cancelling Algorithm.

## Solving and Optimality Condition

**While** the residual network  $G(x)$  contains a negative cost cycle:

- (1) Build a residual graph based on the initial feasible solution.
- (2) Use an algorithm (discussed in detail during complexity section) to detect a negative cost cycle  $W$  in the residual graph of the initial feasible solution. If no negative cost cycle is found end. (The current solution is the optimal solution.)
- (3) If a negative cost cycle  $W$  is found, set  $\delta = \min\{r_{ij} : (i, j) \in W\}$ ; (sets the amount of units to augment the path to the lowest remaining capacity along that path)
- (4) Augment  $\delta$  units of flow along all arcs in  $W$  to "cancel" the negative cycle and update residual graph  $G(x)$ ; (The updated  $G(x)$  becomes the residual graph used for step 1 of the next iteration.)

**Optimality:**

"Theorem 9.1 (Negative Cycle Optimality Conditions). A feasible solution  $x^*$  is an optimal solution of the minimum cost flow problem if and only if it satisfies the negative cycle optimality conditions: namely, the residual network  $G(x^*)$  contains no negative cost (directed) cycle." (Ahuja et al., 1993)

What this means is if there is no negative cost cycle in the residual network, an optimal solution has been reached. This is because the existence of a negative cost cycle implies that an alternate path with a lower cost still exists and/or has not been fully utilized yet. Therefore if we have achieved maximum flow and there are no negative cycles, there is no better path to choose and we have found the optimal solution.  $\square$

## Complexity

The complexity of the Cycle Cancelling algorithm will depend on the choices of subsystem algorithms. There are several components we will look at to find the order of the overall complexity but for a specific number it will end up being based on the algorithm choices made for the max-flow algorithm and the negative cycle location algorithm.

Initially we add arcs in place of supply/demand values. In the worst case, where all nodes have non-zero supply/demand values, we add  $n$  arcs.  **$O(n)$**

Note, this also increases our arc count,  $m$ , to  $m+n$ , and our node count,  $n$ , to  $n+2$ .

Then we run a max-flow algorithm to solve for the initial feasible solution. Here are a few options with varied complexity to use as a starting point:

**$O(nm^2)$**  Edmonds-Karp ([https://en.wikipedia.org/wiki/Edmonds-Karp\\_algorithm](https://en.wikipedia.org/wiki/Edmonds-Karp_algorithm)) (Better for low density graphs)

**$O(mF)$**  Ford-Fulkerson ([https://en.wikipedia.org/wiki/Ford-Fulkerson\\_algorithm](https://en.wikipedia.org/wiki/Ford-Fulkerson_algorithm)) where  $F$  is the value of the max flow (Better if we know there is a small max flow)

**$O(n^3)$**  preflow-push ([https://en.wikipedia.org/wiki/Push-relabel\\_maximum\\_flow\\_algorithm](https://en.wikipedia.org/wiki/Push-relabel_maximum_flow_algorithm)) (Using the FIFO selection rule, best for extremely dense graphs)

Next, we will be running an algorithm to find a negative cost cycle. This algorithm will run one time for each iteration, up to the maximum iteration count of  $\mathbf{mCU}$  due to the integral data. Here are two options included in (Ahuja et al., 1993) which have different running times:

**$\Theta(n^3)$**  Floyd-Warshall ([https://en.wikipedia.org/wiki/Floyd-Warshall\\_algorithm](https://en.wikipedia.org/wiki/Floyd-Warshall_algorithm))

**$O(mn)$**  FIFO label-correcting algorithm for shortest path (Chapter 5.4 (Ahuja et al., 1993))

As an example, if we choose Edmonds-Karp, the FIFO label-correcting algorithm for shortest path, and let  $\mathbf{mCU}$  represent the maximum number of iterations needed to clear all negative cycles, we would be in the order of  $O(n + nm^2 + mCUnm) \subseteq O(CUnm^2)$ .

We can see that while the choice of max flow algorithm can affect the complexity, it is likely that the iteration count will force the negative cost cycle finding algorithm term to dominate the complexity function. For larger problems with specific structure, it can be useful to research a variety of options for these algorithms and find ones that suit your particular problem well.

## Code Outline

As there are multiple algorithms used as a subroutine for the cycle cancelling algorithm, it lends itself well to object oriented programming languages. We can create a "Cycle-Cancelling" class with methods for each step. This will allow the user to select which algorithms they prefer to use for the max-flow and negative cycle detection subroutines. Having a default algorithm for each will allow less knowledgeable users, as well as those who don't need that level of granularity to solve problems using Cycle Cancelling.

The first method would convert the problem to a max-flow problem by taking the supply/demand values and making them into arc capacities leading to dedicated supply and demand nodes. We can write methods for the initial max-flow calculation and for each algorithm you may want to use to solve it. We will also write a method to detect negative cycles by calling a specified algorithm and again another for each algorithm you want to have available for this part. This structure makes it easy to adapt and add to. The class can also have a solve() method which calls these methods and runs until an optimal solution is found.

The other advantage of coding it this way is you could build an algorithm selector which calculates the optimal algorithm to use based on the size/nature of the problem data using the complexity for each algorithm. There is a pseudocode example on the github linked below showing how this might look if done in Python.

## Files and Presentation

Presentation slides and code ran can be found at the following link. [1] (<https://github.com/pgmath/Min-Mean-Cycle-Cancelling-Algorithm>)

## Contributors

Paul Guidas

## Sources

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications*. Prentice-Hall.

Korte, B., & Vygen, J. (2018). *Combinatorial optimization: Theory and algorithms*. Springer.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Cycle\\_Cancelling\\_Algorithm&oldid=4829](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Cycle_Cancelling_Algorithm&oldid=4829)"

- 
- This page was last modified on 2 May 2024, at 18:15.
  - This page has been accessed 51 times.

# Dalton Burke

From CU Denver Optimization Student Wiki

I'm an undergraduate double majoring in Mathematics and Computer Science, currently set to graduate with both in May of 2019. I hope to continue into graduate school for both, my preferences are certainly more theoretically inclined Discrete Mathematics and Theoretical Computer Science, though I don't mind splashing around in applications from time to time. In my free time (if I'm ever so lucky) I enjoy playing video games competitively, listening to music, and hanging out with friends.

Contributions: Equal Flow Problem, Identifying Gentrification

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Dalton\\_Burke&oldid=1612](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Dalton_Burke&oldid=1612)"

Category: Contributors

- 
- This page was last modified on 8 November 2018, at 16:14.
  - This page has been accessed 1,784 times.

# Data Visualization Using QGIS

From CU Denver Optimization Student Wiki

## Contents

- 1 Introduction
- 2 Installation
- 3 Importing Data
  - 3.1 Shapefiles
  - 3.2 Text Files
- 4 Connecting Points
- 5 Example Project
- 6 Resources
- 7 Motivation

## Introduction

Geographic information system (GIS) software can be useful when visualizing spatial data. QGIS (<https://www.qgis.org/en/site/>) is a freely available, open source software program that you can use on your personal computer (other tools are available for use at the Auraria Library (<https://library.auraria.edu/>)).

## Installation

Installation is not difficult. There tend to be two versions available at any time, an older potentially more stable version and a newer version with more features (but perhaps more issues). You will need to install Python (<https://www.python.org/>), and the QGIS installation instructions will guide you on the version.

## Importing Data

### Shapefiles

Spatial data, such as that found on Denver (<https://www.denvergov.org/opendata>) and Colorado (<https://data.colorado.gov/>) open data websites, is shared in a variety of formats, including shapefiles, .shp. When downloading a shapefile, you will notice that it comes packaged with many other files. These must be kept together.

## Text Files

## Connecting Points

Point Connector (<https://plugins.qgis.org/plugins/PointConnector/>) plugin for QGIS.

## Example Project

QGIS was used to produce the visualizations for Denver Hate Crime Mapping: Visualizing Fluctuations through Linear Programming.

## Resources

QGIS (<https://www.qgis.org/en/site/>).

Point Connector (<https://plugins.qgis.org/plugins/PointConnector/>) plugin for QGIS.

Auraria Library (<https://library.auraria.edu/>)).

## Motivation

This page is an attempt by Kathleen\_Gatliffe to document her progress using QGIS during the Fall 2018 session of MATH 5593: Linear Programming. It is far from complete. Other users of QGIS are encouraged to expand on what little is written here.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Data\\_Visualization\\_Using\\_QGIS&oldid=1866](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Data_Visualization_Using_QGIS&oldid=1866)"

- 
- This page was last modified on 2 December 2018, at 12:37.
  - This page has been accessed 2,523 times.



# Degree Sequence

From CU Denver Optimization Student Wiki

## Contributors:

- Hanbyul Lee

## Contents

- 1 Abstract
- 2 Definitions
  - 2.1 Definition 1: Degree Sequence
  - 2.2 Definition 2: 2-switch
- 3 Theorem 1: Havel–Hakimi Theorem
  - 3.1 Detailed Proof
    - 3.1.1 Necessity: If  $d$  is graphic, then  $d'$  is graphic
    - 3.1.2 Sufficiency: If  $d'$  is graphic, then  $d$  is graphic
- 4 Theorem 2: Graph Transformations via 2-switch
  - 4.1 Necessity Proof
  - 4.2 Sufficiency Proof
- 5 Python Implementation of Havel–Hakimi
- 6 Concluding Remarks
- 7 GitHub Repository
- 8 7. References

## Abstract

This project focuses on the fundamental concept of the **degree sequence** in graph theory, which plays a critical role in several fields, including network generation, data preprocessing, and algorithmic design. A degree sequence specifies the number of edges incident to each vertex, and verifying whether such a sequence can be realized by a simple graph (i.e., deciding if it is "graphic") is a classical yet highly applicable problem.

To address this question, we first introduce the notion of degree sequences in a formal manner and discuss the **2-switch** operation, which allows us to transform one graph into another without altering the individual vertex degrees. We then present the well-known **Havel–Hakimi Theorem**, an essential procedure that determines whether a given degree sequence is indeed graphic. Beyond theoretical interest, we also provide a practical Python implementation of the **Havel–Hakimi Algorithm** illustrating how this algorithm can be employed in real-world scenarios, such as quickly generating synthetic networks or validating degree constraints in complex datasets.

Overall, this project highlights how the concept of degree sequences connects fundamental graph-theoretic principles to practical applications in network analysis and computational modeling. Through detailed examples and code demonstrations, we show how one can efficiently leverage the Havel–Hakimi algorithm and 2-switch operations for both theoretical exploration and hands-on problem-solving.

# Definitions

## Definition 1: Degree Sequence

A graph  $G$  on  $n$  vertices has a **degree sequence** consisting of the degrees of each vertex, usually listed in nonincreasing order. A nonnegative integer list is called **graphic** if there exists a simple graph whose degree sequence matches that list.

- Example: The list (3,3,1,1) is not graphic for a 4-vertex graph, because

if two vertices have degree 3, the other two must each be connected to both of those, implying they would have degree at least 2, which contradicts (3,3,1,1).

(For the original statement of this definition in Combinatorial Mathematics 1st Edition, see [Ref Book] Section 5.2.5.)

When we write  $G - v$ , we mean the subgraph obtained by removing vertex  $v$  and all edges incident to  $v$ .

## Definition 2: 2-switch

A **2-switch** is the operation of removing edges  $xy$  and  $zw$  from a graph  $G$  and instead adding the edges  $xz$  and  $yw$ . This preserves each vertex's degree but changes the adjacency relationships.

(See [Ref Book] Section 5.2.7 for the original source.)

# Theorem 1: Havel–Hakimi Theorem

**Theorem 1 (Havel [1955], Hakimi [1962]).** A nonnegative integer list  $d$  of size  $n > 1$  is graphic if and only if the list  $d'$ , formed by removing the largest element  $\Delta$  of  $d$  and subtracting 1 from the next  $\Delta$  largest elements, is also graphic.

## Detailed Proof

We now provide a more comprehensive argument, split into two parts: **Necessity** ( $d \rightarrow d'$ ) and **Sufficiency** ( $d' \rightarrow d$ ).

**Necessity: If  $d$  is graphic, then  $d'$  is graphic**

- Let  $d$  be a graphic degree sequence, and  $G$  be a simple graph realizing  $d$ .
- Let  $\Delta = \max(d)$ . Choose a vertex  $w$  in  $G$  with  $\deg_G(w) = \Delta$ .

## 2. Forming $d'$

- By definition,  $d'$  is formed by removing  $\Delta$  from  $d$  and subtracting 1 from the next  $\Delta$  largest entries.
- In graph terms, this corresponds to removing  $w$  and reducing the degrees of its  $\Delta$  neighbors by 1.

## 3. Selecting $S$

- Let  $S$  be the set of  $\Delta$  vertices other than  $w$  with the highest degrees in  $G$ .
- If  $N_G(w) = S$ , then removing  $w$  directly produces the degree sequence  $d'$ .

## 4. 2-switch (Case $N_G(w) \neq S$ )

- If  $w$  is not adjacent to some  $x \in S$ , or is adjacent to some  $\alpha \notin S$ , we can use a 2-switch to “swap” these adjacencies without changing any vertex’s degree.
- Repeatedly applying 2-switches increases  $|N_G(w) \cap S|$  step by step, until  $w$  is adjacent exactly to the vertices in  $S$ .

## 5. Final Graph $G^*$

- After finitely many switches, we get  $G^*$  with  $N_{G^*}(w) = S$ .
- Removing  $w$  in  $G^*$  yields a graph whose degree sequence is exactly  $d'$ , proving  $d'$  is graphic.

## Sufficiency: If $d'$ is graphic, then $d$ is graphic

### 1. Assumption

- Suppose  $d'$  is already known to be graphic.
- Thus, there is a simple graph  $G'$  on  $n-1$  vertices, say  $v_1, v_2, \dots, v_{n-1}$ , such that  $\deg_{G'}(v_i)$  matches the entries of  $d'$ .

### 2. Identifying $\Delta$ -reduced Vertices

- Recall  $d'$  was obtained by removing the largest element  $\Delta$  from  $d$  and subtracting 1 from the next  $\Delta$  largest elements.
- In reverse, to recover  $d$ , we must add a new vertex  $w$  with degree  $\Delta$  and connect it exactly to those  $\Delta$  vertices that were each reduced by 1.

### 3. Constructing $G$

- In  $G'$ , pick the  $\Delta$  vertices that “lost 1” when forming  $d'$  from  $d$ .
- Add a new vertex  $w$  to  $G'$ . Then, create edges from  $w$  to precisely these  $\Delta$  vertices.

### 4. Resulting Degrees

- The new vertex  $w$  has degree  $\Delta$ .
- Each of the chosen  $\Delta$  vertices in  $G'$  gains 1 in degree, returning to their original degree values as specified by  $d$ .
- All other vertices remain at their same degrees from  $G'$ .

### 5. Conclusion

- The resulting graph  $G$  on  $n$  vertices realizes the sequence  $d$ .
- Hence, if  $d'$  is graphic,  $d$  must also be graphic.

## Theorem 2: Graph Transformations via 2-switch

**Theorem 2. (Fulkerson–Hoffman–McAndrew [1965], Berge [1970]).** Let  $G$  and  $H$  be graphs on the same vertex set  $V$ . Then  $G$  can be transformed into  $H$  by a finite sequence of 2-switches if and only if  $d_G(v) = d_H(v)$  for all  $v \in V$ .

### Necessity Proof

Suppose there is a finite sequence of 2-switches that transforms  $G$  into  $H$ . Concretely, assume we have  $G = G_0 \xrightarrow{\text{2-switch}} G_1 \xrightarrow{\text{2-switch}} \dots \xrightarrow{\text{2-switch}} G_k = H$ .

Each  $G_i$  is obtained from  $G_{i-1}$  by performing exactly one 2-switch. We claim that for every vertex  $v \in V$ , the degree  $\deg(v)$  remains invariant throughout these transformations. Hence  $d_G(v) = d_H(v)$  for all  $v \in V$ .

To see why the degree remains invariant, recall the definition of a 2-switch:

- A 2-switch involves choosing four distinct vertices  $\{x, y, z, w\}$  such that

either the edges  $(x,y)$  and  $(z,w)$  exist while  $(x,z)$  and  $(y,w)$  do not,  
or vice versa.

- Applying a 2-switch replaces the two existing edges with the two missing edges (or the other way around).

In either case, each of the four involved vertices loses exactly one edge and gains exactly one edge, keeping its degree the same. Any vertex not involved in that specific 2-switch remains unaffected, so its degree does not change. Consequently, every vertex  $v$  has the same degree in  $G_{i-1}$  and  $G_i$  for all  $i = 1, \dots, k$ .

By iteration,  $\deg_{G_k}(v) = \deg_{G_{k-1}}(v) = \dots = \deg_{G_0}(v)$  for all  $v \in V$ . Since  $G_k = H$ , we conclude that  $\deg_H(v) = \deg_G(v)$  for every  $v \in V$ . Thus  $d_G$  and  $d_H$  coincide.

This completes the necessity part of the proof.  $\square$

## Sufficiency Proof

We now show that if  $d_G(v) = d_H(v)$  for all  $v \in V$ , then  $G$  can be transformed into  $H$  by 2-switches.

We proceed by induction on  $|V|$ :

- Base Case ( $|V| \leq 3$ ):\*\*

When there are at most 3 vertices, any given degree sequence is realized by at most one graph (with labels). Thus if  $d_G = d_H$ , then  $G$  and  $H$  must be the same graph already—no 2-switch is required.

- Inductive Step ( $|V| = n \geq 4$ ):\*\*

Assume that for any graph on fewer than  $n$  vertices, sharing the same degree sequence implies transformability by 2-switches. We prove the statement for  $n$ -vertex graphs.

1. Let  $w$  be a vertex of maximum degree  $\Delta$  in  $G$ . Since  $d_G(v) = d_H(v)$  for all  $v$ , vertex  $w$  also has degree  $\Delta$  in  $H$ .
2. Choose an arbitrary set  $S \subseteq V \setminus \{w\}$  of size  $\Delta$ . By repeatedly applying 2-switches, we can ensure  $w$ 's neighbor set in  $G$  becomes exactly  $S$ . We do this by removing any “unwanted” neighbor  $\alpha \notin S$  and replacing it with some “missing” neighbor  $\beta \in S$  using a suitable 2-switch. Denote the resulting graph by  $G^*$ .
3. Perform the same procedure in  $H$  to obtain  $H^*$ , where  $w$  has neighbor set  $S$ .
4. Now remove  $w$  from both  $G^*$  and  $H^*$ . Let

$G' = G^* - \{w\}$  and  $H' = H^* - \{w\}$ .  
Since  $w$  had the same set  $S$  of neighbors in both  $G^*$  and  $H^*$ , every other vertex  $v \neq w$  has the same degree in  $G'$  and  $H'$ .

5. By the induction hypothesis (on  $n - 1$  vertices),  $G'$  can be transformed into  $H'$  by 2-switches. Those 2-switches do not involve  $w$  and thus can be carried out within  $G^*$ ,

Hence  $G^*$  transforms into  $H^*$ .

6. Finally, invert the 2-switches used to form  $H^*$  from  $H$ , thereby converting  $H^*$  back into  $H$  (2-switches are reversible).

Combining these transformations, we obtain a finite sequence of 2-switches converting  $G$  into  $H$ . This completes the proof by induction.

## Python Implementation of Havel–Hakimi

Below is a concise Python example illustrating the Havel–Hakimi algorithm. It checks whether a degree list is graphic by repeatedly removing the largest degree, subtracting 1 from the appropriate number of subsequent entries, and ensuring no negative degree arises: Note: The code can be downloaded from this link (<https://github.com/otter275/Degree-Sequence-MATH-6404/blob/main/havel-hakimi.py>).

```
def havel_hakimi(deg_seq):  
    """  
    Determines if a given degree sequence is graphic using the Havel-Hakimi algorithm.  
  
    Parameters:  
    deg_seq (list of int): A list of nonnegative integers representing the degrees.  
  
    Returns:  
    bool: True if the sequence is graphic, False otherwise.  
    """  
    # Step 1: Sort the sequence in descending order for convenience.  
    seq = sorted(deg_seq, reverse=True)  
  
    while True:  
        # Step 2: Remove all zeros (any vertex with degree 0 doesn't affect the process).  
        seq = [d for d in seq if d > 0]  
  
        # If the sequence is empty after removing zeros, it means all degrees are satisfied -> graphic  
        if not seq:  
            return True  
  
        # Sort again in descending order to ensure we always pick the largest remaining degree first  
        seq.sort(reverse=True)  
  
        # Step 3: Take the first (largest) element, call it D  
        D = seq[0]  
        # Remove that element from the sequence  
        seq = seq[1:]  
  
        # If D is larger than the length of the remaining list, we can't subtract from enough vertices  
        # -> not graphic  
        if D > len(seq):  
            return False  
  
        # Step 4: Subtract 1 from the next D elements  
        for i in range(D):  
            seq[i] -= 1  
  
        # If element goes below 0, it means there's an impossible requirement -> not graphic
```

```
        if seq[i] < 0:
            return False
    seq.sort(reverse=True) # keep order

def example_usage():
    """
    Demonstrates how to use the havel_hakimi function.
    """
    # Example degree sequence
    example_list = [3,3,2,2,2,1]

    # Print "Graphic?" if the sequence is graphic, otherwise "Not Graphic"
    print("Graphic?" if havel_hakimi(example_list) else "Not Graphic")
```

This implementation can be extended to construct an actual graph structure (e.g., adjacency list) once the sequence is confirmed to be graphic, by tracking which vertices are decremented in each step.

## Concluding Remarks

We have demonstrated the Havel–Hakimi theorem in full:

- Necessity: If  $d$  is graphic, so is  $d'$ .
- Sufficiency: If  $d'$  is graphic, then  $d$  is also graphic.

Hence  $d$  is graphic if and only if  $d'$  is graphic. This proof (combined with the constructive approach) underpins the Havel–Hakimi algorithm’s effectiveness in testing and realizing degree sequences.

## GitHub Repository

Degree Sequence Project Repository (<https://github.com/otter275/Degree-Sequence-MATH-6404>)

## 7. References

- Combinatorial Mathematics. 1st ed., Cambridge University Press, 2020. (Sections 5.2.5–5.2.8 for original statements)
- “Havel–Hakimi algorithm.” *Wikipedia, the Free Encyclopedia*.

Accessed 2025-04-20 ([https://en.wikipedia.org/wiki/Havel%E2%80%93Hakimi\\_algorithm](https://en.wikipedia.org/wiki/Havel%E2%80%93Hakimi_algorithm)).

- Weisstein, Eric W. “Degree Sequence.” *MathWorld — A Wolfram Web Resource*.

Accessed 2025-04-20 (<https://mathworld.wolfram.com/DegreeSequence.html>).

■ NetworkX Developers. “networkx.generators.degree\_seq.havel\_hakimi\_graph.” *NetworkX Documentation (Stable)*.

Accessed 2025-04-20 ([https://networkx.org/documentation/stable/reference/generated/networkx.generators.degree\\_seq.havel\\_hakimi\\_graph.html](https://networkx.org/documentation/stable/reference/generated/networkx.generators.degree_seq.havel_hakimi_graph.html)).

■ NetworkX Developers. “Graph Generators — NetworkX Documentation (Stable).”

Accessed 2025-04-20 (<https://networkx.org/documentation/stable/reference/generators.html>).

■ Erdős, P. et al. “On Graphic Degree Sequences and Their Applications.” *\*arXiv preprint\* arXiv:2105.12120*, 2021.

Accessed 2025-04-20 (<https://arxiv.org/abs/2105.12120>).

■ (ko) Gazelle & CS Blog. “그래프의 차수열 Havel–Hakimi 알고리즘. (Graph Degree Sequences and the Havel–Hakimi Algorithm)”

Accessed 2025-04-20 (<https://gazelle-and-cs.tistory.com/102>).

■ (ko) Baekjoon Online Algorithm. 문제 23578 “그래프의 차수열 결정. (Determining Graphic Sequences)”

Accessed 2025-04-20 (<https://www.acmicpc.net/problem/23578>).

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Degree\_Sequence&oldid=4986"

Category: Contributors

- This page was last modified on 3 May 2025, at 15:08.
- This page has been accessed 128 times.



# Denver Fire Response Distances

From CU Denver Optimization Student Wiki

This page is for the Fall 2020 Linear Programming project by Sandra Robles and Hope Haygood. In this project, we seek to find the optimal assignment of fire stations to fire (arson) crimes committed over the past 5 years in the Denver metropolitan area.

## Contents

- 1 Abstract
- 2 Overview & The Data
- 3 Methods
- 4 Results
- 5 Policy
- 6 Code and Presentations
- 7 References

## Abstract

Being able to respond to arson calls effectively, efficiently, and quickly is a topic of pivotal interest for fire departments everywhere. The time it takes to drive from the fire station to the scene of the fire needs to be as small as possible in order to ensure the damage is contained. Through methods of linear programming, we show that the choice of fire stations to respond to a set of arson incidents can be optimized in view of minimal overall driving distances between fire stations and incidents, while balancing responses among stations. Our results deviate from general response rules, which leads to policy suggestions for improvements in a coordinated response among fire departments.

We use arson incident data provided by the County of Denver for the Denver Metro area to provide a proof of concept for our methods. Our approach includes visualization techniques to show possible patterns of fires and to recommend locations for new fire stations.

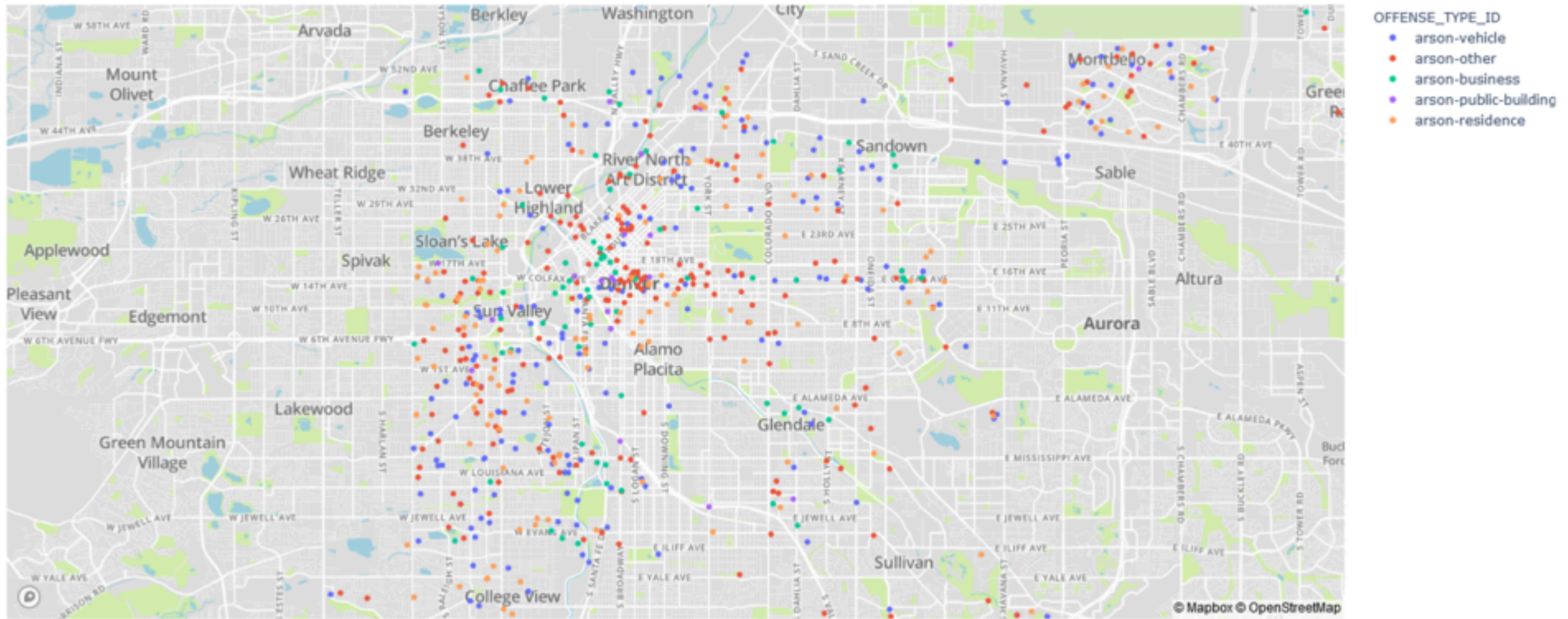


## Overview & The Data

The County of Denver provides its citizens with a data repository (<https://www.denvergov.org/opendata>) detailing the minutia of life from the perspective of a local government. One of these datasets contains all crime (<https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-crime>) that has transpired within country boundaries for the last 5 years. This dataset contains arson crimes, which are the focus of our paper.

Arson is defined (<https://www.merriam-webster.com/dictionary/arson>) as "the willful or malicious burning of property (such as a building) especially with criminal or fraudulent intent". It is also an event that is very time sensitive for first responders, as even seconds can make the difference between partial and full destruction from the fire. In the last 5 years there have been 684 cases of arson within the county.

These cases of arson can be separated into the following subtypes, which specify what type of item or location was enveloped in fire: business, public building, residence, vehicle, other.



Location of each arson, by type of arson.

# Methods

Our project uses an unbalanced assignment problem. This type of linear program is a close sibling to the classic transportation problem. A transportation problem is a type of LP where we have a set of  $m$  supply points with non-zero supplies  $s_i$  and some sort of transportation cost  $c_{ij}$  between these. An assignment problem will just be this set up, but where the supply and demand are both one. Formally, we define it as:

$$\min(\sum_{i \in n} \sum_{j \in m} c_{ij} x_{ij})$$

Such that:

$$\sum_{i \in n} x_{ij} = 1 \text{ for all } i \in n$$

$$x_{ij} \geq 0$$

Said in simpler terms, an assignment problem seeks to assign 'workers' to 'tasks' in such a way that the overall cost (measured in whatever units are relevant to the problem) is minimized.

A hurdle with our data stems from the fact that we have significantly more arson cases than there are available fire stations. This results in our assignment problem being *unbalanced*. This is a problem because an assumption of the Assignment problem is that the  $i, j$  are equal in size -- meaning, the matrix is square. A naive recommendation ([https://en.wikipedia.org/wiki/Assignment\\_problem#Unbalanced\\_assignment](https://en.wikipedia.org/wiki/Assignment_problem#Unbalanced_assignment)) is to take an unbalanced problem and turn it into a balanced problem by adding as many dummy rows in either  $i$  or  $j$  as is needed until equality in size is achieved.

We chose to use Python's SciPy's implementation of the assignment problem, `linear_sum_assignment` ([https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear\\_sum\\_assignment.html](https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html)), which can implement a generalized assignment problem without the need to constrain the problem to a square matrix.

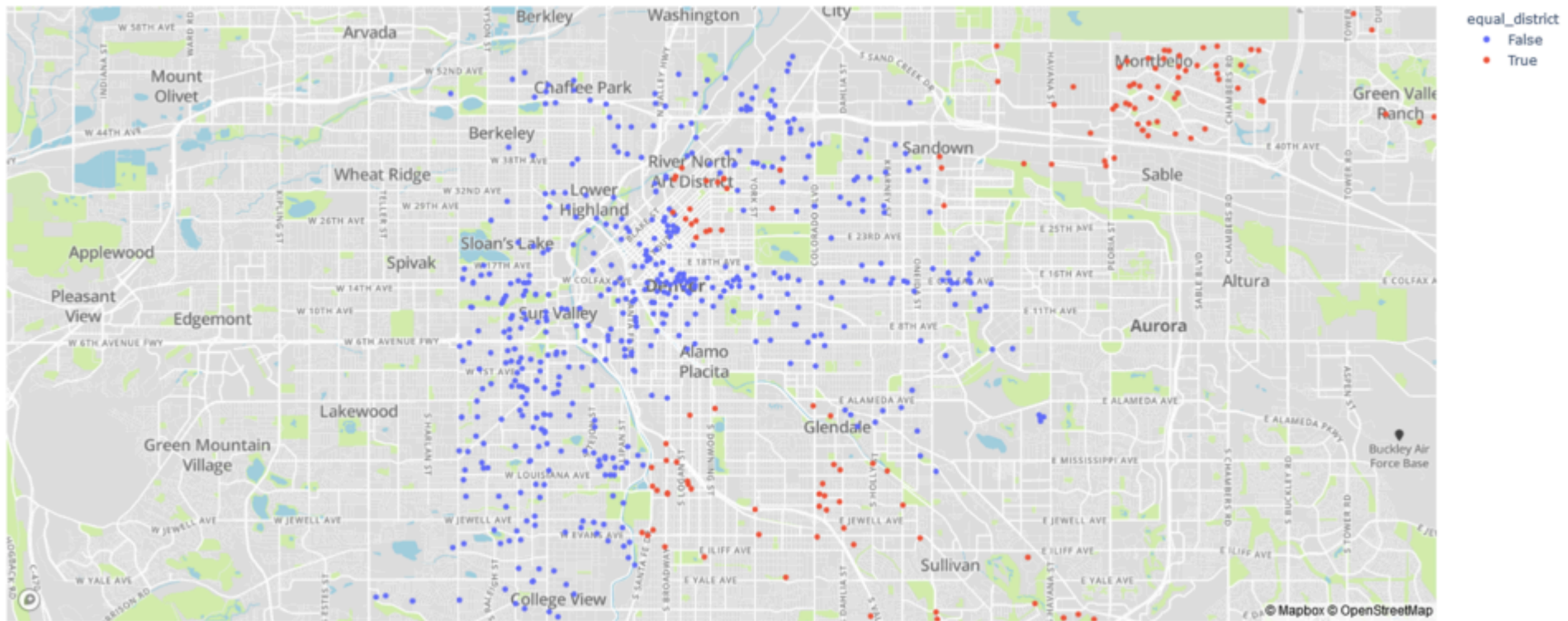
## Results

We saw that our LP recommended a fire station respond outside of its fire district on 78% of instances.

	arson_index	date	firestation	OFFENSE_TYPE_ID	INCIDENT_ADDRESS	arson_district	PRECINCT_ID	NEIGHBORHOOD_ID	STATION_NUM	DISTRICT	different_district
0	0	#####	FS-23	arson-vehicle	1700 BLK W ATLANTIC PL	4	422	college-view-south-platte	FS-23	7	FALSE
1	429	#####	FS-6	arson-other	1600 BLK 15TH ST	6	612	union-station	FS-6	2	FALSE
2	1	#####	FS-26	arson-vehicle	3332 N PONTIAC ST	2	221	northeast-park-hill	FS-26	5	FALSE
3	2	#####	FS-14	arson-other	8300 E COLFAX AVE	2	223	east-colfax	FS-14	4	FALSE
4	3	#####	FS-1	arson-vehicle	1353 N BANNOCK ST	6	611	civic-center	FS-1	2	FALSE
5	4	6/2/2018	FS-1	arson-business	1365 N OSAGE ST	1	123	lincoln-park	FS-1	2	FALSE
6	5	#####	FS-23	arson-vehicle	1500 BLK S KING ST	4	421	mar-lee	FS-23	7	FALSE
7	6	#####	FS-12	arson-vehicle	3125 N FEDERAL BLVD	1	113	west-highland	FS-12	6	FALSE

### Sample Output from our program

- Column 'arson\_district' is the district in which the fire occurred
- Column 'firestation' is which station in which the Linear Program recommends to respond to the fire
- Column 'District' is the district of the fire station that the Linear Program says should respond
- Column 'different\_district' triggers FALSE if the responding fire station is outside of the district in which the fire occurred.



Plot showing which fires should be attended by stations outside of their fire district. 'True' indicates the responding fire station belongs to the same fire district as the location of the fire.

## Policy

Our results show that a majority of fires could be attended by fire stations closest to them, even if outside of their fire district. As such, there is strong evidence to suggest heightened communication between the districts would greatly improve the response to fires. We hope that by minimizing the response times to these fires through open district communication, these crimes of arson can be less damaging.

Further, this is likely indication that building more fire stations would be beneficial.

## Code and Presentations

[https://github.com/srobles09/Minimizing\\_Arson](https://github.com/srobles09/Minimizing_Arson) (Github contains the code and presentations)

# References

Crime data, last accessed Nov 20 2020 (<https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-crime>)

Fire station information, last accessed Nov 20, 2020 (<https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-fire-stations>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Denver\\_Fire\\_Response\\_Distances&oldid=3050](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Denver_Fire_Response_Distances&oldid=3050)"

- 
- This page was last modified on 30 November 2020, at 20:47.
  - This page has been accessed 1,241 times.



# Denver Government Coordinate Systems

From CU Denver Optimization Student Wiki

## Contents

- 1 Introduction
- 2 Colorado
- 3 Working with Alternate Coordinate Systems
- 4 Resources
- 5 Motivation

## Introduction

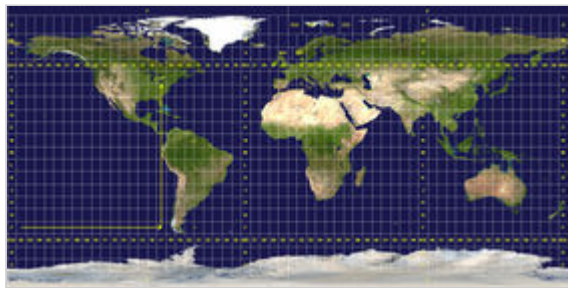
When working with geographic data on a local level (such as the data sets available on the Denver (<https://www.denvergov.org/opendata>) and Colorado (<https://data.colorado.gov/>) open data websites, you may notice that some data sets use x and y coordinates, not latitude and longitude. These coordinates are coded using the State Plane Coordinate (SPC) system. The SPC system is a set of 124 coordinate systems spanning the entire United States. The boundaries between zones are made on the county level. These coordinate systems are highly accurate and generally more convenient when working in two dimensions. Thus they are preferred over latitude and longitude in many civil engineering and surveying applications, which is why they are included in city and county level data.

Another important two dimensional projection to be aware of is the Universal Transverse Mercator (UTM) coordinate system. UTM is a global projection, dividing the earth into sixty-six degree longitude strips running north to south. Unlike longitude, these strips do not vary uniformly in width so care must be taken the closer a projection gets to the poles.

## Colorado

There are three state plane coordinate systems in Colorado: north, south and central. There are two UTM zones that cross Colorado, but, for convenience, all references in Colorado are made to only one, zone 13.

Abbreviation	Code	Designation	Origin Longitude	Origin Latitude
CO N	501	North	105°30'W	39°20'N
CO C	502	Central	105°30'W	37°50'N
CO S	503	South	105°30'W	36°40'N

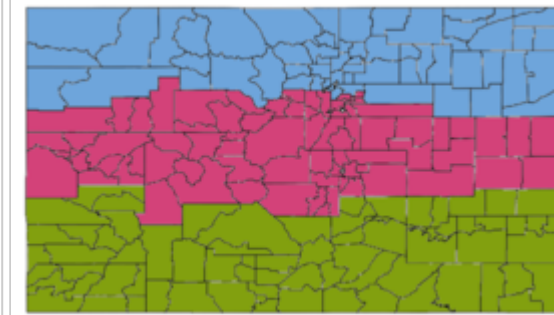


The Universal Transverse Mercator zones

Note that Denver is in Central, as are most of the surrounding counties, but counties just north of Denver (Adams, Boulder, Broomfield) are in North. If using data from these two areas be extremely careful to ensure that you are using the correct projections.

## Working with Alternate Coordinate Systems

Check carefully what coordinate system your data is in. If all the data is using the same coordinate system, you can use the coordinates without much trouble. Unfortunately, there is no easy metric to converting SPC coordinates to (for example) latitude and longitude. Individual points can be entered into online conversion tools (this one (<http://www.earthpoint.us/StatePlane.aspx>), from Earthpoint is very good). In addition, GIS Software (such as QGIS (<https://www.qgis.org/en/site/>)) is able to make the conversions for you, provided you specify the correct projection (more on that ([here](#))).



Colorado state plane coordinate systems (blue is north, fuschia is central, green is south)

The Auraria Library maintains a GIS expert on staff, Diane Fritz (<https://library.auraria.edu/about/staff-directory/diane-fritz>), who is available to meet by appointment ([https://ucdenver.co1.qualtrics.com/jfe/form/SV\\_cYmzZtsxNgoFHx3](https://ucdenver.co1.qualtrics.com/jfe/form/SV_cYmzZtsxNgoFHx3)) to assist you. The Auraria Library also has computers dedicated to working with GIS information that you can use.

## Resources

Auraria Library (<https://library.auraria.edu/>).

Covert State Plane to Latitude and Longitude (<http://www.earthpoint.us/StatePlane.aspx>).

State Plane Coordinate System. ([https://en.wikipedia.org/wiki/State\\_Plane\\_Coordinate\\_System](https://en.wikipedia.org/wiki/State_Plane_Coordinate_System)) Wikipedia.

Universal Transverse Mercator coordinate system ([https://en.wikipedia.org/wiki/Universal\\_Transverse\\_Mercator\\_coordinate\\_system](https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system)) Wikipedia.

Information page (<https://www.codot.gov/business/designsupport/cadd/tips-tricks/miscellaneous/misc-tip-survey-coordinate-system.pdf>) from CDOT concerning the State Plane Coordinate System in Colorado.

Information from NOAA ([https://geodesy.noaa.gov/library/pdfs/NOAA\\_SP\\_NOS\\_NGS\\_0013\\_v01\\_2018-03-06.pdf](https://geodesy.noaa.gov/library/pdfs/NOAA_SP_NOS_NGS_0013_v01_2018-03-06.pdf)) including maps of all US SPCS zones.

SPC Information (<http://stateplane.ret3.net/#UT>) for every county in the United States.

## Motivation

This page is an attempt by Kathleen\_Gatliffe to answer questions that came up during the Fall 2018 session of MATH 5593: Linear Programming. Edits are encouraged! Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Denver\\_Government\\_Coordinate\\_Systems&oldid=1871](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Denver_Government_Coordinate_Systems&oldid=1871)"



- This page has been accessed 17,654 times.

# Denver Hate Crime Mapping: Visualizing Fluctuations through Linear Programming

From CU Denver Optimization Student Wiki

## Contents

- 1 Abstract
- 2 Introduction to Bias Motivated Crime
- 3 Methods
- 4 Analysis and Results
  - 4.1 Case Study: 2011-2014
  - 4.2 Case Study: 2014-2017
- 5 Discussion
- 6 References
- 7 Motivation
- 8 Resources

## Abstract

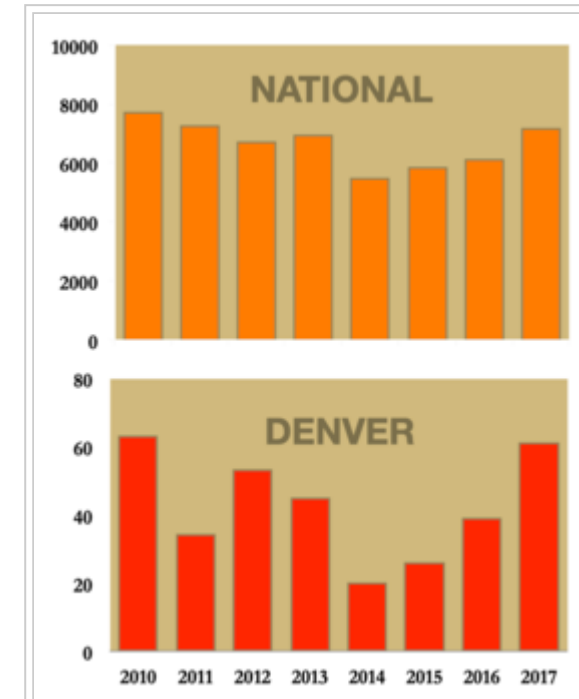
Bias-motivated crime--criminal acts galvanized by prejudice--showed an uptick in the year 2017 after holding relatively steady in the United States for over a decade. This surge in domestic extremism has awakened concerns about public safety. These crimes are of special interest to the Federal Bureau of Investigation and thus many police departments, including Denver, collect information on the bias-motivated crimes committed within their communities and share it with the national database. This data is available from 2010 to 2018 and is updated frequently. The number of bias-motivated crimes reported in Denver also increased in 2017, similar to the national trend, while the crimes reported to date for 2018 suggest that this year will be equally high. In this project, linear optimization techniques were applied to the data released to the public by the Denver Police Department. This research detected patterns of interest, some matching national trends and others in opposition.

## Introduction to Bias Motivated Crime

Bias motivated crime, commonly called hate crime, are acts committed against a person or persons in an attempt to victimize an entire group of people.

In Colorado, protected categories include: disability, ethnicity, gender identity, race, religion, and sexual orientation. The Federal Bureau of Investigation also considers hate crimes by gender, but these are tracked in Denver.

Typesetting math: 100%

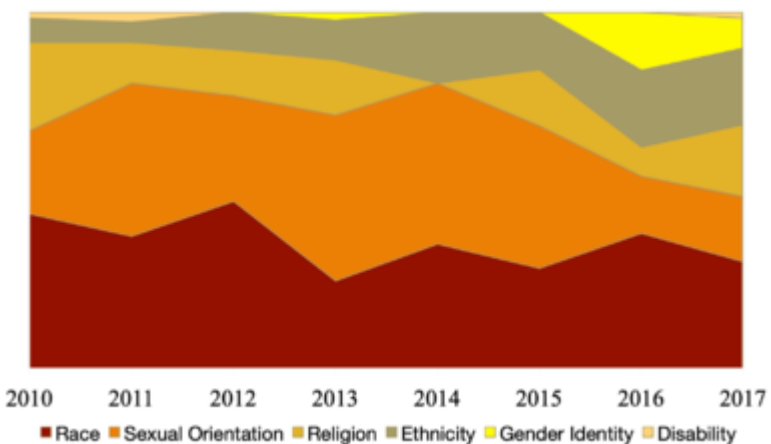


Number of hate crimes reported nationally and in Denver, by year, 2010-2017.

National statistics are gathered monthly by the FBI as part of their Uniform Crime Reporting (UCR) program and summary data is released in a yearly report. Several Colorado agencies contribute, including the Denver Police Department.

Reported hate crimes in Denver tend to follow national trends, but due to the relatively small number of reported crimes, fluctuations can appear magnified. In addition, hate crime is tends to be underreported meaning that increases in reported crime may reflect growing awareness of the issue on the part of the public and not an escalation in bias motivated crime.

Bias motivated crime was high at the beginning of the decade, but declined over the next few years, only to surge again in 2016-2018. Since 2010, the percentage of crimes motivated by race and religion bias in Denver tends to be much lower than the national average. Denver tends to report a higher percentage of crimes related to sexual orientation than the national average. Crimes related to ethnicity (primarily crimes against the Latino community) have increased in Denver, exceeding the national average, especially in recent years. Denver has also seen a dramatic increase in crimes related to gender identity in the past two years, far beyond the national average.



Percentage of various types of hate crime in Denver, by year, 2010-2017.

## Methods

This project used a linear sum assignment problem to connect crimes committed in one year to another. While assignment problems tend to be used to connect people and jobs, or resources and plants, they can also be used to show spatial shifts.

The mathematics is relatively simple. Choose two years. Let  $i$  denote the crimes in the earlier year, and  $j$  be the crimes in the following year. Define the Euclidean distance between locations to be  $c(ij)$  and let  $x(ij)$  be a Boolean variable that is true is there is a connection between  $i$  and  $j$ , and false otherwise . We choose our objective function to be the minimization of the summation of all distances.

$$\min(\sum_{i \in S} \sum_{j \in D} c_{ij} x_{ij})$$

We have two classes of constraints. Without loss of generality, we set the assignments emerging from each origin equal to one,

$$\sum_{i \in S} x_{ij} = 1 \text{ for all } i \in S$$

Since there tend to be a different number of origin  $n_S$  and destination  $n_D$  crimes, the destination constraint is that the assignments to each destination are equal to the ratio between the number of origins and destinations,

$$\sum_{j \in D} x_{ij} = \frac{n_D}{n_S} \text{ for all } j \in D$$

## Analysis and Results

### Case Study: 2011-2014

In this project two assignments were analyzed. Since there were major changes between 2010 and 2018, three years were chosen, 2011, 2014, and 2017, three years apart.

Between 2011 and 2014, reported hate crimes decreased in Denver.

Reported hate crimes were scattered throughout the city in 2011, often occurring near major roadways.

By 2014, we see hate crime condensing into downtown with isolated incidents in less travelled areas. Hate crimes in Five Points and North Capitol Hill moved from major roads onto side streets.

### Case Study: 2014-2017

From 2014 to 2018 there was a dramatic increase in hate crimes in the city.

By 2017 the reported numbers were equal to where they were in 2010. New hot spots developed in 2017, including Hampden and the neighborhoods near Ruby Hill Denver University also was a target with numerous reports of anti-Jewish activities. Numerous hate crimes remain in Downtown, but now center on the Union Station transit center and Arapahoe.

The increase in 2017 seems to be primarily driven by increased reports of criminal mischief, and may not indicate increased aggression and danger.

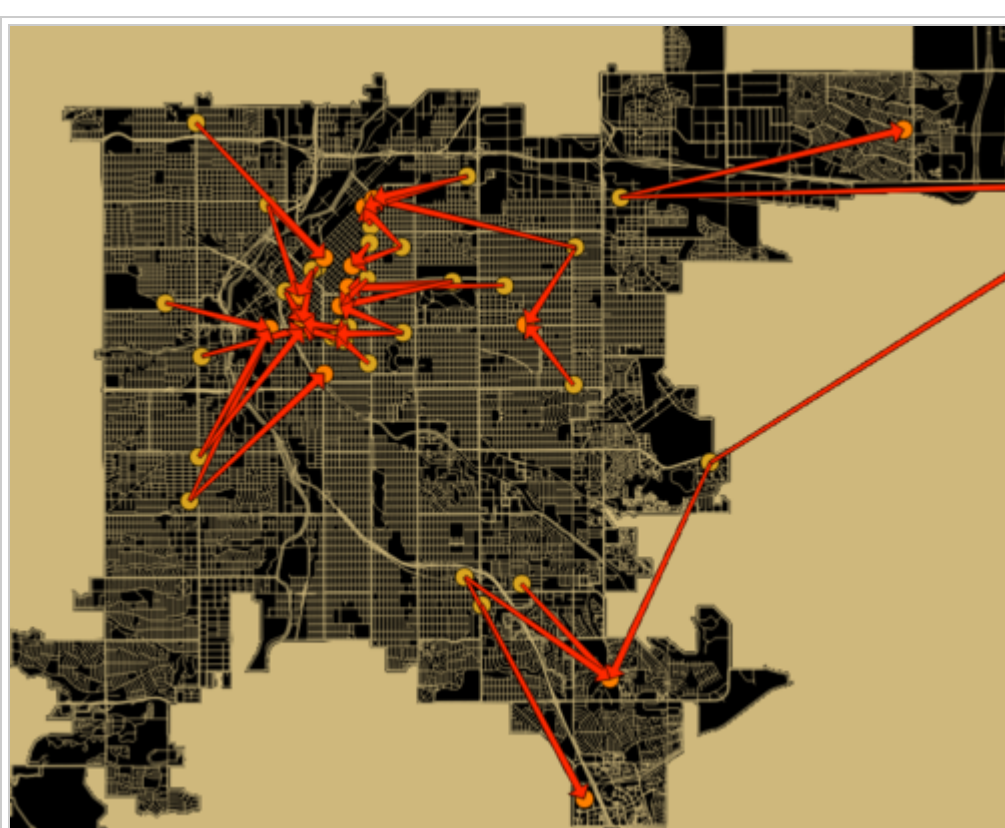
## Discussion

The use of an assignment problem creates a new tool for visualizing changes in criminal activity. Coupled with the associated statistics, the linear sum assignment model can provide new information on crime location.

## References

Burkard R.E., Çela E. (1999) Linear Assignment Problems and Extensions. In: Du DZ., Pardalos P.M. (eds) *Handbook of Combinatorial Optimization*. Springer, Boston, MA

Typesetting math: 100% ning People in Practice ([https://ampl.com/MEETINGS/TALKS/2011\\_01\\_Chiang\\_Mai\\_Plenary.pdf](https://ampl.com/MEETINGS/TALKS/2011_01_Chiang_Mai_Plenary.pdf)). Chiang Mai University International Conference.



Bias motivated crime in Denver for the years 2011 and 2014, linked by assignment. Yellow dots depict reported crimes 2011 and orange dots depict crimes in 2014.

Hauslohnner, A. 11 May 2018. Hate Crimes Jump for Fourth Straight Year in Largest U.S. Cities, Study Shows. ([https://www.washingtonpost.com/news/post-nation/wp/2018/05/11/hate-crime-rates-are-still-on-the-rise/?noredirect=on&utm\\_term=.a597d2eaa2f8](https://www.washingtonpost.com/news/post-nation/wp/2018/05/11/hate-crime-rates-are-still-on-the-rise/?hpid=hp_nation-post-nation&utm_term=.a597d2eaa2f8)) Washington Post.

Luenberger, D.G. and Ye, Y. 1973. *Linear and Nonlinear Programming*. Springer.

Hate Crime Statistics Act. ([https://en.wikipedia.org/wiki/Hate\\_Crime\\_Statistics\\_Act](https://en.wikipedia.org/wiki/Hate_Crime_Statistics_Act)) Wikipedia.

## Motivation

This is Kathleen Gatliffe's Fall 2018 project for MATH 5593: Linear Programming taught by Steffen Borgwardt (<http://math.ucdenver.edu/~sborgwardt/>). This project uses the Hate Crimes (<https://www.denvergov.org/opendata/dataset/hate-crimes>) database from the Denver Open Data Catalog (<https://www.denvergov.org/opendata/>).

This project was performed in AMPL (<https://ampl.com/>), R (<https://www.r-project.org/>), and QGIS (<https://www.qgis.org/en/site/>). This project will be presented at the Auraria Library's Data to Policy (<https://library.auraria.edu/d2pproject>) event on the 30th of November, 2018.

## Resources

The code (<https://github.com/Kgatliffe/DenverBiasMotivatedCrimes>) and other files produced for this project.

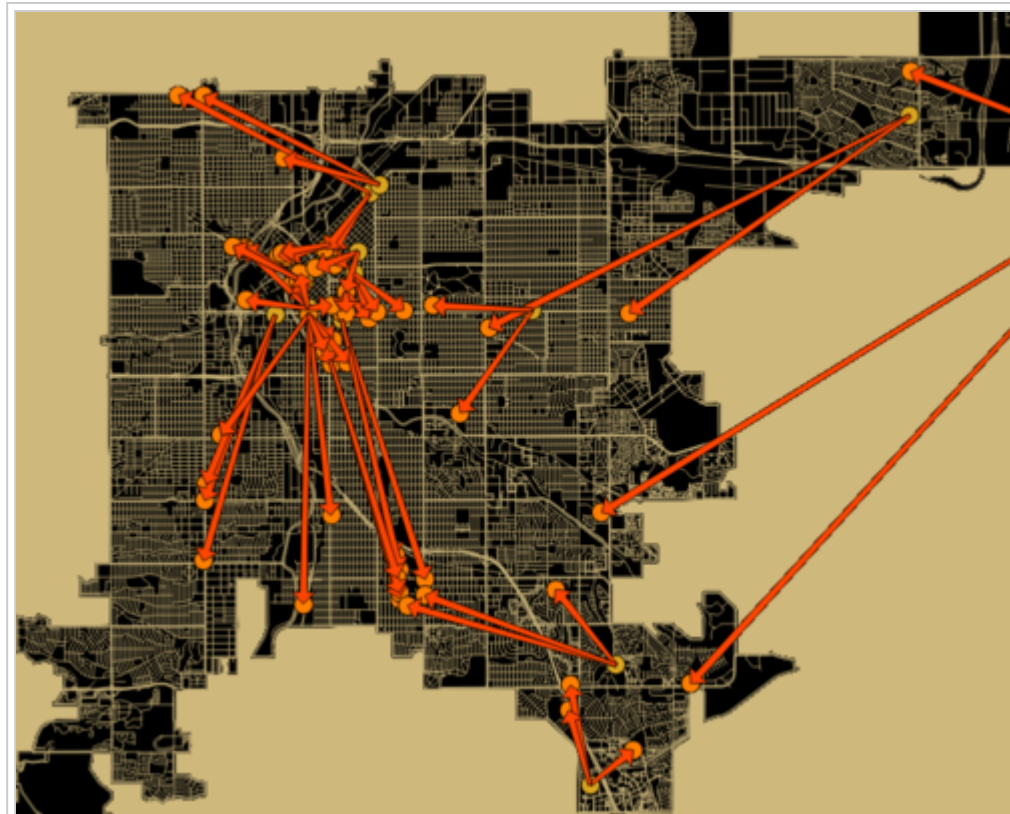
Project data (<https://www.denvergov.org/opendata/dataset/hate-crimes>), last collected 10 October, 2018.

National yearly statistics summaries (<https://www.fbi.gov/investigate/civil-rights/hate-crimes#Hate-Crime%20Statistics>) from the Federal Bureau of Investigation.

Additional information on methods can be found at Data Visualization Using QGIS and Denver Government Coordinate Systems

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Denver_Hate_Crime_Mapping:_Visualizing_Fluctuations_through_Linear_Programming&oldid=1834)

[title=Denver\\_Hate\\_Crime\\_Mapping:\\_Visualizing\\_Fluctuations\\_through\\_Linear\\_Programming&oldid=1834](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Denver_Hate_Crime_Mapping:_Visualizing_Fluctuations_through_Linear_Programming&oldid=1834)"



Bias motivated crime in Denver for the years 2014 and 2017, linked by assignment. Yellow dots depict reported crimes in 2014 and orange dots depict crimes in 2017.

- This page was last modified on 29 November 2018, at 23:06.
- This page has been accessed 6,414 times.

# Derivation of Blood Flow as a Network Flow

From CU Denver Optimization Student Wiki

The circulatory system, also known as the cardiovascular system, is an organ system that permits the circulation of blood to transport oxygen, carbon dioxide, nutrients (such as electrolytes and amino acids), blood cells and hormones to and from the cells in the body in order to provide nourishment and maintain homeostasis, stabilize temperature and pH, and fight diseases. The circulatory system also includes the lymphatic system which helps rid the body of toxins, cellular debris and other waste products. Signs of poor circulation can include dizziness, migraines, varicose veins, numbness, cold hands and feet, and pain in your feet and legs. Untreated poor circulation can lead to high blood pressure, stroke, reduced recovery from strenuous exercise and physically demanding work, a weakened immune system, kidney damage and can also be associated with diabetes, metabolic syndrome and other diseases.

Numerical simulation combining Hemodynamics (the study and transmission of blood flow) with the application of modelling the function of the heart as a network flow might just prove to be an invaluable tool that will help us better understand the function of the circulatory system given the specific scientific criteria. We might be able to better assess poor circulation and/or in detecting a stroke or some other kind of cardiac dysfunction and any other associated diseases before it occurs, and help improve in treating certain health conditions and injuries that might now be less surgery invasive while maximizing optimal blood flow to enhance the healing and recovery process.

The goal of this project is to determine if we can derive any clinical relevance in the application of modelling the function of the heart and blood flow as a network. Given that there is clinical significance in our described application, this could one day result in better prevention and treatment of strokes, cardiac dysfunction and any other associated diseases, a better protocol in the training and recovery of athletes, and in enhancing the effects in Sports, Massage and Physical Therapy by optimizing blood flow in the treatment of the specific injured body part.

We will first introduce some basic and important concepts of physiology pertaining to the function of the heart and blood flow and will then translate these physiological mechanisms to a network flow.

## Contents

- 1 Basic anatomy and physiology of the heart
- 2 Variables That Can Affect Blood Flow and Blood Pressure
- 3 Poiseuille's equation and Compliance
- 4 The heart as a network flow
  - 4.1 Basic formulations
  - 4.2 Problem derivation
- 5 Simplified model
  - 5.1 Blood vessel as network
- 6 Conclusion and Future Work
- 7 References

## Basic anatomy and physiology of the heart

The human heart is a hollow muscular organ that pumps blood filled with oxygen and nutrients (via the arteries) to the body, after dropping off the oxygen and nutrients to the body (via the capillaries), the now deoxygenated blood returns back to the heart (via the veins). The human heart has four chambers: two upper chambers (the atria) and two lower ones (the ventricles). According to the National Institutes of Health, the right atrium and right ventricle together make up the "right heart," and the left atrium and left ventricle make up the "left heart." As the heart pumps blood through the body, it is divided into the circulatory and pulmonary systems. The circulatory system consists of the heart pumping the oxygenated blood from the left ventricle to the body and the now deoxygenated blood returns to the right atrium of the heart. The pulmonary system consists of the heart pumping the deoxygenated blood from the right ventricle to the lungs in order to pick up the oxygen and the now oxygenated blood then returns from the lungs to the left atrium of the heart.

Blood is made up of plasma, red blood cells, white blood cells and platelets. In addition to blood, the circulatory system moves lymph, which is a clear fluid that helps rid the body of unwanted material.

Typesetting math: 100% is also of vital importance pertaining to the function of the heart and it is referred to as the pressure in large arteries of the systemic circulation. This is further expressed in terms of the



systolic pressure (contraction of the heart given a single heart beat) over diastolic pressure (relaxation of the heart that is between two heart beats) and is measured in millimeters of mercury (mmHg), above the surrounding atmospheric pressure (considered to be zero for convenience). Normal blood pressure of the heart is considered to be 120 (Systole)/80 (Diastole) of mercury (mmHg).

In the average human, about 2,000 gallons (7,572 liters) of blood travel daily through about 60,000 miles (96,560 kilometers) of blood vessels<sup>[1]</sup>. An average human's heart pumps about 4.7 to 5.6 liters of blood per minute.

## Variables That Can Affect Blood Flow and Blood Pressure

### Cardiac Output:

Cardiac Output is the amount of blood the heart pumps through the circulatory system in one minute. In other words,  $\text{CARDIAC OUTPUT} = HR \text{ (Heart rate)} \times SV \text{ (Stroke Volume)}$ . Stroke Volume is defined as the amount of blood pumped by the left ventricle of the heart in one contraction (heartbeat). While at rest (for example sitting at home reading a book), a normal healthy male about 25 years of age that weighs about 70 kg with a resting heart rate of 70 bpm (beats per minute) and has a stroke volume of 70 ml per heartbeat has a cardiac output where his heart pumps about 4.9 L of blood in one minute. A world class endurance athlete during a high intensity bike ride however, can have a cardiac output of 35 L in one minute.

### Compliance:

Compliance is the ability of any compartment to expand to accommodate increased content. A metal pipe, for example, is not compliant, whereas a lung is as it has the ability to expand. The greater the compliance of an artery, the more effectively it is able to expand to accommodate surges in blood flow without increased resistance or blood pressure. Veins are more compliant than arteries and can expand to hold more blood. When vascular disease causes stiffening of arteries, compliance is reduced and resistance to blood flow is increased. The result is more turbulence, higher pressure within the vessel, and reduced blood flow which also causes the heart to work harder.

### Volume of the Blood:

The relationship between blood volume, blood pressure, and blood flow is considered to be very intuitive. Just as water may slightly trickle along a creek bed in a dry season, but then it can rush quickly and under great pressure after a heavy rain, the same can be said if blood volume decreases, then its pressure and flow will decrease and if blood volume increases, then its pressure and flow will increase.

### Viscosity of the Blood:

Viscosity is the measure of resistance of a fluid to flow. A fluid that is highly viscous has a high resistance (like having more friction) and flows slower than a low-viscosity fluid. In other words, the viscosity of blood is directly proportional to resistance and inversely proportional to flow; therefore, any condition that causes viscosity to increase will also increase resistance and decrease flow. For example, imagine sipping water, and then a milkshake, through the same size straw. You will experience more resistance and therefore less flow from the milkshake. Conversely, any condition that causes viscosity to decrease (such as when the milkshake melts) will decrease resistance and increase flow.

### Blood Vessel Length and Diameter:

The length of a vessel is directly proportional to its resistance: the longer the vessel, the greater the resistance and the lower the flow. As with blood volume, this makes intuitive sense, since the increased surface area of the vessel will impede the flow of blood. Likewise, if the vessel is shortened, the resistance will decrease and flow will increase. In contrast to length, the diameter of blood vessels changes throughout the body, according to the type of vessel. A slight increase or decrease in diameter causes a huge decrease or increase in resistance. This is because resistance is inversely proportional to the radius of the blood vessel (one-half of the vessel's diameter) raised to the fourth power ( $R = \frac{1}{r^4}$ ). This means, for example, that if an artery or arteriole constricts to one-half of its original radius, the resistance to flow will increase 16 times. And if an artery or arteriole dilates to twice its initial radius, then resistance in the vessel will decrease to 1/16 of its original value and flow will increase 16 times.

## Poiseuille's equation and Compliance

Jean Louis Marie Poiseuille was a French physician and physiologist who devised a mathematical equation describing blood flow and its relationship to known parameters. The same equation also applies to engineering studies regarding the flow of fluids. This equation is mostly focused on the three critical variables: radius ( $r$ ), vessel length ( $L$ ), and viscosity ( $\mu$ ).

$$\text{Poiseuille's equation Blood flow} = \frac{\pi \Delta P r^4}{8 \mu L}$$

$\Delta P$ : represent the pressure difference.  
 $r$ : the radius of the vessel to the fourth power.  
 $\mu$ : the viscosity of the blood.  
 $L$ : the length of a blood vessel.

One of several things this equation allows us to do is calculate the resistance in the vascular system. Normally this value is extremely difficult to measure, but it can be calculated from this known relationship:

Blood flow =  $\frac{\Delta P}{\text{Resistance}}$

If we rearrange this slightly,

Resistance =  $\frac{\Delta P}{\text{Blood flow}}$

Then by substituting Pouseille’s equation for blood flow:

Resistance =  $\frac{8\mu L}{\pi r^4}$

By examining this equation, you can see that there are only three variables: viscosity, vessel length, and radius, since 8 and  $\pi$  are both constants. The important thing to remember is this: Two of these variables, viscosity and vessel length, will change slowly in the body. Only one of these factors, the radius, can be changed rapidly by vasoconstriction and vasodilation, thus dramatically impacting resistance and flow. Further, small changes in the radius will greatly affect flow, since it is raised to the fourth power in the equation.

The heart as a network flow

In order to derive the whole cardiac cycle (One cardiac cycle is defined as the contraction of the two atria followed by contraction of the two ventricles (Sequence of events that occur when the heart beats)) as a network flow, we need to identify several important components as the basic assumption of a network. The basic setup for a graph is given by  $G(N, A)$ , where  $N$  is the node set, and  $A$  represent the connectivity between any two nodes in the node set (arc), and the flow of the network is denoted by  $x$ , where each arc has its corresponding upper capacities  $u_{ij}$ .

Basic formulations

Given the Basic Anatomy and Physiology of the heart Section, we know that the heart pumps oxygenated blood from the left ventricle to the organs and tissues of the body courtesy of the arteries. In this application, we setup the node as any tissue or organ of the body that receives the oxygen and nutrients from the delivery of the oxygenated blood. We also let the blood vessel (arteries and veins) to be a direct arc that transmits the blood to flow to another body part. The compliance of the blood vessel will then determine the upper capacity for each arc,  $u_{ij}$ . The following table shows the basic setup for the graph:

$N$ : node set , important tissue or organ of the body that receives the oxygen and nutrient from the oxygenated blood.  
 $A$ : arc set(direct) , blood vessel(arteries and veins).  
 $x$ : a flow, unit determined by Cardiac Output or blood pressure divided by the total peripheral resistance.  
 $u_{ij}$ : upper capacity, determined by the compliance of the blood vessel (arteries or veins).

Given the laws of fluid mechanics<sup>[2]</sup>, we consider the blood flow in a single distensible vessel subjected to constant, nonpulsatile flow. For ease of exposition, assume Poiseuille flow (pressure induced flow) within the vessel. If a vessel segment is modeled as a distensible right circular cylinder and entrance effects are ignored, the upper capacity of each arc can be determined by



$$u_{ij} = dP_{ij} = \frac{128\mu}{\pi D^4} F \cdot dL_{ij} > 0$$

where  $P$  denote the pressure,  $L$  is vessel length,  $\mu$  is blood viscosity,  $F$  is vascular flow, and  $D$  is vessel diameter. Based on the assumption corresponding to the structure of the circulation, there is no negative cycle in our graph.

Problem derivation

It is a fact that if we have an increase in healthy blood flow, then the increased delivery of oxygen and nutrients will accelerate the healing process given any open wound injury<sup>[4]</sup>.

Given our definition of cardiac cycle and cardiac output from an earlier section and our goal which will then be to try and maximize blood flow to a specific body part that will result in having a bigger delivery of blood and oxygen in the capillary of the specific body part, we have the following setup that will help allow us to derive the problem as a max flow problem:

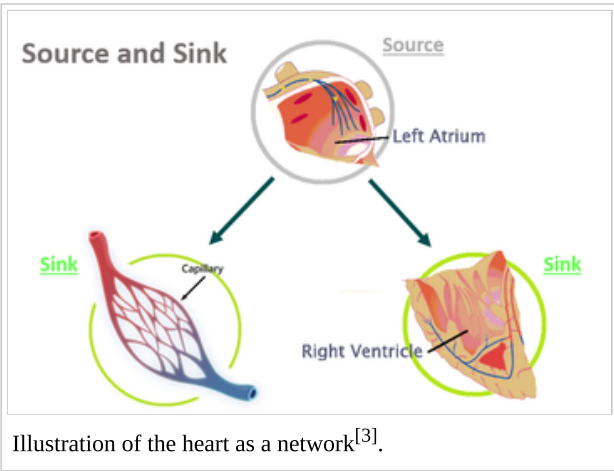
max  $v$

subject to

$$\sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = \begin{cases} v, & \text{for } i = s \\ 0, & \text{for all } i \in N - \{s, t\} \\ -v, & \text{for } i = t \end{cases}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \text{for each } (i, j) \in A$$

These basic constraints also meet the conservation law in fluid mechanics. To solve this particular max flow problem, we need to set the source node  $s$  and the sink node  $t$  which will be sending out the blood flow and the node we want to maximize the blood flow. The circulatory system will then start from the left ventricle (source node) and will then pump the oxygenated blood depending on the different goal(specific tissue) that we would like to achieve, in which we will then have a different setup of the sink node and the arcs for the max flow problems:



Cardiac cycle as different networks

Type of Circulation	Networks		
Systemic Circulation	<b>Network #1</b> Source Node: Left Ventricle Sink Node: Right Atrium Arcs: Arteries and Veins	<b>Network #2</b> Source Node: Left Ventricle Sink Node: Capillaries of issue Arcs:Arteries	<b>Network #3</b> Source Node: Capillaries Sink Node: Right Atrium Arcs:Veins
Pulmonary Circulation	<b>Network #4</b> Source Node: Right Ventricle Sink Node: Left and/or Right Lung Arcs: Pulmonary Artery	<b>Network #5</b> Source Node: Left and/or Right Lung Sink Node: Left Atrium Arcs:Pulmonary Veins	

In this project, we focus on the second network of systemic circulation.

Simplified model

The traditional methodology for solving the heomodynamics problem has a very large amount of computational cost. From the previous section, these concepts may help in describing blood flow system given the systemic circulation. However, the complexity of the vessel may lead to a huge network wherein it will increase quite a bit of construction for a node adjacency model. In the following section, we used geometric properties of arteries to simplify the network<sup>[2]</sup>.

## Blood vessel as network

There are networks that can be used in the modeling of blood flow that are incorporated with other physiological processes in tubular structures<sup>[5]</sup>. In order to help simplify the intricate blood vessel model we can break into sub-problems, which will include all the important nodes such as the capillaries and lungs. We will neglect the small capillary on the aorta in which it has a relatively small radius.

We use the above formulation to simplify and construct a network that can derive the blood flow, and since the end of the capillary might not have the sink node connect to them, therefore, we add the sink node that connect with the end of the capillary.

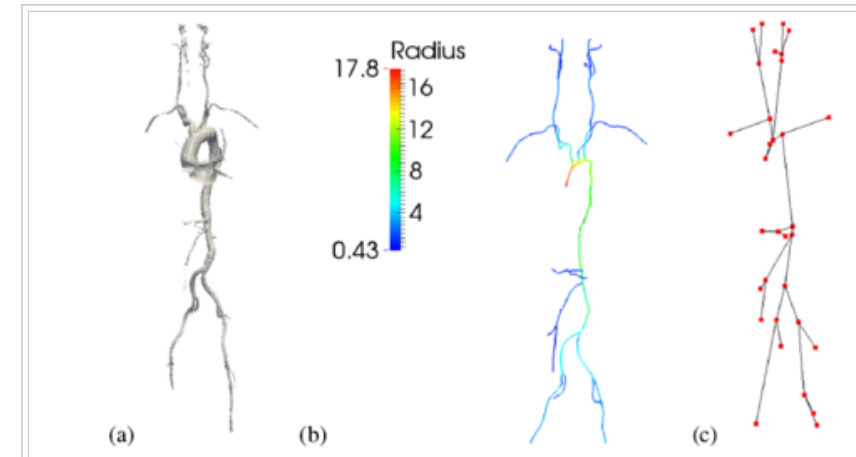
To solve such a maximum flow problem, we will use the pre-flow push algorithm to solve this problem, based on the branching structure of the graph.

## Conclusion and Future Work

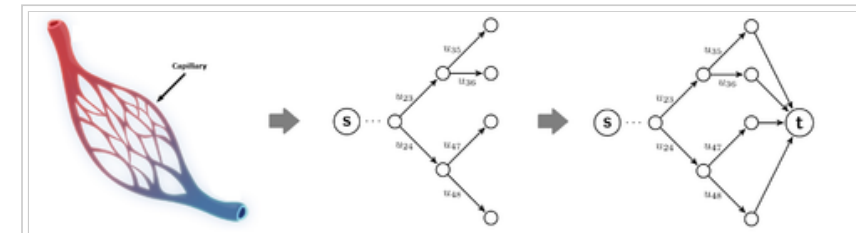
This project has introduced a computationally economical method in attempting to increase blood flow from the human heart towards a specific body part in which it decreases the cost in building the structure of the blood vessel and saving the computational cost from solving the conservation equations. However, given certain time restraints and so many other variables that still needs to be considered, there may still be several questionable factors to consider. For example, the upper bound of each arc may not be rational in which this might lead the algorithm into an infinity loop. Also, this derivation did not factor in certain properties of hemodynamics such as the velocity-dependent viscosity of blood and also, the branching of the capillary may not have been accurately computed especially since we needed more time and better computational instrumentation. Given our future research, we do hope to equate for other variables (Cardiac Output, Total Peripheral Resistance, Compliance, Blood Pressure) regarding maximizing blood flow while minimizing the cost of the heart function and we will also investigate into incorporating such algorithms as Dijkstra's Algorithm and the Ford-Fulkerson Algorithm<sup>[7]</sup> while considering maximum flow and circulation minimum cost flow problems in order to use these modalities regarding increasing blood flow to a certain body part, or looking at certain variables given their costs during blood flow in hopes of detecting strokes and cardiovascular disease or manipulating maximal and minimal blood flow in certain areas that will allow for optimal surgery planning.

## References

- <sup>↑</sup> [1] (<https://www.arheart.com/heart-health/>), Arkansas Heart Hospital:Heart Health.
  - <sup>↑</sup> <sup>2.0</sup> <sup>2.1</sup> [2] (<https://www.physiology.org/doi/abs/10.1152/ajpheart.00762.2002>)Krenz, Gary S., and Christopher A. Dawson. "Flow and pressure distributions in vascular networks consisting of distensible vessels." *American Journal of Physiology-Heart and Circulatory Physiology* 284.6 (2003): H2192-H2203.
  - <sup>↑</sup> [3] (<http://www.edoctoronline.com/medical-atlas.asp?c=4&id=22164>), edoctoronline:Natural pacemaker of the heart.
  - <sup>↑</sup> [4] (<https://www.unitypoint.org/livewell/article.aspx?id=2d59582b-3594-4e51-95d7-10d3f735bcc7>), LiveWell: Blood, Oxygen & Wound Healing: How It Works.
  - <sup>↑</sup> <sup>5.0</sup> <sup>5.1</sup> [5] (<https://onlinelibrary.wiley.com/doi/full/10.1002/cnm.2754>)A. Danilov, Yu. Ivanov, R. Pryamonosov and Yu. Vassilevski. "Methods of graph network reconstruction in personalized medicine " *International journal for numerical methods in biomedical engineering*, 2016, 32.8. .
  - <sup>↑</sup> [6] (<https://en.wikipedia.org/wiki/Capillary>), Capillary: From wikipedia.
  - <sup>↑</sup> [7] (<http://eprints.lincoln.ac.uk/23687/1/23687%20EXTRACTION%20OF%20ARTERIAL%20AND%20VENOUS%20TREES%20FROM%20DISCONNECTED%20VESSEL%20SEGMENT.pdf>)Qureshi, Touseef Ahmad. Extraction of arterial and venous trees from disconnected vessel segments in fundus images. Diss. University of Lincoln, 2016.
- Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Derivation\_of\_Blood\_Flow\_as\_a\_Network\_Flow&oldid=1505"



Reconstruction of large arteries: (a) segmentation, (b) extracted centerlines, and (c) 3D core graph.<sup>[5]</sup>



Derivation of systematic circulation to graph.<sup>[6]</sup>

- This page was last modified on 4 May 2018, at 02:15.
- This page has been accessed 33,108 times.

# DijkstraHeapImplementation

From CU Denver Optimization Student Wiki

Colin Furey Spring 2024 Topics in Combinatorial Optimization Project

## Contents

- 1 Introduction
  - 1.1 Abstract
  - 1.2 Recap of Dijkstra's Algorithm
- 2 Alternative Implementations
  - 2.1 Dial's Implementation
  - 2.2  $\Delta$ -Heap Implementations
- 3 Github
  - 3.1 References

## Introduction

For the spring 2024 section of combinatorial optimization I worked on a project exploring a few different implementations of Dijkstra's shortest path algorithm. This algorithm determines all shortest paths between a designated source node and all other nodes in a network with nonnegative arc lengths. Through the use of clever data structures, the implementations explored in this project either reduce the running time of the algorithm in practice or focus on improving the algorithm's worst case complexity.

## Abstract

The shortest path problem is one of the most common problems in network flows. This can be attributed not only to the ubiquity of practical applications where determining the shortest (alternatively cheapest, quickest) path between two points in a network lies at the heart of a problem, but also because the shortest path problem oftentimes will arise as a sub-problem when one is solving other, more complex combinatorial optimization problems in a given network. One of the most common algorithms for solving the shortest path problem is Dijkstra's algorithm. This is a label-setting algorithm capable of determining the shortest path between a designated source node and all other nodes in a network with nonnegative arc lengths. In this project we explore three variations of Dijkstra's algorithm where researchers have shown that by using slightly different data structures either the practical running time or the worst case complexity of the algorithm can be improved upon.

## Recap of Dijkstra's Algorithm

Given a directed network  $G(V, E)$  with nonnegative arc lengths and source node  $s \in V$ , Dijkstra's algorithm computes the shortest path between the source and all other nodes. This is accomplished by maintaining an array of distance labels  $d(i)$  for each node  $i$  in the network. At each iteration the node set  $V$  is partitioned into two sets:  $S$  and  $\bar{S}$ . The nodes  $i \in S$  are those which are permanently labeled and so  $d(i)$  is the length of the shortest path from  $s$  to  $i$ . The nodes in  $\bar{S}$  are designated as temporarily labeled; for  $i \in \bar{S}$ , the label  $d(i)$  is an upper bound on the length of the shortest path between  $s$  and  $i$ . The algorithm begins by assigning a permanent label of 0 to the source node  $s$  and temporary labels of  $\infty$  to all other nodes in the network, so that initially  $S = \{s\}$  and  $\bar{S} = V \setminus \{s\}$ . It then proceeds to fan out from the source to adjacent nodes  $j \in A(s)$  and changes their temporary labels from  $\infty$  to the length of the arc between  $j$  and  $s$ . Then, at the next iteration we select that node in  $i \in \bar{S}$  whose label is minimum, designate this label as permanent and then update the temporary label of nodes in the adjacency list  $A(i)$ . Once all nodes have been moved from  $\bar{S}$  to  $S$  the algorithm terminates. The correctness of the algorithm relies upon proving the twin inductive hypotheses: (1) that the distance labels for all nodes in  $S$  are in fact true shortest path distances and (2) if  $i \in \bar{S}$  and the temporary distance label corresponds to a path whose internal nodes consists solely of nodes in  $S$ , then this distance label does in fact correspond to the length of the shortest path. The validity of these two hypotheses is what allows us to designate the node with minimum temporary label as permanent, the crux of Dijkstra's algorithm. <sup>[1]</sup>:

In it's simplest form Dijkstra's algorithm maintains the distance labels in a simple array that needs to be scanned a total of  $n$  times, where  $n$  is the number of nodes in the network. There are two fundamental operations involved in updating the array:

1. Node selection: In order to determine which temporary label is minimum, the algorithm must scan all temporary labels. This operation must be performed  $n$  times and since at each iteration one node is moved from temporary to permanent the node selection time is given by

$$n + (n - 1) + \cdots + 1 = O(n^2).$$

2. Distance updates: After designating node  $i$ 's label as permanent, we must then update all temporary labels of nodes in  $A(i)$ . Each of these updates requires  $O(1)$  time and since this must be done for each node in the network the algorithm requires  $O(m)$  time for all distance updates. Since

$$m \leq \binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$$

putting all of this together implies that solving the shortest path problem with Dijkstra's algorithm in its simplest form requires  $O(n^2)$  time. The main bottleneck here is node selection. In what follows we explore some alternative data structures for storing our temporary labels that can reduce the time requirement of node selection.

# Alternative Implementations

## Dial's Implementation

As mentioned above, the main bottleneck in terms of computation time for Dijkstra's shortest path algorithm is the node selection procedure<sup>[2]</sup>. In effect, on each iteration the temporary labels for nodes in  $\bar{S}$  need to be sorted in order to identify the smallest temporary label and designate it as permanent. Dial's implementation of Dijkstra's algorithm addresses this bottleneck by storing temporary distance labels in a sorted fashion, thereby bypassing the need to sort all temporary labels at each iteration. The ability to do so relies upon the clever observation that the distance labels designated as permanent throughout the algorithm form a nondecreasing sequence.

To see why, consider a node  $I$  that is permanently labeled  $d(i)$  on some iteration. The next step for Dijkstra is to scan the nodes in  $A(i) = \{j \in V : (i, j) \in E\}$  and determine whether or not the temporary labels need to be updated. If a label for  $j \in A(i)$  does in fact need to be updated, we relabel it as

$$d(j) = d(i) + c_{ij} \geq d(i)$$

where the inequality follows since all arc lengths  $c_{ij}$  are assumed to be nonnegative.

Instead of storing the temporary distance labels in a simple array, Dial's implementation<sup>[3]</sup> stores them in a an array of  $nC + 1$  "buckets", where  $C$  denotes the largest arc length in the network. Since there are  $n$  nodes in the network,  $nC$  represents an upper bound on the distance label of any finitely labeled node and this allows us to store those nodes with temporary label  $k$  in bucket  $k$  for  $k = 0, 1, \dots, nC$ . Thus, in the node selection step we need not examine all temporary labels to find the minimum but can instead scan the buckets  $0, 1, \dots$  and so on until we find the first nonempty bucket.

If the  $k$ th bucket is the first nonempty bucket, then since all nodes in this bucket have the same temporary label  $k$  and they're all minimal, we can designate each node in the bucket with the permanent label  $k$  and update the temporary labels of all its adjacent nodes. At the next iteration, we need only scan buckets  $k + 1, k + 2, \dots$  since as we mentioned above, all of the updated labels are at least  $k$ . Checking whether a bucket is empty or not, deleting a node from a bucket, adding a node to a bucket and distance updates are all  $O(1)$  operations. It follows that the distance updates require  $O(m)$  time and the scanning of the  $nC + 1$  buckets is  $O(nC)$ , implying Dial's implementation is  $O(m + nC)$ .

Some potential disadvantages to Dial's implementation compared to Dijkstra's original implementation is that if  $C$  is large, a large amount of storage is required to be allocated for the  $nC + 1$  buckets. Moreover, the running time of the algorithm is pseudopolynomial so if  $C$  is in fact large relative to  $n$ , the running time of Dial's implementation is strictly worse than Dijkstra's; e.g. if  $C = n^2$ , then Dial's implementation is  $O(n^3)$ .

## $d$ -Heap Implementations

Improvements in the running time can be achieved via even more clever data structures for storing of the nodes in  $\bar{S}$  and their corresponding temporary distance labels. One such example is the  $d$ -heap. Before discussing how  $d$ -heap data structures can be utilized in Dijkstra's algorithm, first we briefly discuss these data structures in their own right.  $d$ -heaps are data structures capable of efficiently storing and manipulating a collection  $H$  of objects where each object  $i \in H$  has an associated real number key, denoted  $key(i)$ . The manipulations are relatively basic but include

<sup>[1]</sup> Create-Heap: Creates an empty heap.

- $\text{find-min}(i, H)$ : Return the object  $i \in H$  of minimum key.
- $\text{insert}(i, H)$ : add a new object  $i$  to  $H$  with predefined key.
- $\text{delete}(i, H)$ : delete object  $i$  from the heap.
- $\text{decrease-key}(i, H)$ : Reduce the key of object  $i \in H$  from current value to  $v$ . Only defined when current value is greater than  $v$ .
- $\text{increase-key}(i, H)$ : increase the key of object  $i$  from current value to  $v$ . Only defined when current value is less than  $v$ .
- $\text{delete-min}(i, H)$ : Delete the object  $i$  with minimum key.

Assuming a  $d$ -heap has  $n$  objects, the operation  $\text{find-min}$  runs in  $O(1)$  time, the insert and decrease-key operations run in  $O(\log_d n)$  time and the delete, delete-min and increase-key operations run in  $O(d \log_d n)$  time.

In most applications of heaps to problems in network flows, the elements of  $H$  are the nodes and their corresponding keys are some kind of label associated with a node.  $d$ -heaps are tree-based data structures where each node in the has up to  $d \geq 2$  children. In the context of Dijkstra's algorithm, the elements of  $H$  are those nodes with a finite temporary distance label and their corresponding keys are their current label. The  $d$ -heap allows us to efficiently maintain a priority queue of nodes that have yet to be visited but are candidates for the next step of the algorithm. Given a graph  $G(V, E)$  with  $m$  edges,  $n$  nodes and source node  $s$  the following pseudocode is an example of how a  $d$ -heap can be utilized in Dijkstra's algorithm:

```

algorithm heap-Dijkstra;
begin
  create-heap( $H$ );
   $d(j) := \infty$  for all  $j \in N$ ;
   $d(s) := 0$  and  $\text{pred}(s) := 0$ ;
  insert( $s, H$ );
  while  $H \neq \emptyset$  do
    begin
      find-min( $i, H$ );
      delete-min( $i, H$ );
      for each  $(i, j) \in A(i)$  do
        begin
           $\text{value} := d(i) + c_{ij}$ ;
          if  $d(j) > \text{value}$  then
            if  $d(j) = \infty$  then  $d(j) := \text{value}$ ,  $\text{pred}(j) := i$ , and insert ( $j, H$ )
            else set  $d(j) := \text{value}$ ,  $\text{pred}(j) := i$ , and decrease-key( $\text{value}, i, H$ );
          end;
        end;
      end;
    end;
  end;

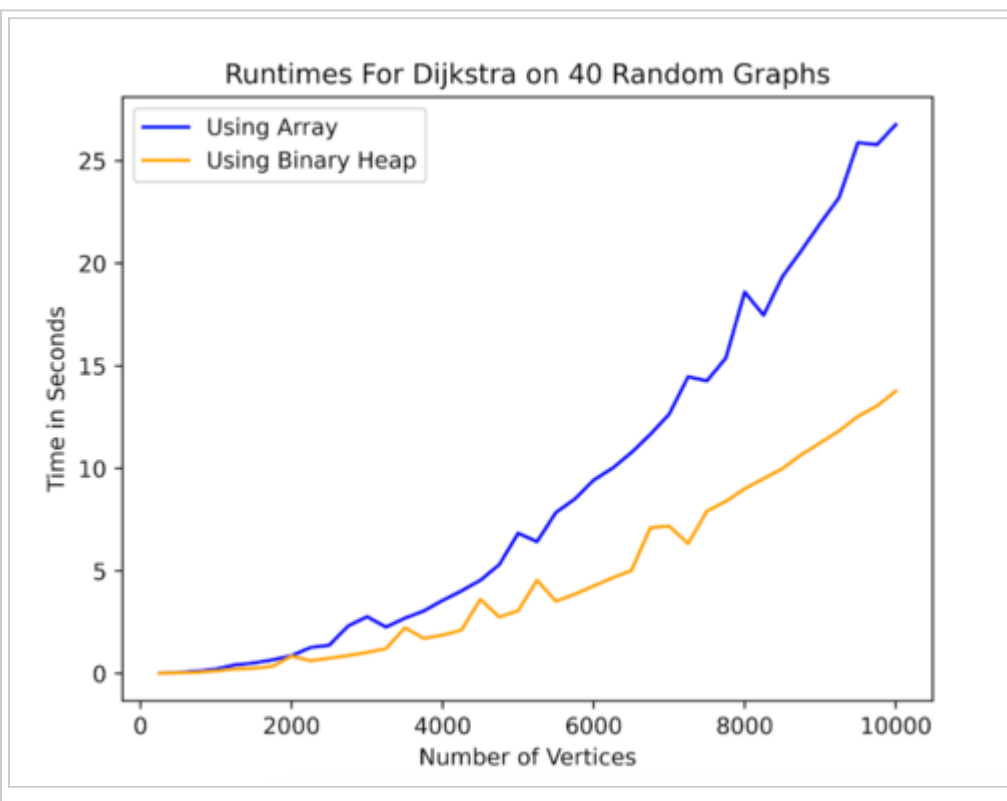
```

When a  $d$ -heap is utilized as the priority queue in Dijkstra's algorithm, the time complexity can be reduced from the original  $O(n^2)^{[2]}$ . The main while loop runs  $n$  times as each node needs to be processed at least once. Inside this loop, we need to perform the extract-min operations  $n$  times, yielding a time complexity of  $O(n)$ . We also need to perform the delete-min operation  $n$  times, yielding a time complexity of  $O(nd \log_d n)$ . In the worst case, we need to perform the operation decrease-key for each neighbor of the current vertex, which would imply that the total number of edges processed throughout all iterations is  $m$ . Then the total time complexity for processing all edges is  $O(m \log_d n)$ . Thus, it follows that the total time complexity of the algorithm is  $O(m \log_d n + nd \log_d n)$ .

As one can imagine, the choice of  $d$  makes a difference when it comes to efficiency. To determine the optimal value of  $d$  we set the two terms in the sum equal and solve for  $d$ . Doing so obtains  $d = \min\{2, \lceil \frac{m}{n} \rceil\}$  where the ceiling function comes in since  $d$  is an integer. Using this choice of  $d$ , the running time reduces to  $O(m \log_d n)$ . For sparse networks where  $m = O(n)$ , the running time is improved to  $O(n \log_d n)$ . For dense networks where  $m = O(n^{1+\epsilon})$  for some  $\epsilon > 0$ , the running time can be shown to be  $O(m)$ , which is optimal.

For Dijkstra's algorithm, a common choice is  $d = 2$ . This special case is referred to as the binary heap since each node can have at most two children. All  $d$ -heaps can be stored as arrays but the memory footprint of  $d$ -heaps increases with the value of  $d$ . The binary heap is a common choice because it requires storage proportional to the number of objects in  $H$ . In the context of Dijkstra's algorithm the storage requirement is proportional to the number of vertices.

In order to examine how Dijkstra's original implementation compares with the theoretically more efficient binary heap implementation, both were used to solve the shortest path problem on 40 random graphs containing between 10 and 10,000 vertices using the python programming language. For Dijkstra's original implementation, we loop through an array of distances on each iteration to find the minimum temporary label while for the heap implementation python's `heapq` (<https://docs.python.org/3/library/heapq.html>) module was utilized. As can be seen in the plot below, the savings in terms of runtime are significant when the binary heap is employed.



Github Containing Code Used for Dijkstra Heap Implementation Project (<https://github.com/fureyc/Dijkstra-Project>)

## References

1. <sup>↑</sup> Dijkstra, E.W. A note on two problems in connexion with graphs. Numer. Math. 1, 269–271 (1959)
2. <sup>↑</sup> <sup>2.0</sup> <sup>2.1</sup> R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: Theory, algorithms, and applications*. Prentice Hall Inc., Englewood Cliffs, NJ, 1993.
3. <sup>↑</sup> Dial, Robert B. Algorithm 360: Shortest-path forest with topological ordering. Communications of the ACM. 12 (11): 632–633 (1969). Retrieved from "<https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=DijkstraHeapImplementation&oldid=4825>"

- 
- This page was last modified on 2 May 2024, at 11:20.
  - This page has been accessed 84 times.



# Disaster Evacuations for Colorado

From CU Denver Optimization Student Wiki

The authors of this project are Drew Horton and Nicholas Crawford.

## Contents

- 1 Abstract
- 2 Motivation
- 3 Methods
- 4 Data
- 5 Results
- 6 Policy Recommendations
- 7 Limitations of Algorithm
- 8 References

## Abstract

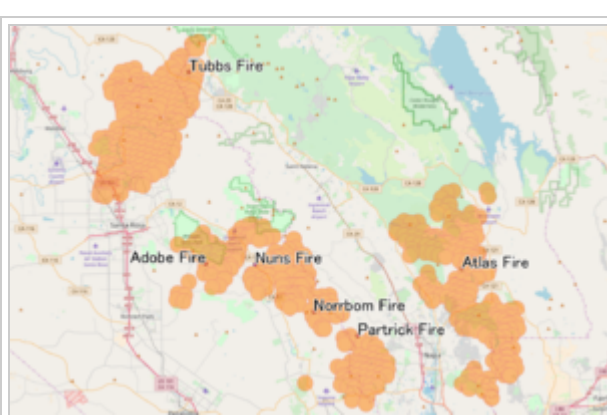
During disaster evacuations, it is crucial to have an evacuation plan that allows people to find safety in the shortest amount of time. In 2018, California experienced its most deadly fire to date, the Camp Fire, in which 85 civilians died. Due to the capacity of the roadways along evacuation routes, traffic jams trapped residents in their cars, killing at least 7 people. Colorado is no stranger to disaster. Last year there were 1,017 fires reported by the National Interagency Fire Center which burned a total 48,195 acres. This year we have already seen both the Marshall Fire and the NCAR fire force thousands of residents out of their homes.

Our project uses methods from optimization to identify 'bottlenecks' in evacuation routes so that we can ensure that every resident can safely evacuate during a disaster. We model the potential evacuation routes as a network and solve a maximum flow problem. The solution to this maximum flow instance identifies both the capacities and locations of bottlenecks in our network. This information can inform policy makers on areas to consider expanding capacities to reduce the risk of a traffic jam in evacuation situations.

## Motivation

The authors of this project were both affected by wildfires in California. There are more than 7000 wildfires per year in California and on average 1.2 million acres have been burned each year over the past 5 years. The deadliest fire in California's history, the Camp Fire, occurred in 2018 when the town of Paradise burned down. Due to high winds, the fire spread at a rate of 7500 acres an hour at its peak<sup>[1]</sup>. By the time the fire was contained it had destroyed 18804 structures, killed 85 people, and burned over 153,336





Map of 2017 Fires in Northern California

acres<sup>[2]</sup>. In addition to the destruction, there were many challenges during evacuations. One such being that roadways were congested with cars, causing residents to abandon their vehicles and resort to evacuating on foot. The abandoned of vehicles further exacerbated the traffic issues.

Now both authors live in Colorado, which is no stranger disaster, and wildfires are not an exception. Not only have 15 of the top 20 largest wildfires in Colorado's history occurred in the last decade, but also the top 4 of 5 wildfires happening since 2018<sup>[3]</sup>. With the Marshall fire happening a in December of 2021 and over 1000 homes destroyed, and being affected by wildfires ourselves, we decided to try and explore a way for people to be prepared if a wildfire started near them. The goal

of the project is to find evacuation routes in which the maximum amount of people can escape the fire boundary.

Another motivation for this project was the Love Parade disaster in Duisburg Germany. The Love Parade was a free music festival is which over a million people attended. The disaster happened when people sought to get into the delayed festival, but the problem was there was only one entrance and exit to the festival. Once people were allowed into the festival it created a stampede of rushing people in which several people were killed and hundreds more injured. In addition to wildfire evacuation our project could be extended to make evacuation routes for buildings, festivals, etc...

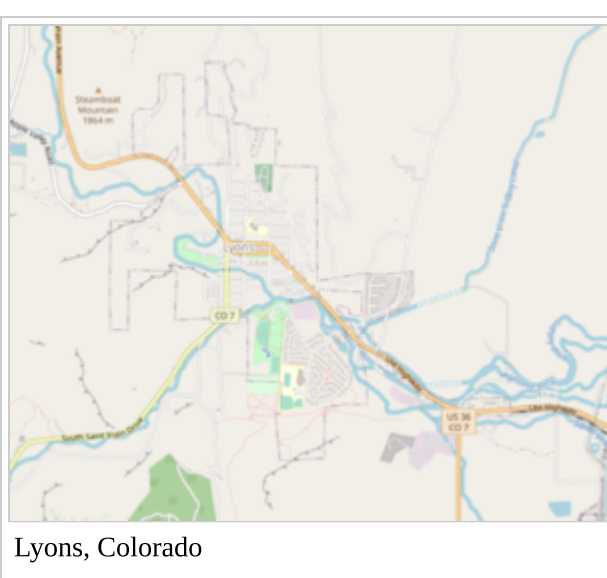
## Methods

The goal of this project is to get as many people to safety in the shortest amount of time, therefore, we decided to model evacuation routes as a maximum flow problem. There are two main types of algorithms when talking about solving maximum flow problems; preflow push algorithms and augmenting path algorithms. For this project, we implemented an augmenting path algorithm--specifically the shortest augmenting path algorithm. In general, in an augmenting path algorithm we push flow along the 'shortest' paths from a source (start) to a sink (end), while making sure to not surpass the capacities of the networks. (In the case of our application, this is the capacities of the roads). The shortest augmenting path algorithm selects a shortest path in the residual network and augments as much flow as possible along the path. Then, it updates our residual network and continues iterating through this same process until no more s-t paths exist.

For our specific program, we chose to choose each shortest s-t path by travel time. The travel time for each arc (road) was computed using OSMNX (see data section). In addition to considering the travel time, we also had to calculate the capacity of each arc. In 2013 Moore et. al published a method to calculate the capacities of roadways based on safe breaking distances between vehicles, speed limits, and the number of lanes. We chose this method to calculate capacities of the roads in our network because during evacuations, if there are blockages such as crashes on roads or construction, the algorithm could be adjusted to to reduce the number of lanes available. The next consideration was which path to send people on. For example, if 500 people could escape if they took route A, but it took 15 minutes versus 700 people could escape on route B, but it took 25 minutes. What path should be chosen for the evacuation route? The way that we made this decision was based on two factors. First, what was the population density of the area that was being evacuated. Take for instance Lyons, Colorado which has a population of around 2300 people in 800 households. In many cases there were only 1 or 2 routes leaving the town away from the fire zone. So, by taking the most densely populated neighborhood (roughly 500 households) and sending them along Highway 66 which leads to the larger Highway 36, that would leave the other 300 households to travel along Highway 7 . In this case even though that neighborhood was closer to Highway 7 by roughly 5 minutes it would be more beneficial for them to travel along Highway 66 so that the road would not become backed up. The second factor was the evacuation route density. For example there were over 50 routes in Denver area for residents to evacuate along if the fire started near 16th Street Mall. All of these routes had travel times within



Aftermath of the Camp Fire in Paradise, CA



10 minutes of each other, but the problem was that the capacity of these routes were much smaller since they used city streets. In this scenario we would recommend splitting groups of people along multiple different routes to try and not clog any particular roadway but allow flow even if it meant travel time was longer. Now that we have covered what the algorithm does not consider we will talk through our algorithm process. The pseudocode is listed in Figure 3. For our full code please see our github

Input: Capacities, Centerline data, City/ County, Source and sink nodes

([https://github.com/drewhort/disaster\\_evacuations\\_for\\_colorado](https://github.com/drewhort/disaster_evacuations_for_colorado)) Our algorithm starts by calculating the distances using the above method of Moore et al. We then take in our data (see next section) and add capacity to our data frame. Using the osmnx package in python we then create our original graph with nodes, edges, capacities and travel times. We then calculate the shortest path from our designated source node to our sink node. The sink and source node are determined by where the evacuation starts to outside the fire area respectively. We then find a feasible flow along our shortest path and augment the flow. We now update the residual network. This will be one iteration of our algorithm. We continue iterating until there are no more paths from our source to our sink node. We then plot all of these shortest paths and overlay a street map to get our evacuation route.

## Data

The data needed for the project consisted of a list of streets, speed limits, and how many lanes per road. We used Open Street Maps (OSM) ([4]) to collect this data. OSM is a community open source data hub that uses GIS and aerial imagery to ensure up to date and accurate information. We also needed the gather fire data regarding different fire boundaries. Boulder County Geospatial Open Data contains different data on plats and geographical surveys. Our last piece of data that we needed was to determine the population density of a given town. We utilized IPUMS (Integrated Public Use Microdata Series) National Historical GIS ([5]) which is a data base that provides different census and survey data using GIS systems.

In order to utilize our algorithm we had to construct a graph from our data. First, we imported our data using the python package OSMNX<sup>[6]</sup>. Using the imported data we calculated the capacities by the method above. Next, we had to clean the data by removing intersections on our graph that were not really intersections. We then added edge speeds, travel times, and capacities to our data frame. Then, using OSMNX we constructed our graph. Knowing that our graph would be constantly changing as we updated our residual network we created a copy so we could overlay the shortest paths when the algorithm was finished.

```

Input: Capacities, Centerline data, City/ County, Source and sink nodes
Output:
Create Graph from input
Shortest path list= [ ]
Calculate Shortest s-t paths
    Shortest path list= list_shortest s-t paths
    While shortest path list is not empty
        Augment flow along shortest path
        Update residual network
        Calculate shortest s-t path along residual network
        Append shortest path list
Plot shortest path list
Plot residual network

```

## Pseudocode



## Results

The two main cities that we looked were Lyons and Pueblo West. We choose these cities based on the criteria that they: only have a few entrances and exits, close to high fire danger, have/next to a large population density <sup>[7]</sup>. The first city we will look at is Lyons, Colorado. As mentioned above, Lyons only has a few entrances and exits but is next to a massive high fire wilderness area. We decided to set our start node as the center of Lyons and see the maximum flow to the Boulder city evacuation area. Our maximum flow algorithm found several shortest paths (Figure 3) to the evacuation area. With all of these routes we were able to find a maximum flow of approximately 1950 vehicles an hour. What this means for us is that every household in Lyons would be able to take at least one vehicle if we split the maximum flow over our three evacuation routes in Figure 3. It is worth noting that the maximum flow is so high because Lyons is next to highway 36 which is a major highway that has a high capacity theoretically. The next town that we looked at was West Pueblo, Colorado. Similar to Lyons, West Pueblo has limited entrances and exits and is located in a high fire danger wilderness area. For the start node of this instance we decided to start in a neighborhood that backs up to one of the major fire areas. In Figure 5 you can see the shortest paths from this node to the evacuation site in Pueblo. The problem is that West

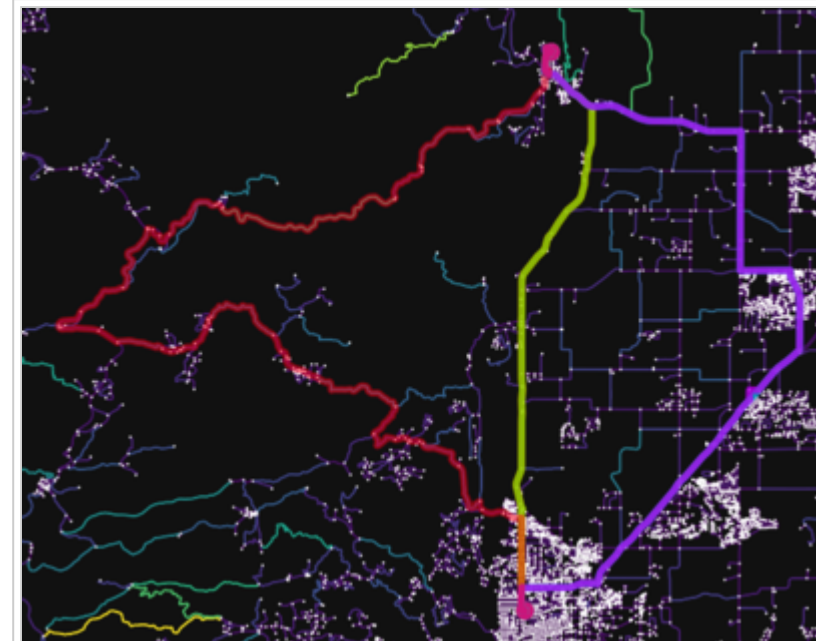
Pueblo has a population of approximately 32,000 people, but using all of these evacuation routes, only 1800 vehicles an hour can fill those roads. That means it would take over 10 hours to have everyone evacuated from West Pueblo (assuming that each household has 3 people). This is problematic since West Pueblo is surrounded on all sides by wilderness area allowing fires to surround the town easily. Another issue is that many of the evacuation routes stem by going West first. Too the west of town lies a river and the wind moves South West. So in the result of a fire that stemmed on the west side of town some of the evacuation routes would be blocked.

## Policy Recommendations

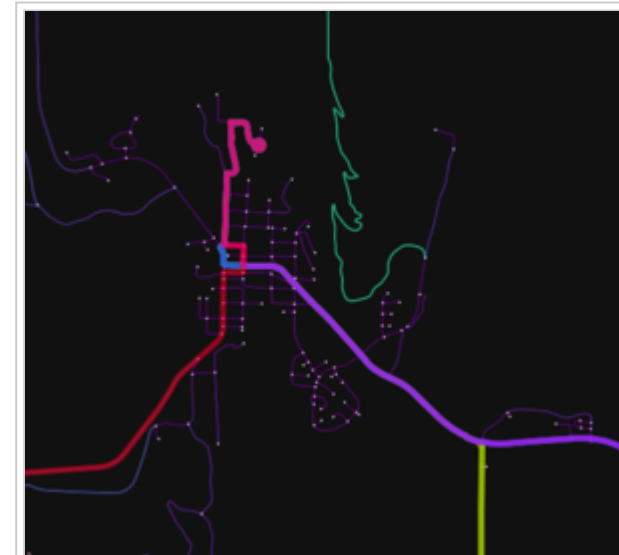
Our first type of policy recommendation is to make an evacuation plan for towns that do not have one, but are located in high fire danger areas <sup>[8]</sup>. In the case of Lyons, on the city website it says to always be prepared for a fire evacuation but does not have a plan listed. Along these same lines we can use the algorithm to assess current evacuation procedures. By comparing current plans and the plan developed by our algorithm we can assess whether the current procedures are up to date or need to be revised. As populations continue to grow across Colorado, many cities will need to update their



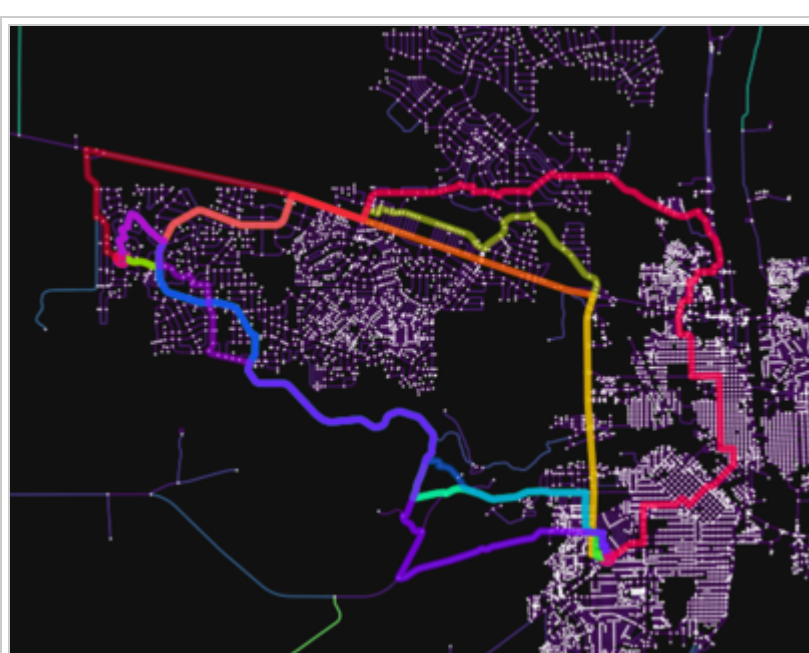
Shortest Path



Lyons Evacuation Route



Zoomed in Lyons Evacuation Route



Pueblo West Evacuation Route

evacuation plans. Utilizing the algorithm to reassess evacuation plans can be an easy way to make sure citizens are prepared in case of an emergency. Our final type of policy recommendation is useful for towns like West Pueblo. In these towns we can identify "bottlenecks" and recommend that either more exits be built out of the town or that current roads need to be expanded.

## Limitations of Algorithm

Given these results, it is important to discuss the limitations of our algorithm. The biggest of these being that our algorithm is based on calculated capacities of roads. As mentioned above the capacities are calculated based on a formula by Moore et al. <sup>[9]</sup> These are theoretical capacities in which everyone is driving at a safe distance and speed. During an evacuation however, these distances and speeds are likely to differ greatly from just a regular day of driving. Collecting data on driving during an evacuation would allow us to calculate realistic capacities, leading to more accurate results.

The other major issue that our algorithm does not cover is that if there is only 1 path from our start node to our end node it just runs one iteration. An improvement for this would be to calculate another shortest path to a different end node that is still outside the fire zone. This would require that the algorithm be able to search for the shortest path to an entire boundary. This could be implemented by taking the centroid of fire area and calculating the shortest path to each boundary point. Once those distances had been calculated we

could run the algorithm to each of those end nodes and choose the one with max flow or most evacuation routes. Along this same problem, we manually have to enter in our start and end nodes to find maximum flow along. Implementing a choice program for people to input there start location and then just it tell them where to evacuate to would be more ideal.

## References

1. ↑ [1] (<https://www.nytimes.com/2018/11/11/us/california-fire-paradise.html>)<https://www.nytimes.com/2018/11/11/us/california-fire-paradise.html>
2. ↑ [2] ([https://www.fire.ca.gov/media/t1rdhizr/top20\\_destruction.pdf](https://www.fire.ca.gov/media/t1rdhizr/top20_destruction.pdf))[https://www.fire.ca.gov/media/t1rdhizr/top20\\_destruction.pdf](https://www.fire.ca.gov/media/t1rdhizr/top20_destruction.pdf)
3. ↑ [3] (<https://dfpc.colorado.gov/wildfire-information-center/historical-wildfire-information>)<https://dfpc.colorado.gov/wildfire-information-center/historical-wildfire-information>
4. ↑ [4] (<https://www.openstreetmap.org/copyright>)<https://www.openstreetmap.org/copyright>
5. ↑ [5] (<https://data2.nhgis.org/downloads>)<https://data2.nhgis.org/downloads>
6. ↑ [6] (<https://geoffboeing.com/publications/osmnx-complex-street-networks/>)<https://geoffboeing.com/publications/osmnx-complex-street-networks/>
7. ↑ [7] (<https://www.streetlightdata.com/limited-emergency-evacuation-routes-map/>)<https://www.streetlightdata.com/limited-emergency-evacuation-routes-map/>
8. ↑ [8] (<https://coloradosun.com/2022/04/20/boulder-marshall-fire-evacuation/>)<https://coloradosun.com/2022/04/20/boulder-marshall-fire-evacuation/>
9. ↑ [9] ([https://www.researchgate.net/publication/285199854\\_Maximum\\_flow\\_in\\_road\\_networks\\_with\\_speed-dependent\\_capacities\\_-\\_Application\\_to\\_Bangkok\\_traffic](https://www.researchgate.net/publication/285199854_Maximum_flow_in_road_networks_with_speed-dependent_capacities_-_Application_to_Bangkok_traffic))[https://www.researchgate.net/publication/285199854\\_Maximum\\_flow\\_in\\_road\\_networks\\_with\\_speed-dependent\\_capacities\\_-\\_Application\\_to\\_Bangkok\\_traffic](https://www.researchgate.net/publication/285199854_Maximum_flow_in_road_networks_with_speed-dependent_capacities_-_Application_to_Bangkok_traffic)

This page was last modified on 3 May 2022, at 14:23.  
This page has been accessed 1,400 times.



# Dongdong Lu

From CU Denver Optimization Student Wiki

Welcome!

I am a graduate student in applied mathematics at the University of Colorado Denver and currently, I am teaching college algebra recitation classes 1110-003 & 004.

I am from Anhui, China, a province with a rich cultural background: it is also the birthplace of China's first Nobel prize Laureate, first electronic computer, and current prime minister.

Also, I am also a big fan of national parks and outdoor activities, paddling in particular. Besides, I play table tennis and was the Intramural Champion last year.

I enjoy reading Scientific American, National Geographic and Chinese literature.

My personal philosophy is deeply affected by Rudolf Carnap and primitive principles of Buddhism.

Life Value: Help less-resourced people in a scientific, systematic and sustainable way.

For fall 2018, I was working with Kushmakar Baral. The link to the project is: Crime & Temperature: Scheduling Awareness Programs

For spring 2019, I am considering to work with Christina Ebben and User:Culvere on Police vs Firefighters or doing the project Optimizing the patrolling route.

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Dongdong\_Lu&oldid=1956"

Category: Contributors

- 
- This page was last modified on 21 March 2019, at 12:28.
  - This page has been accessed 2,135 times.

# Dongdong lu

From CU Denver Optimization Student Wiki

Bio goes here!

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Dongdong\\_lu&oldid=1591](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Dongdong_lu&oldid=1591)"

- 
- This page was last modified on 6 November 2018, at 12:16.
  - This page has been accessed 418 times.



# Drew Horton

From CU Denver Optimization Student Wiki

## Contents

- 1 Contact Links
- 2 About Me
  - 2.1 Education
  - 2.2 Programming Languages/Experience
  - 2.3 Projects

## Contact Links

- Email (<mailto:drew.Horton@ucdenver.edu>)
- Twitter (<https://twitter.com/drewhort>)
- GitHub (<https://github.com/drewhort/>)

## About Me

My name is Drew Horton, and I am a PhD student at University of Colorado Denver. I graduated with my BA in pure mathematics from Sonoma State University in Spring 2019. As an undergraduate I did research in enumerative combinatorics working with a generalization of arithmetic progressions, which we called pseudo progressions. I also did research in Leibniz Algebras at the Institute of Mathematics in Uzbekistan during a summer REU funded by CSU Fullerton. My current research is in optimization, in particular, non-linear programming. My advisor is Dr. Emily Speakman.

## Education

1. Santa Rosa Junior College, A.A. in Spanish, A.A. in Mathematics
2. Sonoma State University, B.A. in Pure Mathematics
3. University of Colorado Denver, M.S. in Applied Mathematics

## Programming Languages/Experience

- AMPL



- Mathematica
- Python

## Projects

In Fall 2020 I worked with Makayla Cowles and Michael Burgher on fighting voter suppression through our project **Location! Location! Where polling places are and where they should be**(Making Voting More Accessible). This project won “Best in Policy” at the Fall 2020 Data to Policy Symposium.

In Spring 2021 I worked with Rebecca Robinson on combating food deserts through our project Hungry for Equality: Fighting Food Deserts. This project was the Honorable Mention for “Data Analysis” at the Spring 2021 Data to Policy Symposium.

In Fall of 2021 I worked on a project Claws and Effect: Finding Strategies to Increase Pet Adoption, where we use sign-compatible circuit walks to figure out a strategy for transferring pets among shelters and rescues in such a way that increases adoption rates.

In Spring of 2022 I worked with Nicholas Crawford on our project **Extinguishing Bottlenecks: Optimizing Fire Evacuation Routes** (Disaster Evacuations for Colorado). This project won grand prize at the Spring 2022 Data to Policy Symposium.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Drew\\_Horton&oldid=3800](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Drew_Horton&oldid=3800)"

Category: Contributors

- 
- This page was last modified on 3 May 2022, at 02:53.
  - This page has been accessed 1,335 times.

# Duality

From CU Denver Optimization Student Wiki

In mathematical programming, duality is the theory that any program can be both viewed and solved from two perspectives. The first is the **primal program** and the second is the **dual program**. Using both of these methods to look at a program provides deeper insight in to the program functions.

## Contents

- 1 The Dual Program
- 2 Constructing the Dual Program
  - 2.1 General Lagrangian Dual Program
  - 2.2 Linear Dual Program
- 3 Interpretations of the Dual Program
  - 3.1 Economic Interpretation
  - 3.2 Dual Program as Bounding
  - 3.3 Analyzing Sensitivity with the Dual

## The Dual Program

The dual program is obtained from the primal program, and can be created and understood in various ways. The dual problem referred to in this section is the Lagrangian dual problem, but there are special cases of dual problems that have constraints on when they can be used. The primal problem is a mathematical program that has yet to be solved. The dual program is the program generated by transposing the negative constraint matrix of the primal problem. Then the objective function and right hand side values of the primal problem are are switched and negated. When attempting to understand the relationship between the primal problem and the dual problem, it is important to note that the dual problem of a dual problem is the primal problem from which the dual is obtained. That is to say, if a problem is dualized and the resulting dual problem is dualized, the resulting problem is the primal problem.

## Constructing the Dual Program

### General Lagrangian Dual Program

Given the primal program P:

$$\min f(\mathbf{x})$$

Typesetting math: 100%

$$\begin{aligned}
s.t. \quad & g_i(\mathbf{x}) \leq 0 & i &= 1, 2, \dots, k \\
& h_i(\mathbf{x}) = 0 & i &= 1, 2, \dots, l \\
& \mathbf{x} \in X
\end{aligned}$$

where  $X$  is the domain of definition for  $\mathbf{x}$ .

\* The Lagrangian is defined to be:

$$L(\mathbf{x}, \mathbf{u}, \mathbf{v}) := f(\mathbf{x}) + \sum_{i=1}^k u_i g_i(\mathbf{x}) + \sum_{i=1}^l v_i h_i(\mathbf{x})$$

\* The Lagrangian Dual function,  $\theta$ , is defined to be:

$$\theta(\mathbf{u}, \mathbf{v}) := \inf \left\{ f(\mathbf{x}) + \sum_{i=1}^k u_i g_i(\mathbf{x}) + \sum_{i=1}^l v_i h_i(\mathbf{x}) : \mathbf{x} \in X \right\}$$

\* And the Lagrangian Dual Problem to solve is defined to be:

$$\begin{aligned}
& \max \theta(\mathbf{u}, \mathbf{v}) \\
& \forall \mathbf{u} \geq 0
\end{aligned}$$

This is the general form for a non-linear program.

## Linear Dual Program

Given a linear program in standard form:

$$\begin{aligned}
 &Max \sum_{j=1}^n c_j x_j \\
 &S.T. \sum_{j=1}^n a_{ij} x_j \leq b_i \\
 &\quad x_j \geq 0 \\
 &\quad i = 1, 2, 3 \dots m \\
 &\quad j = 1, 2, 3 \dots n
 \end{aligned}$$

The dual program is:

$$\begin{aligned}
 &Min \sum_{i=1}^m b_i y_i \\
 &S.T. \sum_{i=1}^m a_{ji} y_i \geq c_j \\
 &\quad y_i \geq 0
 \end{aligned}$$

Notice that the primal  $m \times n$  constraint matrix becomes transposed into the dual  $n \times m$  constraint matrix. The maximization problem becomes a minimization problem, and the right hand side values are switched with the objective function. Lastly, instead of  $\boldsymbol{x}$  as the variable, a new variable  $\boldsymbol{y}$  is introduced to take its place in the new problem. There will be a  $\boldsymbol{y}$  variable for each of the constraints in the primal problem, more information on that is given in the section on bounding. This example is rather simple, because all of the constraints are less than the right hand side value and all of the variables are non-negative. This, however, is rarely the case in practical applications. Each inequality and turns into an inequality in the comparable part of the dual program.

Minimization Problem	Maximization Problem
Constraint	Variable
$\geq$	$\geq 0$
$\leq$	$\leq 0$
$=$	Unrestricted
Variable	Constraint
$\geq 0$	$\leq$
$\leq 0$	$\geq$
Unrestricted	$=$

This table is a reference for how to switch between each component of a primal problem and its respective constraints and variables in the dual problem. Each individual 

Typesetting math: 100%

 are looked at individually, not as a whole matrix.

# Interpretations of the Dual Program

The dual program can be constructed and interpreted in various ways. Each way is more useful for some applications than others, and each gives further insight into the primal program.

## Economic Interpretation

In an economic setting, the program can be thought of as not only a way to maximize profit given certain resources for an entity doing business, but as a dual program minimizing the resource allocation given a certain amount of profit. For example, it takes a certain amount of resources to build a product that can be sold for a profit. However, the resources are limited and other businesses are willing to buy them. The business can then either sell the product or the resources, but if they sell the resources they want to make sure they make the same amount of profit while selling as few resources as possible. The dual problem minimizes resources distributed while constraining the resources to be at least at a certain level of profit. Contrast this with maximizing the profit while only having a certain amount of resources, and the relationship between the primal and dual problems becomes apparent. There is an example of this on its own page.

## Dual Program as Bounding

Any feasible point in a mathematical program is a lower bound on what the optimal objective function value can be for a maximization problem. This follows from the logic that any feasible point is either an element of the Pareto set or has a worse objective function value than any element of the Pareto set. For small problems, like the ones outlined in this Wiki, finding the optimal solution from any feasible point is not difficult. However, for very large and complicated problems, this process becomes harder. In order to bound the problem from the top, a dual program can be constructed. Any feasible point in the dual program provides an upper bound on the optimal objective function value for the primal problem. The space between these feasible points is the duality gap. The goal of programming search algorithms is to close the duality gap to find the optimal objective function value. Many commercial solvers use an algorithm called the primal-dual algorithm that uses both the primal and dual problem to close the duality gap.

## Analyzing Sensitivity with the Dual

Another thing the dual problem can tell about the primal problem is what happens when small changes are made in the constraints. The change in objective function value based on a marginal change in the right hand side coefficients of the primal problem. These small changes are referred to as the shadow prices. The optimal solution vector of the dual problem gives the shadow prices for the respective constraints at the optimal solution of the primal problem. For the calculus inclined, the gradient of the objective function value with respect to each constraint is equal to the objective function vector of the dual program.

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Duality&oldid=387"

- This page was last modified on 25 April 2017, at 13:54.
- This page has been accessed 177,416 times.

# Duality: Bounding the Primal

From CU Denver Optimization Student Wiki

Constructing the dual program is as simple as taking the negative constraint matrix and switching the negative right hand side values for the negative objective function. To gain an understanding of why that is, first consider the same problem constructed for the economic interpretation:

$$\begin{array}{llllll} \text{max} & 11x_1 & + & 5x_2 & + & 5x_3 \\ \text{s.t.} & 2x_1 & + & 1x_2 & + & 1x_3 & \leq & 400 \\ & 2x_1 & + & 2x_2 & + & 1x_3 & \leq & 500 \\ & 10x_1 & + & 5x_2 & + & 6x_3 & \leq & 2,400 \\ & x_1 & , & x_2 & , & x_3 & \geq & 0 \end{array}$$

First, find any feasible point in the primal problem given. For example  $(0, 200, 100)$  is a feasible point for the program. The objective function value for this point is

$$11(0) + 5(200) + 5(100) = 1,500$$

Because this point is feasible, the optimal objective function value is either at this point or at a greater value. Therefore, any feasible point gives a lower bound on what the objective function value can be. This function value can be improved with another feasible solution  $(0, 400, 0)$ . This point gives the objective function value of 20,000

Now that there is a lower bound on the optimal solution, the next step is to find an upper bound. This can be done by adding the constraints together as such:

$$x_1(2 + 2 + 10) + x_2(1 + 2 + 5) + x_3(1 + 1 + 6) \leq (400 + 500 + 2,400)$$

$$\text{which is simplified to } 14x_1 + 8x_2 + 8x_3 \leq 3,300$$

Since each variable is greater than or equal to 0, we know

$$11x_1 + 5x_2 + 5x_3 \leq 14x_1 + 8x_2 + 8x_3 \leq 3300.$$

This implies that 3,300 is an upper bound on the objective function value. the difference between these two values is called the **duality gap**, and the goal of optimization is to close this gap as much as possible. Since linear programs are always convex, the duality gap is 0. The only time there will be non-zero gap is if the program is concave.

Back to the original problem, the upper bound can be improved by adding the constraints together in different ways to get a lower upper bound. For example, adding 3 of the first constraint to 3 of the second constraint gives

$$3(2x_1 + 1x_2 + 1x_3 \leq 400) + 3(2x_1 + 2x_2 + 1x_3 \leq 500) \equiv 12x_1 + 9x_2 + 6x_3 \leq 2,700$$

which is a lower upper bound than the previous one. The goal is to find the infimum of the upper bounds, which can be done by assigning variables to each of the constraints

ose variables to find an optimal solution. Using the variable  $y$ , the constraints become:

$$\begin{array}{lcl} y_1 & (2x_1 + 1x_2 + 1x_3 & \leq 400) \\ y_2 & (2x_1 + 2x_2 + 1x_3 & \leq 500) \\ y_3 & (10x_1 + 5x_2 + 6x_3 & \leq 2,400) \end{array}$$

which can be re-written as

$$\begin{array}{lclclcl} 2x_1y_1 & + & x_2y_1 & + & x_3y_1 & \leq & 400y_1 \\ 2x_1y_2 & + & 2x_2y_2 & + & x_3y_2 & \leq & 500y_2 \\ 10x_1y_3 & + & 5x_2y_3 & + & 6x_3y_3 & \leq & 2400y_3 \end{array}$$

and then reconfigured without the inequalities to be

$$\begin{array}{l} x_1(2y_1 + 2y_2 + 10y_3) \\ x_2(y_1 + 2y_2 + 5y_3) \\ x_3(y_1 + y_2 + 6y_3) \end{array}$$

It is assumed that each  $x$  variable's coefficient is at least as large as its objective function coefficient, so

$$\begin{array}{lclclcl} 2y_1 & + & 2y_2 & + & 10y_3 & \geq & 11 \\ y_1 & + & 2y_2 & + & 5y_3 & \geq & 5 \\ y_1 & + & y_2 & + & 6y_3 & \geq & 5 \end{array}$$

Last bring back the inequalities and remember that each of the variables must be less than their corresponding coefficients, which implies that is the upper bound. In order to find the least upper bound, it is best to minimize this bound to obtain the program

$$\begin{array}{lcl} Min & 400y_1 & + \quad 500y_2 & + \quad 2400y_3 \\ S.T. & 2y_1 & + \quad 2y_2 & + \quad 10y_3 & \geq & 11 \\ & 1y_1 & + \quad 2y_2 & + \quad 5y_3 & \geq & 5 \\ & 1y_1 & + \quad 1y_2 & + \quad 6y_3 & \geq & 5 \\ & y_1 & , \quad y_2 & , \quad y_3 & \geq & 0 \end{array}$$

This program is the dual of the original optimization problem. The reader is challenged to follow this logic on the dual program and see if they obtain the primal.

[1] [2] [3]



# References

1. ↑ Vanderbei, Robert J. Linear programming. Springer US, 2014. International Series in Operations Research & Management Science 196.
2. ↑ Rardin, Ronald L. Indtroduction to Optimization in Operations Research. 2nd ed., Pearson, 2016
3. ↑ Uhan, Nelson “Lesson 34. An Economic Interpretation of LP Duality” Spring 2013, Lecture  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Duality:\\_Bounding\\_the\\_Primal&oldid=245](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Duality:_Bounding_the_Primal&oldid=245)"

- This page was last modified on 21 March 2017, at 12:33.
- This page has been accessed 6,550 times.

# Duality: Economic Example

From CU Denver Optimization Student Wiki

The following is an example of an economic interpretation of a dual program.

Klein's Bottle Company makes 3 different types of bottles. They make a growler that takes 2 kg of glass, 2 sections of label paper, and 10 minutes to make. Klein also makes a short, fat bottle that takes 1 kg of glass, 2 sections of label paper, and 5 minutes to make. Lastly, he makes a tall, thin bottle that takes 1 kg of glass, 1 section of label paper, and 6 minutes to make. He sells the growler for \$11 and each of the smaller bottles for \$5. Klein stocks up on supplies weekly, and therefore has a limited number of supplies to make bottles with each week. His stock at the beginning of each week contains 400 kg of glass and 500 sections of label paper. He can only spend up to 40 hours per week making the bottles, as he has to spend much of the rest of his time selling the product, balancing the books, and performing other various administrative tasks for his business. With this information, he hires a savvy operations researcher and she creates the following LP:

$$\begin{array}{ll} \textit{Max} & 11x_1 + 5x_2 + 5x_3 \\ \textit{S.T.} & 2x_1 + 1x_2 + 1x_3 \leq 400 \\ & 2x_1 + 2x_2 + 1x_3 \leq 500 \\ & 10x_1 + 5x_2 + 6x_3 \leq 2400 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

Klein will be out of town for a week at an important bottle making conference, but does not want to disrupt his reputation with the glass and label paper providers by stopping shipment for a week. He knows that he can sell the resources he has, but does not want to sell them unless he can recover at least what they would be worth in the finished bottles. What he wants is to make the same relative profit as he would if he stayed in business for the week he is taking off. What he wants is to make \$11 for every 2 kg of glass, 2 sections of label paper, and 10 minutes that he loses. He tells this to the operations researcher and she writes his request as:

$$2y_1 + 2y_2 + 10y_3 \geq 11$$

She then generalizes to the other two bottles to generate the set of constraints:

$$\begin{array}{ll} 2y_1 + 2y_2 + 10y_3 & \geq 11 \\ 1y_1 + 2y_2 + 5y_3 & \geq 5 \\ 1y_1 + 1y_2 + 6y_3 & \geq 5 \end{array}$$

Lastly, Klein wants to minimize the effect to his business that selling off resources would have, so the operations researcher finishes her program by adding in an objective function:

$$\begin{array}{ll}
 \text{Min } 400y_1 + 500y_2 + 2400y_3 & \\
 \text{S.T. } 2y_1 + 2y_2 + 10y_3 & \geq 11 \\
 \quad 1y_1 + 2y_2 + 5y_3 & \geq 5 \\
 \quad 1y_1 + 1y_2 + 6y_3 & \geq 5 \\
 \quad y_1, y_2, y_3 & \geq 0
 \end{array}$$

This is the dual problem as a resource allocation model to complement the primal problem of maximizing profit.

[1] [2] [3]

## References

1. ↑ Rardin, Ronald L. Indtroduction to Optimization in Operations Research. 2nd ed., Pearson, 2016
  2. ↑ Vanderbei, Robert J. Linear programming. Springer US, 2014. International Series in Operations Research & Management Science 196.
  3. ↑ Uhan, Nelson “Lesson 34. An Economic Interpretation of LP Duality” Spring 2013, Lecture
- Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Duality:\\_Economic\\_Example&oldid=207](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Duality:_Economic_Example&oldid=207)"

- This page was last modified on 14 March 2017, at 11:15.
- This page has been accessed 4,001 times.

# Dynamic Programming

From CU Denver Optimization Student Wiki

Dynamic programming can be used to solve many optimization problems. In particular, this page was inspired by how to use dynamic programming to solve a subproblem in column generation. This subproblem is classified as a knapsack problem; column generation seeks to find a variable to add to a master problem, and this is done by using the dual problem to generate a column corresponding to a variable with negative reduced cost. The reduced costs is parallel to the value of an item, and the dual variables are the items. We will, however, be exploring dynamic programming in the more general context of the classical knapsack problem.

## Contents

- 1 The Knapsack Problem
- 2 Dynamic Programming
- 3 An Example
- 4 References

## The Knapsack Problem

The classical knapsack problem gives a set of items that each have a given value, and finds the optimal combination of items to yield the maximum value the knapsack can hold. Let  $I = \{i_1, i_2, \dots, i_n\}$  be the set of items, each with weight  $w_k$ . Let  $v_k$  be the value associated with  $i_k$  and the total capacity of the knapsack be  $C$ . Lastly, define integer valued variables  $x_k$  which will determine how many of each item  $i_k$  we will put in the knapsack. The linear program associated to this problem can now be given as follows.

$$\begin{aligned} \max \quad & \sum_{i=1}^n v_k \cdot x_k \\ \text{s.t.} \quad & \sum_{i=1}^n w_k \cdot x_k \leq C \\ & x_k \geq 0 \end{aligned}$$

This optimization problem (finding the maximum value) is NP-hard, so cannot be solved in polynomial time. The decision problem, however, is NP-complete, meaning if we can find a solution to the decision problem then we can calculate the maximum optimal value in polynomial time.<sup>[1]</sup> Clearly, the decision problem is more difficult to solve. But, it has the type of structure that can be broken into a iterative sequence of problems that are easy to solve. This is the idea behind dynamic programming: use an iterative process to solve the decision problem and use that solution to calculate the solution to the optimization problem.

# Dynamic Programming

The goal of dynamic programming is to take a complex problem and break it into a sequence of simpler problems. To do this, we can consider the following questions.

1. Can we view the choice of a feasible solution as a sequence of decisions occurring in stages? This will allow us to say the total value is the sum of the values of individual decisions.
2. How can we define the current state of the problem as a summary of all past decisions?
3. What are possible ways can we transition from one state to another?
4. How can we write a recursive formula for the objective function so we can deduce the optimal value from the previous states?

Let us attempt to answer these questions in the context of the knapsack problem. First, we try to find a way to view our item choices as a series of decisions. For this problem, it is fairly clear that we will choose one item at time. Then, the total value of the knapsack will be the sum of the items chosen at each decision stage.

At each stage we make one decision and the state of the problem at that stage informs the decision. We have no other information: the state of previous stages is not known at the current stage. In our problem, in order to make a decision on the number of an item to include, we need to know the remaining capacity,  $c$ , of the knapsack and this will be our state. The state is "independent" of the previous states in way that at each stage we don't need to know the capacity at previous stages, only how much we have left currently. To move to the next stage, we simply have to find the greatest total value of the items in the current and remaining stages. This can be described by the following iterative relation. <sup>[2]</sup>

$$V_k(c) = \max_{0 \leq x_k \leq \lfloor \frac{c}{w_k} \rfloor} v_k x_k + V_{k+1}(c - w_k x_k)$$

In order to implement this on the computer we must implement the iteration recursively. In other words, we only want to make decisions based on what we already know instead of what the following decisions will be. Some pseudocode for this process is given below. <sup>[3]</sup>

```
# VALUES (stored in list v), WEIGHTS (stored in list w)
# No. of distinct items (N), Knapsack capacity (C)

for j from 0 to C:                                     # no items are chosen in stage 0
    V[0, j] := 0

for i from 1 to N:
    for j from 0 to C:
        if
            w[i] > j then m[i, j] := m[i-1, j]          # if not enough capacity, move on
        else:
            m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] + v[i])  # recursive implementation of iterative process
```

## An Example

Suppose we have  $N = 3$  types of items  $i_1, i_2, i_3$  which have respective weights  $w_1 = 6, w_2 = 4, w_3 = 9$  and values  $v_1 = 3, v_2 = 7, v_3 = 4$ . Further, our knapsack has the capacity  $C = 12$ . Below are the computations for each state at each stage.

$\mathbf{c}$	$V_3(c)$	$x_3^*$
-	0	-
1	0	-
2	0	-
3	0	-
4	0	-
5	0	-
6	3	1
7	3	1
8	3	1
9	3	1
10	3	1
11	3	1
12	6	2

$\mathbf{c}$	$x_2 = 0$	$x_2 = 1$	$x_2 = 2$	$x_2 = 3$	$V_2(c)$	$x_2^*$
0	0	-	-	-	0	0
1	0	-	-	-	0	0
2	0	-	-	-	0	0
3	0	-	-	-	0	0
4	0	$7 + V_3(0)$	-	-	7	1
5	0	$7 + V_3(1)$	-	-	7	1
6	$V_3(6)$	$7 + V_3(2)$	-	-	7	1
7	$V_3(7)$	$7 + V_3(3)$	-	-	7	1
8	$V_3(8)$	$7 + V_3(4)$	$14 + V_3(0)$	-	14	2
9	$V_3(9)$	$7 + V_3(5)$	$14 + V_3(1)$	-	14	2
10	$V_3(10)$	$7 + V_3(6)$	$14 + V_3(2)$	-	14	2
11	$V_3(11)$	$7 + V_3(7)$	$14 + V_3(3)$	-	14	2
12	$V_3(12)$	$7 + V_3(8)$	$14 + V_3(4)$	21	21	3

<b>c</b>	$x_1 = 0$	$x_1 = 1$	$V_1(c)$
9	$0 + V_2(9)$	$4 + V_2(0)$	7
10	$0 + V_2(10)$	$4 + V_2(1)$	14
11	$0 + V_2(11)$	$4 + V_2(2)$	14
12	$0 + V_2(12)$	$4 + V_2(3)$	21

The value of  $V_1(12) = 21$ , which implies that the optimal value of the knapsack is 21. This happens when  $x_1 = 0$ ,  $x_2 = 3$ , and  $x_3 = 0$ .

# References

- ↑ https://en.wikipedia.org/wiki/Knapsack\_problem#0/1\_knapsack\_problem
- ↑ https://www.utdallas.edu/~scniu/OPRE-6201/documents/DP3-Knapsack.pdf
- ↑ AMPL, R. Fourer, D.M. Gay, and B.W. Kernighan, Duxbury Press, 2002.

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Dynamic\_Programming&oldid=999"

- This page was last modified on 7 December 2017, at 18:59.
- This page has been accessed 4,578 times.

# Elise Reed

From CU Denver Optimization Student Wiki

Hello! I am a second year graduate student at the University of Colorado Denver, pursuing a PhD. I got my undergraduate degree in mathematics from the University of Minnesota in 2015. I then spent a year at Smith College before moving to Colorado. During my undergraduate studies, I participated in many math related activities.

The research I did during my junior and senior year investigated human effects on the climate. We used simple ODEs to describe how humans thousands of years ago contributed to global warming with their primitive and inefficient agriculture. Our results indicated that it is not unreasonable to assume that early human activity did in fact contribute to the global warming we are currently experiencing.

Another research project I was able to work with is called Girls Excel in Math (or GEM). The purpose of this project was to train elementary school teachers on how to teach higher level math concepts (e.g. graph theory, combinatorics). One Saturday a month, female teachers and their female students all met on the U of MN campus. During the morning, the teachers attended a training session on the lesson they would be teaching in the afternoon. During this time, undergraduate students (such as myself) would spend time with the elementary students doing basic and fun math activities. The teachers would then return to the room and teach the lesson they were just trained on, with undergraduate students available for assistance. I much enjoyed working with this project and strongly believe we should be trying to train more school-aged teachers on interesting math.

I am most proud of being a founding member of the undergraduate Women in Math club at the U of MN. This group is still active and providing support to undergraduate women who have an interest in mathematics.

My contribution to this wiki:

Dynamic Programming

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Elise\\_Reed&oldid=717](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Elise_Reed&oldid=717)"

Category: Contributors

- 
- This page was last modified on 5 December 2017, at 12:32.
  - This page has been accessed 10,474 times.



# Em Gibbs

From CU Denver Optimization Student Wiki

## Contents

- 1 Contact
- 2 About
  - 2.1 Education
  - 2.2 Programming Languages
- 3 Projects

## Contact

Email (<mailto:emma.gibbs@ucdenver.edu>)

## About

### Education

Pursuing a B.S. in Mathematics and M.S. in Applied Mathematics from CU Denver.

### Programming Languages

- C++
- Java
- Python
- R
- AMPL

## Projects

For the Spring 2022 Network Flows class, I worked with Evan Shapiro and Michael Schmidt on the project Finding Optimal Shared Streets in Denver. In this project, we worked to identify streets in Denver as potential candidates for conversion to Shared or Open Streets, using equitable street selection criteria, and by identifying streets with low traffic flow impact.

- This page was last modified on 9 May 2022, at 13:41.
- This page has been accessed 342 times.

# Embedding AMPL In C

From CU Denver Optimization Student Wiki

## Abstract

Many times when we are optimizing, we may not want to leave the comfort of our programming language of choice or we may not have access to GUI for the AMPL IDE. Whatever the reasoning may be, we demonstrate how to run an AMPL program from within C using Sudoku as a motivating example.

1. Files are already created and you want to run them inside an existing C program
2. You need to create the files from scratch

The first use case requires us to only create a "run" file of AMPL commands and execute this file.

The second use case requires us to generate the AMPL model and data files from scratch, which will vary depending on the problem being solved, but is really just printing strings to files and then repeating the steps in the first use case.

We show a general form that could be abstracted to fit more types of problems as well as mention how one could add extra constraints to solve related forms of Sudoku

## Helpful Project Links

GitHub (<https://github.com/jprhyne/AMPLInsideC>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Embedding\\_AMPL\\_In\\_C&oldid=4293](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Embedding_AMPL_In_C&oldid=4293)"

- 
- This page was last modified on 7 April 2023, at 16:43.
  - This page has been accessed 41 times.

# Emissions and Equality: Colorado Car Share Optimization

From CU Denver Optimization Student Wiki

## Contents

- 1 Abstract
- 2 Data
- 3 Optimization Model
- 4 Results
- 5 Presentation
- 6 References

## Abstract

Car sharing programs, like the nonprofit, federally funded organization Colorado Car Share (CCS), offer an innovative solution to lower the ownership costs of personal vehicles while concurrently contributing to emissions reduction through short term car rentals. To expand these programs, an easier way to find the ideal locations for new vehicles is needed. As the funding for non-profit organizations can come with stipulations on how it can be spent, any model created needs flexibility to allow for them to focus on the target demographic(s) of each grant they receive. We created a linear model that uses US Census tracts to divide the city of Denver into regions. Using population density, and restrictions chosen by the user, it identifies how many vehicles should be added to each tract. This model can be weighted by inputting a variety of factors to locate where the maximum number of people in a particular group may live. We used data from the Denver Open Data Catalog cross-referenced with the current vehicle locations listed on Colorado Car Share's public website. We focused on adaptability given the broad range of stipulations which can be placed on the grant money they receive so, unsurprisingly, we found that the answer depends on the inputs chosen by the user. Our policy recommendations after seeing our results would be to have the city work with CCS to find locations within the identified tracts which allow federal grant money to reduce the cost to build them as well as increase funding/ease of building charging capable carshare spots for new electric vehicles around Denver. By identifying regions that best fit the funding criteria, it allows the city to expend less resources identifying possible locations for charging capable carshare spots and can possibly help to cover the cost of their construction as well.

# Data

To conduct our project, we needed two key pieces of data, census tract basic information of Denver, including the tract name and common indicators of each track, we also needed the information of which tract already has a car from Colorado CarShare. The tract information was easy to find and accessible for the city of Denver<sup>[1]</sup>. The CarShare data on the other hand needed to be manually collected from their website<sup>[2]</sup>. The census tract data comes from American community survey tracts, from 2016-2020 and included 178 tracts. For our project we eliminated to tracts that were parts of DIA and Montebello because they didn't represent areas where we wanted our CarShare parking spaces.

## Optimization Model

For our model, overall, want we want to optimize is placing car parking spots in areas that are most dense. Therefore, we create sets, parameters, and variables with this in mind. The model looks as follows: First, we define some sets, parameters, and variables that will be used throughout.

- Set:  $R$  the set of all census tracts in Denver under consideration.  $i = 1 \dots n$
- Variable:  $x_i$  The number of cars to add to census tract i,  $\forall i \in R$
- Parameter:  $t_i$ . The number of cars currently in census tract i,  $\forall i \in R$
- Parameter:  $p_i$  The population density of census tract i,  $\forall i \in R$
- Parameter:  $w_i$  Weighting based on model variation for census tract i,  $\forall i \in R$
- Parameter:  $c$  The total number of cars to be added
- Parameter:  $m = \max\left(\frac{t_i}{p_i}\right), \forall i \in R$
- Parameter:  $\ell$  An upper bound on the number of cars added to any one tract.

This gives way for our basic model where we only care about maximizing population density.

Here we are not using any weighting, so  $w_i = 1$  for all tracts in  $R$ . Our objective function is given by:

$$\max \sum_{i \in R} w_i \cdot p_i (t_i + x_i)$$

subject to the constraints:

$$\begin{aligned} \sum_{i \in R} x_i &\leq C, \\ m(t_i + x_i) &\leq p_i, \text{ for all } i \in R, \\ x_i + t_i &\leq \ell \text{ for all } i \in R, \end{aligned}$$

and the variable constraints:

We have four more iterations of the model where we focus on maximizing total density and either, percent of households without access, percent of cost burdened population, and percent of population in poverty. The models are as follows, note all constraints stay the same the only thing that changes is the objective function so, only the objective functions will be shown:

### Optimizing Vehicle Access for Households Currently Without Access

For this model, we would use the percentage of households with no vehicle access in each tract as our weight. In this case, our objective function would be:

$$\max \sum_{i \in R} w_i \cdot p_i(t_i + x_i).$$

### Optimizing Vehicle Access for Cost Burdened Population

For this model, we want to maximize vehicle access for that portion of the population classified as "cost burdened" in each region. We'll denote the proportion of region  $i$  classified as "cost burdened" as  $w_i$ . Here, our objective function is:

$$\max \sum_{i \in R} w_i \cdot p_i(t_i + x_i).$$

**Optimizing Vehicle Access for Population in Poverty** For this model, we want to maximize access for that portion of the population in a given region classified as "cost burdened". In this case, our objective function in this case becomes:

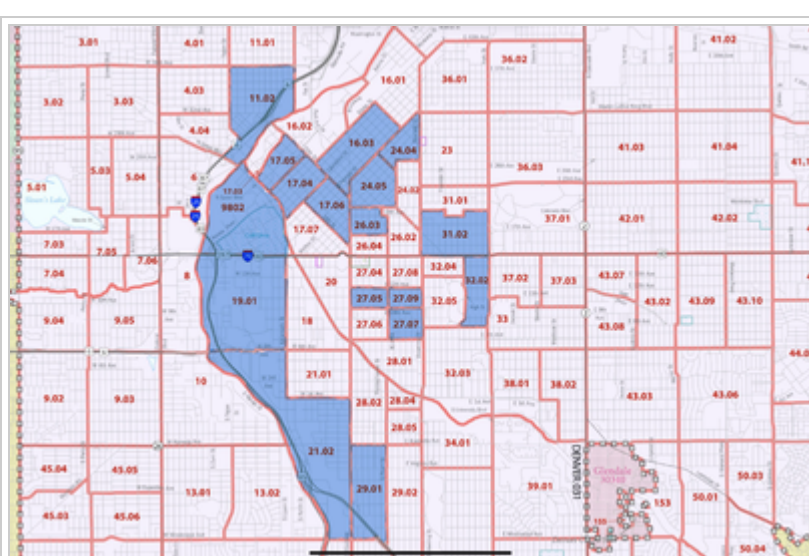
$$\max \sum_{i \in R} w_i \cdot p_i(t_i + x_i).$$

**Optimizing Vehicle Access for Renters** For this model, we want to maximize access for that portion of the population in a given region classified as "renter". Our objective function in this case becomes:

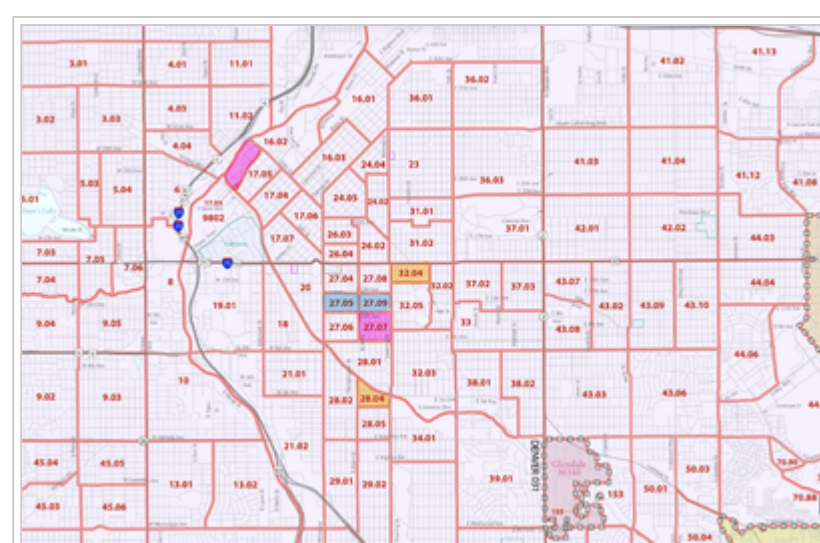
$$\max \sum_{i \in R} w_i \cdot p_i(t_i + x_i).$$

## Results

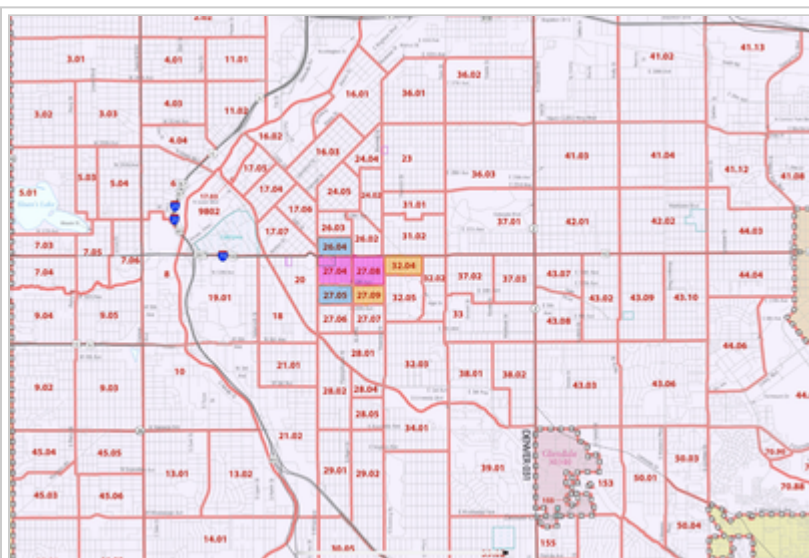
We can see in each of the following images that adjusting our model with the different weights will prioritize different tracts to service different population with vehicle access.



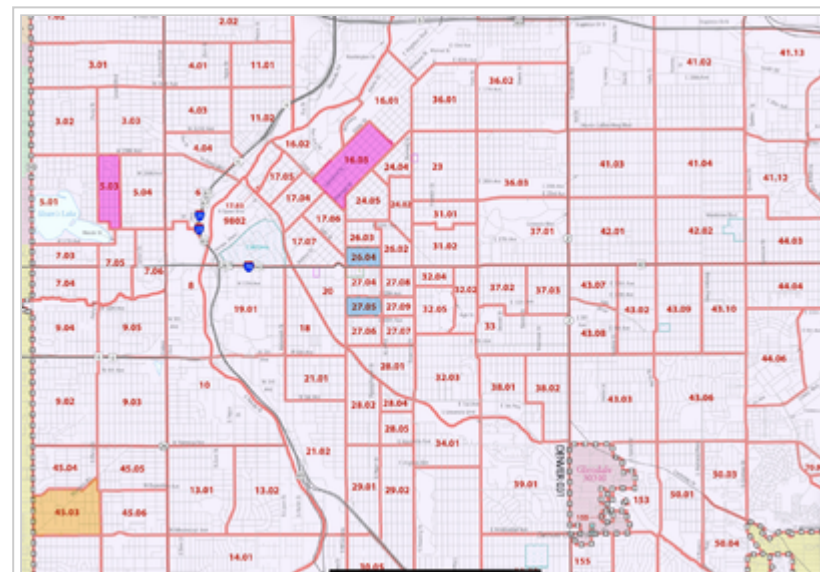
Here we see which tracts are already being served by Colorado CarShare.



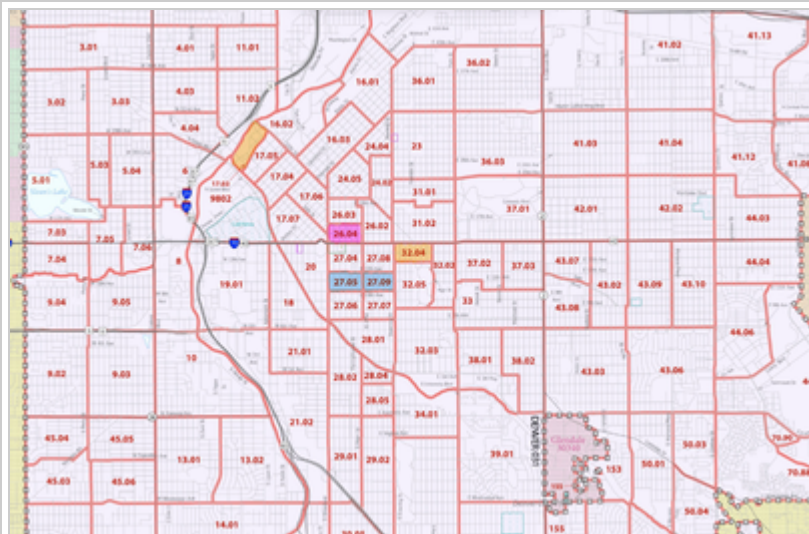
After using our model, with no weighting, focusing only on density of tracts, here is where it is recommended to place cars.



After using our model, with weighting for percent of population without vehicle access these are the tracts where it is recommended to place cars.



After using our model, with weighting for percent of population in poverty these are the tracts where it is recommended to place cars.



After using our model, with weighting for percent of population paying more than 1/3 of their wages to rent these are the tracts where it is recommended to place cars.

Key:  
blue - adding 9 cars to Colorado CarShares fleet  
magenta - adding 18 cars to Colorado CarShares fleet  
orange - adding 27 cars to Colorado CarShares fleet  
Note: in each increase cars we are still adding to the same tracts previously identified.

There are lots of overlap in placements of the cars. We see in each iteration of our model, it is recommended to place new cars in 27.05.

## Presentation

Presentation slides and code ran can be found at the following link. [1] (<https://github.com/pgmath/Emissions-and-Equality>)

## References

- ↑ <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-american-community-survey-tracts-2016-2020>
- ↑ <https://carshare.org/vehicles-locations/>

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Emissions\\_and\\_Equality:\\_Colorado\\_Car\\_Share\\_Optimization&oldid=4578](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Emissions_and_Equality:_Colorado_Car_Share_Optimization&oldid=4578)"





# Equal Flow Problem

From CU Denver Optimization Student Wiki

The Equal Flow Problem is a generalization of the Minimum Cost Flow Problem where there are classes of arcs, valid solutions require that all arcs in a given class have the same flow. This problem has many applications, and many variants. Applications include water resource management<sup>[1]</sup>, federal matching of funds to projects, estimating driver costs for transit operations, and the two duty scheduling problem. Variants, such as "pair only" equal flow problems (each class has two arcs), simple equal flow problems, and integer equal flow problems (an NP complete equal flow problem where the classes must have integer flow) are also all seen.<sup>[1]</sup>

## Contents

- 1 General Linear Programming Formulation
- 2 Variants
  - 2.1 The Simple Equal Flow Problem
    - 2.1.1 The Network Simplex Algorithm
    - 2.1.2 Parametric Algorithms
      - 2.1.2.1 The Parametric Simplex Algorithm
      - 2.1.2.2 The Combinatorial Parametric Algorithm
      - 2.1.2.3 The Binary Search Algorithm
      - 2.1.2.4 The Capacity Scaling Algorithm
  - 2.2 The Pair Equal Flow Problem
  - 2.3 The Integer Equal Flow Problem
- 3 References

## General Linear Programming Formulation

Similar to the LP formulation of a Minimum Cost Problem, the only difference is the arc class restrictions,  $R_m$ .

Given a network  $G = (N, A)$ , and class sets  $R_m$ , minimize  $\sum_{(i,j) \in A} c_{ij} x_{ij}$

with restrictions:

$$\sum_{\{j:(i,j) \in A\}} x_{ij} - \sum_{\{j:(j,i) \in A\}} x_{ji} = b(i), \forall i \in N,$$

$$0 \leq x_{ij} \leq u_{ij}, \forall (i, j) \in A,$$

$$x_{ij} = x_{kl} = x_{R_m}, \forall (i, j), (k, l) \in R_m.$$

# Variants

## The Simple Equal Flow Problem

In the Simple Equal Flow Problem, there is only one arc class( $R$ ) with more than one member arc. This variant originated by the consideration of a water resource management issue in Sardinia, Italy. The seminal problem was quite large (60,000 nodes and 180,000 arcs) and required something more efficient than the standard LP solution. Ahuja et al. presented 5 different algorithms to address it<sup>[1]</sup>. In what follows,  $x_R$  represents the common flow across arcs in  $R$ , and  $u_R$  represents the least upper capacity of the arcs in  $R$

### The Network Simplex Algorithm

A complex Linear Programming method.

### Parametric Algorithms

Parametric Algorithms take the common flow  $x_R$  and parameterize it as an input to an objective function ( $f(x)$ ) measuring the cost of the minimum cost flow where the common flow value is forced. This function, by an LP result, is a piecewise linear convex function, in which breakpoints only on (reduced) rational numbers having a denominator smaller than the size of the arc class  $R$ , and defined on  $[0, u_R]$ . The left and right derivatives ( $f^-$ ,  $f^+$  respectively) can be determined by some small sequence of shortest paths algorithms and  $f$ , they are necessary for certain algorithms. The goal for parametric algorithms is to obtain the minimum of the objective function, and thus an optimal solution.

### The Parametric Simplex Algorithm

The Parametric Simplex Algorithm is a parametric algorithm utilizing LP techniques to find a minimum of the objective function to arrive at an optimal solution.

## The Combinatorial Parametric Algorithm

The Combinatorial Parametric Algorithm is a parametric algorithm utilizing sequences of shortest paths in order to find the minimum of the objective function to arrive at an optimal solution. Unfortunately, though the algorithm makes use of some fairly effective tools, it can get hung up unexpectedly in an exponential number of consecutive degenerate pivots.

## The Binary Search Algorithm

The Binary Search Algorithm is a parametric algorithm utilizing a binary search to find the minimum of the objective function to arrive at an optimal solution. In essence, the algorithm is run by taking the midpoint of the current interval, computing left and right derivatives, and taking the path which leads "downhill" towards the minimum, this works out as the objective function is convex. Also note that the left endpoint has to be checked before starting the algorithm proper, as 0 may be the only solution. If lucky, when the algorithm sees that there is no "downhill" path, the minimum is also found, however due to the nature of representing a real number in a computer, it's possible that the process never converge. However, with some cleverness, the algorithm can be terminated early once the interval size is smaller than  $\frac{1}{|R|^2}$ , as at that point there is a unique break point of the function between the two endpoints (necessary, as the algorithm would have terminated earlier).

The number of interval cuts is  $O(\log |R|U) = O(\log nU)$ , and each interval cut requires the recalculation of the derivatives, which means that a minimum cost flow must be calculated at every step.

## The Capacity Scaling Algorithm

The Capacity Scaling Algorithm is a parametric algorithm utilizing scaling of arc capacities and shortest path calculations to find the minimum of the objective function to arrive at an optimal solution. This algorithm is the best in terms of complexity of the parametric algorithms, yielding a cost of  $O(m(m+n)\log nU)$ .

## The Pair Equal Flow Problem

In the Pair Equal Flow Problem (Ali, Kennington, Shetty [1988]) we have an Equal Flow Problem where each flow class has two arcs.

## The Integer Equal Flow Problem

In the Integer Equal Flow Problem (Meyers, Shulz [2009]), we have an Equal Flow Problem where each flow must have integer quantity. This problem is known to be NP-complete.

## References

- <sup>1.0</sup> <sup>1.1</sup> <sup>1.2</sup> <https://pubsonline.informs.org/doi/abs/10.1287/mnsc.45.10.1440> "Algorithms for the Simple Equal Flow Problem" Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Equal\\_Flow\\_Problem&oldid=1500](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Equal_Flow_Problem&oldid=1500)"

- This page was last modified on 2 May 2018, at 13:25.
- This page has been accessed 1,490 times.

# Equitable Fire Hydrant Placement

From CU Denver Optimization Student Wiki

This project is by Johnathan Rhyne

## Contents

- 1 Abstract
- 2 Overview
  - 2.1 Facility Location
- 3 Implementation
- 4 Results
- 5 Future Work
- 6 Github
- 7 Presentation Materials

## Abstract

We investigate the equitability of fire hydrant placement throughout the Denver Metro area. We do this by assigning each home in select neighborhoods to a fire hydrant, and consider how far away houses can get from their assigned hydrants. We pay special attention to regions of both relatively large and small size, which have room for potential improvement. The former would mean we have few hydrants per building while the latter means the area would be more densely populated. Our aim is to highlight these areas especially in the context of costs for potential locations for new housing, and the safety of the occupants by proposing locations to add new hydrants in order to increase equitabiliy. Furthermore, we aim to see if fire hydrant density is correlated to other socioeconomic factors which could warrant further study around potential effects or other mitigating factors.

## Overview

We looked at several select regions inside the Denver Metro area and their fire hydrant distribution. We determined if we could decrease the overall distance from houses to their closest fire hydrant by adding more.

## Facility Location

In this project, we perform a facility location problem, all this means is that we want to find the optimal placement of some kind of facility (in our case fire hydrants) to minimize some kind of cost (in our case distance). A part of this process will be similar to a clustering problem where we need to assign every

# Implementation

We approach this problem in two steps

1. Isolate regions inside the Denver-Metro area
2. Run a facility location problem to determine where we can benefit from adding fire hydrants

The first step is accomplished through manual inspection of regions on OpenStreet maps. A more data driven approach could look into population densities and potentially factors like socioeconomic status

Our second step is accomplished by proposing some grid points to add hydrants to, and then running a facility location problem on these proposed grid points. Our model looks like

$$\begin{aligned} \min \quad & \sum_{i=1}^n \sum_{j=1}^m c[i, j] \cdot x[i, j] + \sum_{j=1}^m M \cdot y[j] \cdot d[j] \\ \text{subject to} \quad & \sum_{j=1}^m x[i, j] = 1; \text{ for } i = 1, \dots, n \\ & x[i, j] \leq y[j]; \text{ for } i = 1, \dots, n; j = 1, \dots, m \\ & \sum_{i=1}^n x[i, j] \leq N; \text{ for } j = 1, \dots, m \\ & c[i, j] \geq 0 \text{ for } i = 1, \dots, n; j = 1, \dots, m \\ & x[i, j] \in \{0, 1\} \text{ for } i = 1, \dots, n; j = 1, \dots, m \\ & y[j] \in \{0, 1\} \text{ for } j = 1, \dots, m \\ & d[j] \in \{0, 1\} \text{ for } j = 1, \dots, m \end{aligned}$$

Our  $c[i, j]$  values determine the distance from house  $i$  to hydrant  $j$ . These are parameters determined prior to running our program

$x[i, j] = 1$  if and only if house  $i$  is assigned to hydrant  $j$

$y[j] = 1$  if and only if we are adding hydrant  $j$

Our  $d[j]$  parameters are indicators if hydrant  $j$  was already constructed. The purpose of this parameter is to model the fact that we don't want to remove existing fire hydrants.

$M$  is the "cost" of constructing a new hydrant. Since it is difficult to compare the money and time needed to construct the hydrant, we use a surrogate value that represents this cost. The smaller this value is, the more hydrants our model wants to create.

$N$  is the maximum number of houses we want assigned to each hydrant. This models the fact that we want to also not over allocate the hydrants in the event of several connections needed or a large emergency being needed.

## Results

From the regions that we considered, we found that we would not really benefit from adding anymore hydrants with our current model. This can be for several reasons

- We may have only considered neighborhoods with good coverage
- We may have made it too costly to construct a new hydrant ( $M$  too large)
- We may have allowed too many houses for each hydrant ( $N$  too large)
- The current distribution is sufficient.

In order to make more definitive statements, further testing would be needed

## Future Work

This work can be extended to areas outside the Denver Metro area and considering larger regions at a time.

We can also consider more complicated ways to penalize adding a house to a particular hydrant like penalizing based on number of inhabitants (apartment complexes are more costly than a single family home)

## Github

Github Repository (<https://github.com/jprhyne/EquitableFireHydrantPlacement>)

## Presentation Materials

Presentation Files (<https://github.com/jprhyne/EquitableFireHydrantPlacement/tree/main/presentation>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Equitable\\_Fire\\_Hydrant\\_Placement&oldid=4353](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Equitable_Fire_Hydrant_Placement&oldid=4353)"

- This page was last modified on 2 May 2023, at 11:46.
- This page has been accessed 64 times.



# Eric Hu

From CU Denver Optimization Student Wiki

Hello! I am currently a second-year graduate student of Mathematical and Statistical Sciences department at the University of Colorado Denver, my research interest are in the area of scientific computation and application on engineering problem.

During my M.S. study at the National Central University in Taiwan, my research topic primarily discussing the rheological properties, the strain rate dependence viscosity of blood, and its effect in the different complexity of arteries. Typically, the blood expresses the nonlinear visco-property as the strain rate is lower than the threshold value; we adopt the Carreau-Yasuda model to indicate the relationship of blood flow and its viscosity. In order to deal with the irregular geometry of blood vessels, I use the tetrahedral element for the spatial discretization. For the tetrahedral-based finite element scheme, we applied P1-P1 Galerkin-Least Square finite element method for solving Navier-Stokes equation and added two stabilization parameters to solve discretization caused by convection-dominated.

Another research project in my undergraduate was on Mathematical Biology. The topic is related to the regeneration of fish scales using reaction-diffusion model to rebuild the pattern on the fish body. We tried to simulate the gene regulation to achieve the pattern formation in the given domain which represents the fish body, and it also led me to explore the integrated research in scientific computing in my graduate study.

My contribution for the network flow project:

Derivation of Blood Flow as a Network Flow

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Eric\\_Hu&oldid=1147](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Eric_Hu&oldid=1147)"

Category: Contributors

- 
- This page was last modified on 23 April 2018, at 13:44.
  - This page has been accessed 3,698 times.

# Eric Olberding

From CU Denver Optimization Student Wiki

Eric Olberding is a Master's student in Statistics at CU Denver. He plans to graduate Spring 2020. His BS in Mathematics was earned at the University of Tennessee Knoxville.

Follow this link to see my Linear Programming class project: [Linear Regression as Linear Programming](#)

Follow this link to see my Network Flows class project: [Separable Convex Cost Network Flow Problems](#)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Eric\\_Olberding&oldid=2559](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Eric_Olberding&oldid=2559)"

Category: Contributors

- 
- This page was last modified on 26 April 2020, at 18:03.
  - This page has been accessed 1,089 times.

# Evan Shapiro

From CU Denver Optimization Student Wiki

## Contents

- 1 Contact Info
- 2 About Me
- 3 Education
  - 3.1 Programming Languages/Experience
- 4 Optimization Related Project

## Contact Info

- Email (<mailto:Evan.Shapiro@ucdenver.edu>)
- Linkedin (<https://www.linkedin.com/in/evan-shapiro-01953794/>)

## About Me

I am a 3rd year PhD Student in the department of Applied Mathematics and Statistics at CU Denver. I am interested in studying whether uncertainty quantification methods used for dimension reduction in a high-dimensional setting can be applied effectively in a machine learning and AI setting. Personal interests include hiking, cooking, and travel.

## Education

### Programming Languages/Experience

Python Matlab AMPL Fortran C

# Optimization Related Project

During the fall 2022 semester I worked on the D2P project Finding Optimal Shared Streets in Denver for MATH 5490 - Network Flows. I contributed to the policy portion of the project, and I constructed a tutorial using Jupyter Notebooks.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Evan\\_Shapiro&oldid=3862](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Evan_Shapiro&oldid=3862)"

Category: Contributors

---

- This page was last modified on 9 May 2022, at 15:10.
- This page has been accessed 438 times.

# Exploring The Blossom Algorithm

From CU Denver Optimization Student Wiki

This page is for Anne Kreeck's Spring 2025 Graph Theory project 'Exploring the Blossom Algorithm'.

## Project Summary

The blossom algorithm is an extension of the augmenting path algorithm for an  $X, Y$ -bigraph  $G$  that handles the problem of odd cycles using blossoms and stems. The method was presented by Jack Edmonds in 1965 in a famous paper titled "Paths, Trees, and Flowers", a rather appropriate title given that the blossom algorithm uses a forest (made of trees) to find, if they exist, augmenting paths and blossoms.

The approach of the blossom algorithm hinges on Berge's Theorem: "A matching  $M$  in a graph  $G$  is a maximum matching in  $G$  if and only if  $G$  has no  $M$ -augmenting path." Essentially, the algorithm wanders the forest looking for exposed vertices (vertices not in  $M$ ) searching for  $M$  augmenting paths and blossoms until it can find no more and determines the matching  $M$  to be maximum.

A blossom is a cycle in a graph  $G$  consisting of  $2k + 1$  edges of which exactly  $k$  edges belong to the initial matching  $M$ , and where one of the vertices  $v$  of the cycle is such that there exists an alternating path of even length from  $v$  to an exposed vertex  $w$ . This path of even length is referred to as the stem.

Figure 1 shows a blossom (of length 5) and a stem (of length 4) in green, where dotted lines are in the matching  $M$  and solid lines are not. The blossom algorithm contracts these blossoms to a single vertex resulting in a graph  $G'$  that has no odd cycles, allowing us to continue finding augmenting paths without needing to check every edge of the blossoms as each blossom already has a maximum matching.

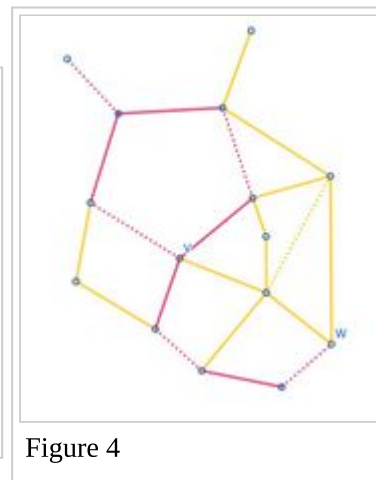
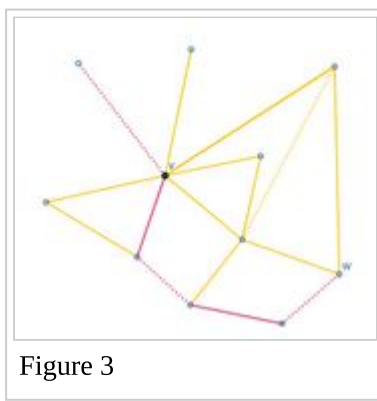
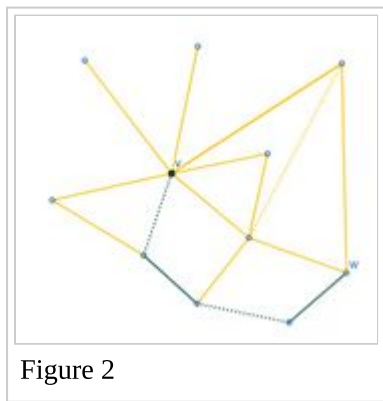
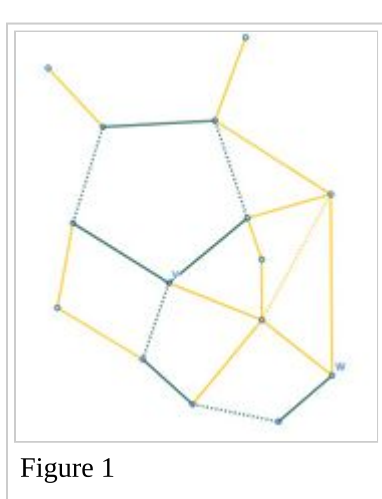


Figure 2 shows the graph after the blossom has been contracted, while figures 3 and 4 show the  $M$ -augmenting path and the graph after we have expanded our blossom, with  $M$  now having 1 additional edge where the edges of the blossom in  $M$  have been adjusted to line up with the augmentation.

The blossom algorithm runs in  $O(|E||V|^2)$  time and was the first proof of a polynomial time algorithm for finding a maximum matching. Edmonds followed this work later in 1965 with an approach for minimum-weight matchings using a linear programming polyhedral description of the matching polytope.

## GitHub Repository

In the GitHub (<https://github.com/A-Kreeck/Exploring-The-Blossom-Algorithm>) repository for this project you will find the slide deck used for the end of semester presentation.

## Bibliography

Jack Edmonds. "Maximum matching and a polyhedron with 0,1-vertices". In: *Journal of research of the National Bureau of Standards* B 69.125-130 (1965), pp. 55-56

Jack Edmonds. "Paths, Trees, and Flowers". In: *Canadian Journal of Mathematics* 17 (1965), pp. 449-467.

Giacomo Zambell, Michelle Conforti, Gérard Cornuejols. *Integer Programming*. Graduate Texts in Mathematics'. Springer International Publishing, 2014. ISBN: 9783319110073'.

Douglas B. West. *Combinatorial Mathematics*. Cambridge University Press, 2021. ISBN: 9781107058583.

*Blossom Algorithm*. URL: [https://en.wikipedia.org/wiki/Blossom\\_algorithm](https://en.wikipedia.org/wiki/Blossom_algorithm). (accessed: 04.30.2025)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Exploring\\_The\\_Blossom\\_Algorithm&oldid=5026](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Exploring_The_Blossom_Algorithm&oldid=5026)"

- 
- This page was last modified on 7 May 2025, at 13:00.
  - This page has been accessed 32 times.

# Exploring the Network Simplex Method

From CU Denver Optimization Student Wiki

Angela Morrison Summer 2021 readings course work.

## Contents

- 1 Review of the Simplex Method
  - 1.1 What is the Simplex Method?
  - 1.2 Generalization of the Simplex Method
  - 1.3 Simplex Method Example
- 2 Networks
  - 2.1 What is a Network?
  - 2.2 Spanning Trees in the Network Simplex Method
  - 2.3 Network Simplex Method
  - 2.4 Network Simplex Example
  - 2.5 References

## Review of the Simplex Method

### What is the Simplex Method?

The Simplex method is an algorithm used to find the optimal solution for a Linear Program (LP). Starting at an initial feasible solution, the algorithm determines a direction of improvement for the objective function. From there, the algorithm performs an edge walk between vertices of the polyhedra in the direction which improves the objective function. Each "step" in the Simplex method is called a pivot. These pivots are how the algorithm chooses which direction to traverse during its edge walk.

### Generalization of the Simplex Method

We will start by looking algebra behind the tableau notation for the Simplex method. Suppose we are given an LP in standard form:

$$\begin{array}{lll} \max & c^T x & = Z \\ \text{s. t.} & Ax & = b \\ & x & \geq 0 \end{array}$$

We can separate all of the information in the LP based on the basic and non-basic variables in the following fashion:

$$\begin{aligned} A &= (A_B, A_N) \\ x &= (x_B^T, x_N^T)^T \\ c &= (c_B^T, c_N^T)^T \end{aligned}$$

where  $B$  is the set of basic variables and  $N$  is the set of non-basic variables. With this in mind, we can rewrite our LP as:

$$\begin{aligned} \max \quad & c_B^T x_B + c_N^T x_N = Z \\ \text{s. t.} \quad & A_B x_B + A_N x_N = b \\ & x_B, x_N \geq 0. \end{aligned}$$

Because of the way we have rewritten the LP, we can rewrite the constraints in the LP in terms of the constants and the non basic variables by solving for  $x_B$  to obtain

$$x_B = A_B^{-1}b - A_B^{-1}A_N x_N.$$

From here we can make some substitutions in the objective function of the LP:

$$\begin{aligned} \max \quad & c_B^T (A_B^{-1}b - A_B^{-1}A_N x_N) + c_N^T x_N = Z \\ \text{s. t.} \quad & A_B^{-1}b - A_B^{-1}A_N x_N = x_B \\ & x_B, x_N \geq 0, \end{aligned}$$

which simplify to

$$\begin{aligned} \max \quad & c_B^T A_B^{-1}b - (c_B^T A_B^{-1}A_N - c_N^T)x_N = Z \\ \text{s. t.} \quad & A_B^{-1}b - A_B^{-1}A_N x_N = x_B \\ & x_B, x_N \geq 0. \end{aligned}$$

From this, we can see where some of the terms in our dictionary come from. In particular

$$\begin{aligned} \overline{Z} &= c_B^T A_B^{-1}b \\ (\overline{c_j})_{j \in N} &= c_B^T A_B^{-1}A_N - c_N^T \\ (\overline{a_{ij}})_{i \in B, j \in N} &= A_B^{-1}A_N \\ (\overline{b_i})_{i \in B} &= A_B^{-1}b \end{aligned}$$



where  $\overline{Z}$  is the objective function value at a particular vertex, and  $\overline{c_j}$ ,  $\overline{a_{ij}}$ , and  $\overline{b_i}$  are the updated cost, coefficients, and bounds for the vertex.

We can represent this LP in a simplex tableau as follows:

$$\begin{bmatrix} 0 & -(\overline{a_{ij}})_{i \in B, j \in N} & -I & (\overline{b_i})_{i \in B} \\ -1 & -(\overline{c_j})_{j \in N} & 0 & \overline{Z} \end{bmatrix}$$

With this tableau, we can use the largest coefficient rule and the minimum ratio test to choose the pivot location within the tableau, call it  $\alpha$ .

Because we know the position of  $\alpha$ , we can perform a matrix multiplication with a Gaussian-Jordan pivot matrix, in order to perform the pivot itself <sup>[1]</sup>. A Gaussian-Jordan pivot matrix is defined as:

$$G = \begin{bmatrix} I_{s \times s} & -\alpha^{-1}a & 0 \\ 0 & \alpha^{-1} & 0 \\ 0 & -\alpha^{-1}d & I_{t \times t} \end{bmatrix}$$

where  $s$  is the number of rows above the pivot position  $\alpha$ ,  $t$  is the number of rows below the pivot position,  $a$  are the entries in the pivot column above the pivot position, and similarly  $d$  are the pivot column entries below the pivot position.

This Gaussian-Jordan pivot matrix is what we multiply the simplex tableau by in order to perform the row operations which would switch out our new basic and non-basic variables in the tableau. It also updates all other entries in the tableau accordingly. This is the general process behind the iterations of the Simplex method.

## Simplex Method Example

Suppose we are given the following LP <sup>[2]</sup>, and are asked to find the vector  $(x, y)$  which gives the optimal solution. It is important to note that any optimal solution to this LP will occur at a vertex of the feasible region depicted in Figure 1. In order to traverse these vertices and find our optimal solution, we will use the Simplex Algorithm.

$$\begin{array}{llllll} \max & 3x & + & 5y & = & Z \\ \text{s. t.} & 2x & + & 3y & \leq & 10 \\ & x & + & 2y & \leq & 6 \\ & x & & & \leq & 4 \\ & & & y & \leq & 3 \\ & x & , & y & \geq & 0 \end{array}$$

A requirement of the Simplex method is that our LP be in standard form, so we will need to add slack variables to our constraints. The new formulation of the LP is therefore:

$$\begin{array}{rcllclclclcl}
 \max & 3x & + & 5y & & & & = & Z \\
 \text{s. t.} & 2x & + & 3y & + & w_1 & & = & 10 \\
 & x & + & 2y & & & + & w_2 & = & 6 \\
 & x & & & & & & + & w_3 & = & 4 \\
 & & & y & & & & + & w_4 & = & 3 \\
 & x & , & y & , & w_1 & , & w_2 & , & w_3 & , & w_4 & \geq & 0
 \end{array}$$

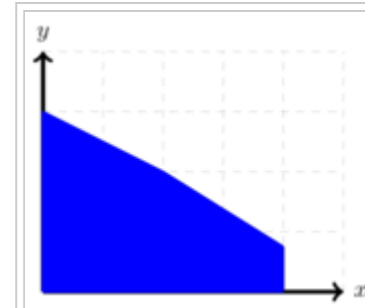


Figure 1: Feasible region of the LP.

With our LP in standard form we can begin finding an initial feasible solution. This gives us a vertex to start at for our edge walk along the feasible region of the polyhedra. For this example, it is rather easy to find an initial solution since the point  $(0, 0)$  works. However, for some LP's it is just as much work to find an initial feasible solution as it is to solve the problem itself.

With our initial feasible solution  $(0, 0)$ , we can write the initial tableau of the Simplex method<sup>[3]</sup> in the following format:

$$\begin{array}{rcllcl}
 Z & = & & 3x & + & 5y \\
 w_1 & = & 10 & - & 2x & - & 3y \\
 w_2 & = & 6 & - & x & - & 2y \\
 w_3 & = & 4 & - & x & & \\
 w_4 & = & 3 & & & - & y
 \end{array}$$

Looking at this initial tableau, we employ the largest coefficient rule in order to determine which variable moves from non-basic to basic, also called the entering variable. The largest coefficient rule states that the entering variable will correspond to the variable which has the largest non-negative coefficient in the objective function of the LP. Based on this, our entering variable would be  $y$ .

To choose which variable will then move from basic to non-basic, also called the leaving variable, we apply the minimum ratio test. The minimum ratio test is way to determine what the step size will be when we move in the direction of our entering variable. To perform the minimum ratio test, we take all the nonzero coefficients of our entering variable in each constraint, divide its corresponding  $b$  value, and choose the smallest ratio. Our example has the following ratios:

$$\begin{array}{l}
 \frac{10}{3} = 3.\overline{33} \\
 \frac{6}{2} = 3 \\
 \frac{3}{1} = 3
 \end{array}$$

Since there is a tie for the minimum ratio test, we could choose either row as the pivot row. However, whenever such a tie happens, the vertex that we would be moving to is degenerate, i.e., there are more active facets than necessary to specify the vertex. In such a situation, it is necessary to prevent the Simplex method from cycling or stalling. One common way to do so would be to instead use Bland's Rule of choosing our entering and leaving variables. Using the largest coefficient rule and minimum ratio test alone

do not guarantee the algorithm will terminate. However, Bland's Rule is guaranteed to terminate assuming that the entering and leaving variables are chosen in the same fashion.

While our example would have terminated successfully had we kept going in this fashion, invoking Bland's Rule adds another level to the example that is worth discussing. For reference, Bland's Rule is that the entering and leaving variable be chosen from their respective sets such that their index is the smallest <sup>[3]</sup>. In short, we look at the sets of all eligible entering and leaving variables, and choose the variable with the smallest index in each set respectively. Since our variables are not written in a notation that is easy to see which index is larger than another, we can think of them as

$$(x, y, w_1, w_2, w_3, w_4) = (x_1, x_2, x_3, x_4, x_5, x_6).$$

Applying Bland's Rule to this example, the entering variable would be  $x$ . In choosing  $x$  as the entering variable, we have the following ratios from the minimum ratio test:

$$\begin{array}{l} \frac{10}{2} = 5 \\ \frac{6}{1} = 6 \\ \frac{4}{1} = 4 \end{array}$$

Based on these ratio values, we see that our leaving variable will be  $w_3$ .

With our new basic and non-basic variables chosen, we use the substitution  $x = 4 - w_3$  to rewrite our Simplex tableau as:

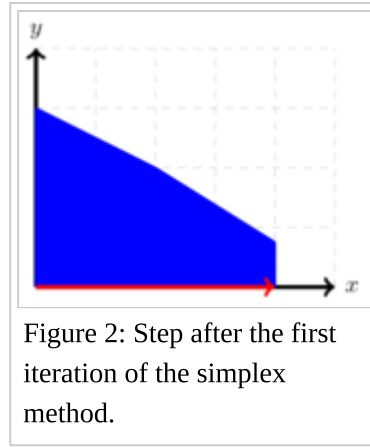
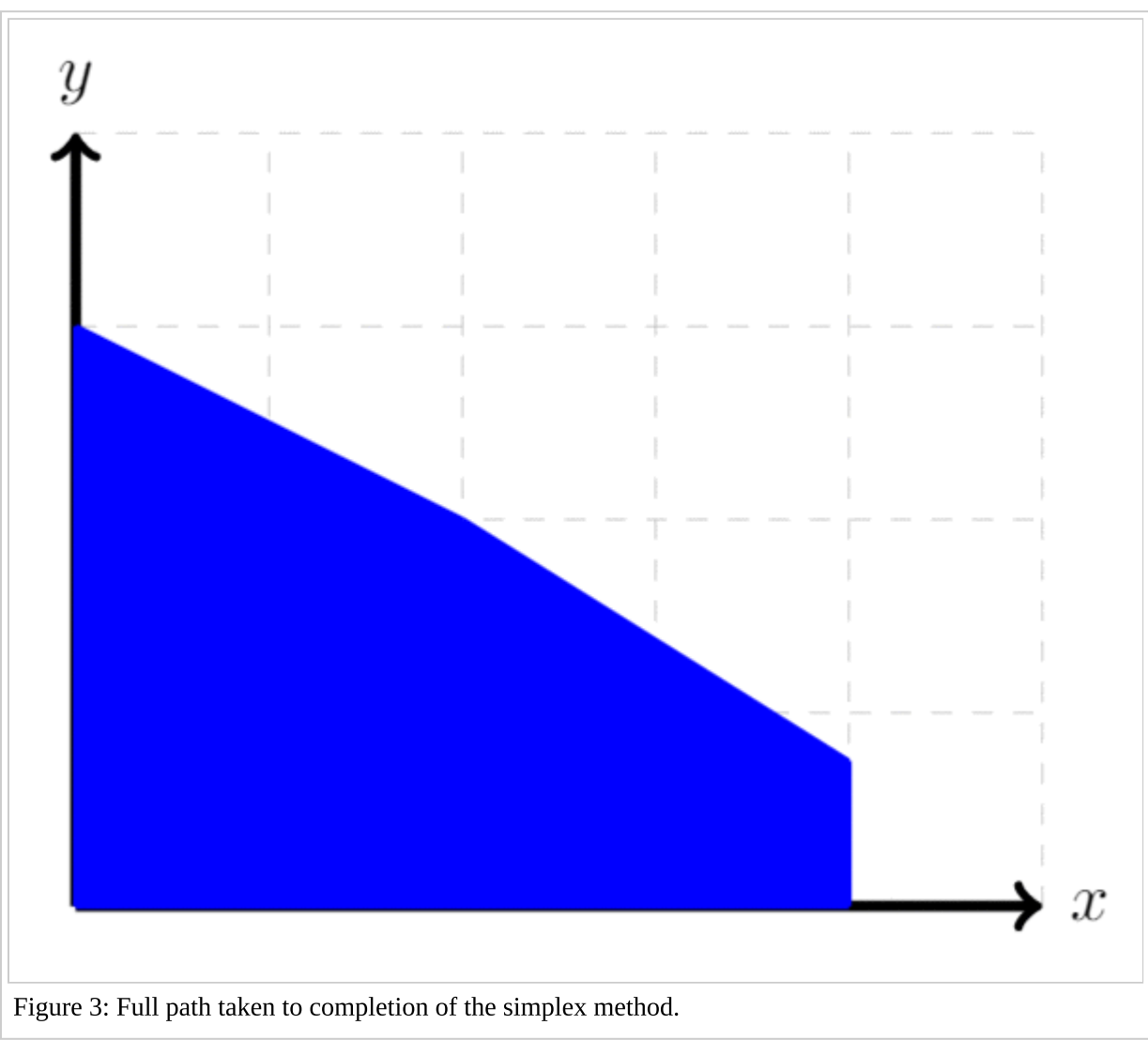
$$\begin{array}{rcccccl} Z & = & 12 & + & 5y & - & 3w_3 \\ w_1 & = & 2 & - & 3y & + & 2w_3 \\ w_2 & = & 2 & - & 2y & + & w_3 \\ x & = & 4 & & & - & w_3 \\ w_4 & = & 3 & - & y & & \end{array}$$

Looking at our feasible region, this pivot of the Simplex method corresponds to the path shown in Figure 2.

Repeating these steps, we arrive at the final Simplex tableau.

$$\begin{array}{rclclcl}
 Z & = & 16 & - & w_1 & - & w_2 \\
 y & = & 2 & + & w_1 & - & 2w_2 \\
 w_3 & = & 2 & + & 2w_1 & - & 3w_2 \\
 x & = & 2 & - & 2w_1 & + & 3w_2 \\
 w_4 & = & 1 & - & w_1 & + & 2w_2
 \end{array}$$

With this final tableau, we can now determine the vector for the optimal solution. The optimal solution is  $(2, 2)$  with an objective value of 16. Our path for the Simplex method is shown in Figure 3.



# Networks

## What is a Network?

A network  $G = (V, E)$  is a directed graph which contains nodes  $V$  and arcs  $E$ . Each arc has an associated lower capacity  $\ell$  and upper capacity  $u$  which sets bounds on how much flow can be sent along it. Each arc also has an associated cost  $c_{ij}$  for carrying flow from node  $i$  to  $j$ . Lastly, each node itself has a supply/demand  $b_i$  for all  $i \in V$ .  $b_i$  is if that node is able to supply flow, 0 if it meant to have no flow at the end of the problem, negative if the node demands flow. The particular network of interest to us is the minimum-cost flow problem which is represented as:

$$\begin{array}{llll} \min & & c^T x & = & Z \\ \text{s. t.} & & Ax & = & b \\ & \ell & \leq & x & \leq & u. \end{array}$$

Here the matrix  $A$  represents a node-arc incidence matrix. It expresses how the arcs connect the nodes within the network where each row of the matrix represents a node and each column an arc. In particular, the matrix will contain a 1 the places where an arc comes out of a particular node and a -1 where it is going into a particular node. Also,  $x$  represents the amount of flow on each arc in  $E$ .

This type of network flow problem is of interest because unlike general LP's, minimum-cost flow problems are efficiently solvable through the so called network simplex method (a special derivation of the simplex method) or min-mean cycle cancelling.

Before jumping into the network simplex method itself and without any loss of generality, we make a few assumptions about our minimum-cost flow problems. The first is that the system is closed in order to allow for feasible flow. The second is that the network is connected. That is to say, all the nodes in the network for the minimum-cost flow problems are have a have path between them.

## Spanning Trees in the Network Simplex Method

In general, the network simplex algorithm rotates through a series of spanning tree solutions until it reaches an optimal one. In order to better understand the iterations of the Network Simplex method, some terms associated with these spanning trees are necessary and are thus defined in this section.

For any feasible solution to our network  $x$ , an arc  $(i, j)$  is a **free** or **basic** arc if  $0 < x_{ij} < u_{ij}$  and a **restricted** or **non-basic** arc if  $x_{ij} = 0$  or  $x_{ij} = u_{ij}$  <sup>[4]</sup>. The feasible solution  $x$  and its associated spanning tree is what is known as a **spanning tree solution** if every nontree arc is a non-basic arc. This type of solution is exploited by the network simplex algorithm in that it only looks for spanning tree solutions. We can also guarantee this type of solution exists by the Spanning Tree Property which states that if the objective function of a minimum-cost flow problem is bounded from below over the feasible region, the problem always has an optimal spanning tree solution <sup>[4]</sup>.

Due to the way spanning tree solutions are defined, we can partition our set of arcs,  $E$ , into three subsets call them  $T$ ,  $L$ , and  $U$  with  $T = \{x_{ij} | x_{ij} \text{ in spanning tree solution} \}$ ,  $L = \{x_{ij} | x_{ij} = 0, x_{ij} \in E \setminus T\}$ , and  $U = \{x_{ij} | x_{ij} = u_{ij}, x_{ij} \in E \setminus T\}$ . A spanning tree structure is **feasible** if it satisfies all the arc's flow bounds, and is **degenerate** if not every arc is a basic arc.

# Network Simplex Method

Now that we have an understanding of the type of solution we are expecting to get from the network simplex method, we can get into the steps of the algorithm itself. These steps are given as follows <sup>[5]</sup>

1. Obtain an initial feasible spanning tree solution for the minimum-cost flow problem by:
  - Establishing a feasible flow by solving a maximum flow problem and convert that solution to a spanning tree solution <sup>[4]</sup>.
  - Northwest-Corner Rule, which takes the first supply node and sends as much as possible on an arc that goes to a demand node, and each subsequent supply node does the same until a new demand node must be satisfied, see <sup>[5]</sup>.
2. Compute initial dual variables, assuming the root node has a value of 0, using

$$y_j - y_i = c_{ij}$$

and the dual slack variables on non-tree arcs using

$$z_{ij} = y_i + c_{ij} - y_j.$$

3. Begin iterations:
  1. Add an arc to spanning tree solution which creates a cycle. The arc that is added must be dual infeasible. That is to say  $z_{ij} < 0$ . In fact, one could even pick the most negative  $z_{ij}$ .
  2. Since this forms a cycle we need to remove an arc in order to form a new spanning tree. The arc we remove must be in the reverse direction of the entering arc and have the smallest flow among all such arcs. One could think of this as being the arc which goes to zero the fastest when increasing flow on the entering arc and adjusting all other arcs in the cycle for this increase in flow.
  3. Update the flows of the primal spanning tree. When the leaving arc is removed, but before adding in the entering arc, the original spanning tree is divided into two disjoint subtrees: one which contains what we have decided is the root node and one which does not. Only the arcs of the new spanning tree that lie in the non-root node subtree need to be updated for this change this flow in the primal spanning tree. This is because only the flow in the cycle created by having the entering and leaving arcs is effected by this change from non-basic to basic arcs. The arcs that meet this criteria are updated as follows:
    - If the arcs are in the same cycling direction as the leaving arc, then their flows are all decreased by the amount of flow that was put on the entering arc.
    - If the arcs are in the opposite direction of the leaving arc, then the flows along those arcs are increased by the amount of flow sent along the entering arc.
4. Update the dual variables:

Dual variables are also updated based on lying in the non-root node subtree or not. Because the dual variables are calculated starting at the root node, the arcs which would be part of the root node subtree would not need to be updated. This means the only dual variables (nodes) that need to be updated are those of the non-root node subtree. This is done as follows:

- If the entering arc connects the root node subtree to the non-root node subtree, increase all nodes on the non-root node subtree by the amount of flow on the leaving arc.
  - Otherwise, decrease all non-root node subtree nodes by the amount of flow on the entering arc.
5. Update dual slack variables:

Again, the only dual slack variables that will be updated are those that connect the root node subtree to the non-root node subtree. These dual slack variables (non-tree arcs) are updated as follows:

- Arcs that are in the same cycle direction as the entering arc as decreased by the amount of flow on the leaving arc
- Arcs that are in the opposite cycle direction as the entering arc are increased by the amount of flow on the leaving arc

With this outline of the network simplex process, let us begin working through a simple example to better understand what this procedure looks like.

## Network Simplex Example

Suppose we are given the following LP <sup>[3]</sup>:

$$\begin{array}{lll} \min & c^T x & = Z \\ \text{s. t.} & Ax & = b \\ & 0 & \leq x \end{array}$$

where

$$x^T = [x_{ac} \quad x_{ad} \quad x_{ae} \quad x_{ba} \quad x_{bc} \quad x_{be} \quad x_{db} \quad x_{de} \quad x_{fa} \quad x_{fb} \quad x_{fc} \quad x_{fg} \quad x_{gb} \quad x_{ge}],$$

$$A = \begin{bmatrix} -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 0 \\ -6 \\ -6 \\ -2 \\ 9 \\ 5 \end{bmatrix},$$

and

$$c^T = [48 \quad 28 \quad 10 \quad 7 \quad 65 \quad 7 \quad 38 \quad 15 \quad 56 \quad 48 \quad 108 \quad 24 \quad 33 \quad 19].$$

The network associated with this LP is shown in Figure 4.

From the outline given in the Network Simplex Method section, the first step in the Network Simplex method is finding a feasible spanning tree. This can be done in a variety of ways, but for this example we will be using the Northwest-corner Rule. In using this rule, we want to form a table of our supply and demand nodes, and allocate as much as we can to the demand node in the first column from the supply node in the first row. We will continue in this fashion until all requirements have been met. For our example, this allocation looks as seen in Table 1.

Table 1

	<i>c</i>	<i>d</i>	<i>e</i>	Supply
<i>f</i>	6	3	0	9
<i>g</i>	0	3	2	5
<b>Demand</b>	6	6	2	

Along with finding an initial feasible spanning tree solution, we also need to pick a root node for our spanning tree. For the sake of this problem, we will choose node *g* as the root node for our spanning tree solution. With this assignment of the root node done, we can compute the values for the dual variables and dual slack variables. The dual variables are computed as:

$$\begin{aligned}
 y_g &= 0 \\
 y_e - y_g &= 19 \Rightarrow y_e = 19 \\
 y_b - y_g &= 33 \Rightarrow y_b = 33 \\
 y_a - y_b &= 7 \Rightarrow y_a = 40 \\
 y_d - y_a &= 28 \Rightarrow y_d = 68 \\
 y_a - y_f &= 56 \Rightarrow y_f = -16 \\
 y_c - y_f &= 108 \Rightarrow y_c = 92
 \end{aligned}$$

Similarly the dual slack variables are computed as:

$$\begin{aligned}
 y_c - y_a + z_{ac} &= 48 \Rightarrow z_{ac} = -4 \\
 y_e - y_a + z_{ae} &= 10 \Rightarrow z_{ae} = 31 \\
 y_c - y_b + z_{bc} &= 65 \Rightarrow z_{bc} = 6 \\
 y_e - y_b + z_{be} &= 7 \Rightarrow z_{be} = 21 \\
 y_b - y_d + z_{db} &= 38 \Rightarrow z_{db} = 73 \\
 y_e - y_d + z_{de} &= 15 \Rightarrow z_{de} = 64 \\
 y_b - y_f + z_{fb} &= 48 \Rightarrow z_{fb} = -1 \\
 y_g - y_f + z_{fg} &= 24 \Rightarrow z_{fg} = 8
 \end{aligned}$$

With the distribution of flow along with dual variables and dual slack variable information, we arrive at the initial solution depicted in Figure 5.

The entering arc that is selected based on the dual slack variable values in Figure 5 is  $x_{ac}$ . Having this arc enter creates a cycle between *a*, *c*, and *f*. Our leaving arc must then be an arc that is going in the opposite direction of  $x_{ac}$  and also the lowest flow of all such arcs. Thankfully, there is only one such arc to choose from, so our leaving arc is  $x_{fc}$ . Because of the entering and leaving arcs, the non-root node subtree is just the node *c*, and the remaining nodes are then part of the root node subtree. This means that only

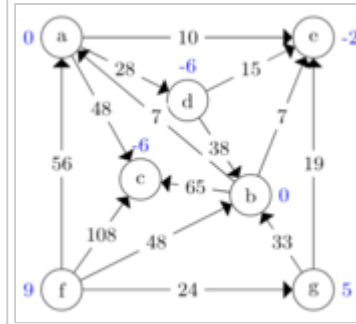


Figure 4: The blue next to the nodes represent the supply/demand values, while the black on the arcs represent the cost for each arc.

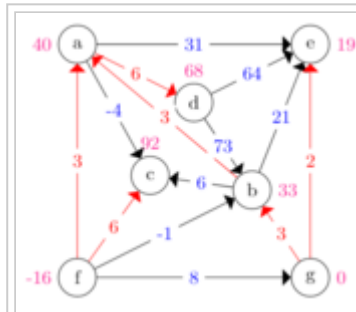


Figure 5: The red arcs are part of the spanning tree. The blue values on the non-tree arcs are the dual slack variable values. The pink values next to the nodes represent the dual variable values.



We update the spanning tree values as described in the network simplex algorithm, and arrive at the new spanning tree solution in Figure 6.

Once again, we need to choose an entering and leaving arc. There is only one remaining dual slack variable which is negative, so the entering arc will be  $x_{fb}$ , and the leaving arc will then be  $x_{fa}$ . Note that once again the non-root node subtree is a single node,  $f$ . Therefore the dual variable associated with node  $f$  along with all arcs connected to node  $f$  will need to be updated.

Once everything is updated, the spanning tree for iteration looks as shown in Figure 7.

There are no longer any negative reduced costs to choose as an entering arc, so there is no need for another iteration of the Network Simplex method. The solution after the second iteration, Figure 7, is the final answer for our problem.

## References

- ↑ J. V. Burke. Lecture notes in linear optimization. <https://sites.math.washington.edu/~burke/crs/407/notes/section2.pdf>, Fall 2020.
- ↑ J. B. Orlin. Lecture 17: The network simplex algorithm. In *Network Optimization—MIT Course No. 15.082J*. <https://ocw.mit.edu/courses/sloan-school-of-management/15-082j-network-optimization-fall-2010/index.htm>, Cambridge MA, 2010. MIT Open-CourseWare.
- ↑ <sup>3.0</sup> <sup>3.1</sup> <sup>3.2</sup> R. J. Vanderbei. *Linear Programming Foundations and Extensions*′. Springer International Publishing, 2014.
- ↑ <sup>4.0</sup> <sup>4.1</sup> <sup>4.2</sup> R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: Theory, algorithms, and applications*. Prentice Hall Inc., Englewood Cliffs, NJ, 1993.
- ↑ <sup>5.0</sup> <sup>5.1</sup> S. P. Bradley, A. C. Hax, and T. L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley, 1977.

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Exploring\_the\_Network\_Simplex\_Method&oldid=3608"

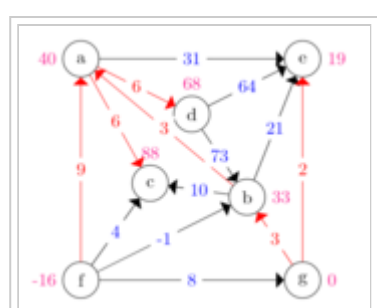


Figure 6:This is the spanning tree solution and updated variables for the first iteration of the Network Simplex method.

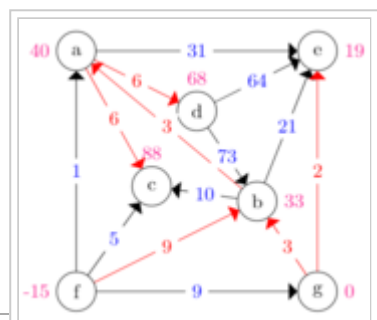


Figure 7:This is the spanning tree solution and updated variables for the second iteration of the Network Simplex method.

# Finding Optimal Shared Streets in Denver

From CU Denver Optimization Student Wiki

This is the page for the project Finding Optimal Shared Streets in Denver, by Michael Schmidt, Evan Shapiro, and Em Gibbs. Relevant data, Python notebooks, and presentation slides can be found in the [#GitHub](#) repository.

## Contents

- 1 Overview
  - 1.1 Background
  - 1.2 Abstract
- 2 Data
  - 2.1 Street Selection Criteria
  - 2.2 Candidate Neighborhoods
  - 2.3 Candidate Streets
- 3 Algorithms
  - 3.1 Assumptions
  - 3.2 Determining Road Capacity
  - 3.3 Max-Flow with Preflow Push
    - 3.3.1 Initialization
    - 3.3.2 Push or Relabel
  - 3.4 All Pairs Impact
- 4 Results and Policy
  - 4.1 Top 3 Candidates
    - 4.1.1 Curtis Street (15th Street – 17th Street)
    - 4.1.2 Marion Street (East 18th Ave – East 19th Ave)
    - 4.1.3 Lafayette Street (East 17th Ave – East 18th Ave)
    - 4.1.4 Marion Street (East 18th Ave – East 19th Ave) & Lafayette Street (East 17th Ave – East 18th Ave)
  - 4.2 All Candidates Impact By Neighborhood
  - 4.3 Further Research
  - 4.4 Generality of Code
- 5 GitHub
- 6 Tutorial
- 7 Resources

# Overview

## Background

During the COVID-19 pandemic, in an effort to encourage people living in cities to go outside while maintaining proper social distancing measures, many cities completely or partially closed city streets to cars, creating Open or Shared Streets, respectively. In its Shared Streets Initiative, Denver closed seven stretches of street when the program launched in April 2020.<sup>[1]</sup> The program was very popular, with the Denver Streets Partnership finding that 71% of voters approved of the program,<sup>[2]</sup> with some streets seeing an increase in pedestrian and bicyclist usage from 351 per day to 1,700 per day.<sup>[3]</sup> While the Shared Streets Initiative officially ended in the week of August 16, 2021, due to the popularity of the program, the City of Denver is considering implementing a permanent version of the Shared Streets Initiative, although the development of this project is still underway.

## Abstract

Demand for pedestrian-friendly business districts and recreational areas in the Denver area has increased since the beginning of the COVID-19 pandemic. A 2020 Denver Streets Partnership survey found respondents increased their walking and biking by 80%. In one area, the Colorado Public Interest Research Group saw a 400% increase in pedestrian traffic. In response to this, the city and county of Denver’s Shared and Open Streets program has established precedence of partially or completely closing streets around Denver to improve the safety of pedestrians and increase areas for recreation. These Open or Shared Streets also provide safer, more accessible walkways for people in wheelchairs and people living with disabilities.

In this project, we intend to design a framework for identifying potential candidates for shared street conversion to promote shared streets and accommodate a growing population. First, we find the maximum traffic flow for both the current street layout and the layout after the road is converted to an open street to understand the impact. Then, using criteria for equitable street selection, we can compare the effects of potential conversions across several candidate roadways. With these results, policymakers can determine which roads are ideal for conversion to shared access by minimizing impact on traffic flow and maximizing equity.

## Data

For this project we use the Statistical Neighborhoods<sup>[4]</sup> and Street Centerline<sup>[5]</sup> data from the Denver Open Data Catalog, for Denver neighborhoods and streets, respectively.

## Street Selection Criteria

To serve as a framework for selecting candidate streets for testing maximum flow differences, we wanted to consider how cities previously selected streets for their respective open and shared street programs during the pandemic. Boulder published their Low-Stress Walk and Bike Network Plan,<sup>[6]</sup> in which they identify four main criteria for street selection, those being:

- Equity Index
- Population and Employment Density
- Traffic Accidents
- Key Destinations

In this project, we use the equity index and population and employment density to identify candidate neighborhoods from which to pick streets, and we use traffic accidents and key destinations to identify candidate streets within those neighborhoods.

## Candidate Neighborhoods

In Boulder's Network Plan, they develop an equity index consisting of the following:

- % population with a disability
- % families living below the poverty level
- % households with no vehicle
- % non-white population
- % population under 17 or over 65 years old

The Network Plan then ranks neighborhoods within Boulder by census tract, and gives higher prioritization to those with a lower equity index.

In the Open Data Catalog, Denver defines their own equity index<sup>[7]</sup> by the following criteria:

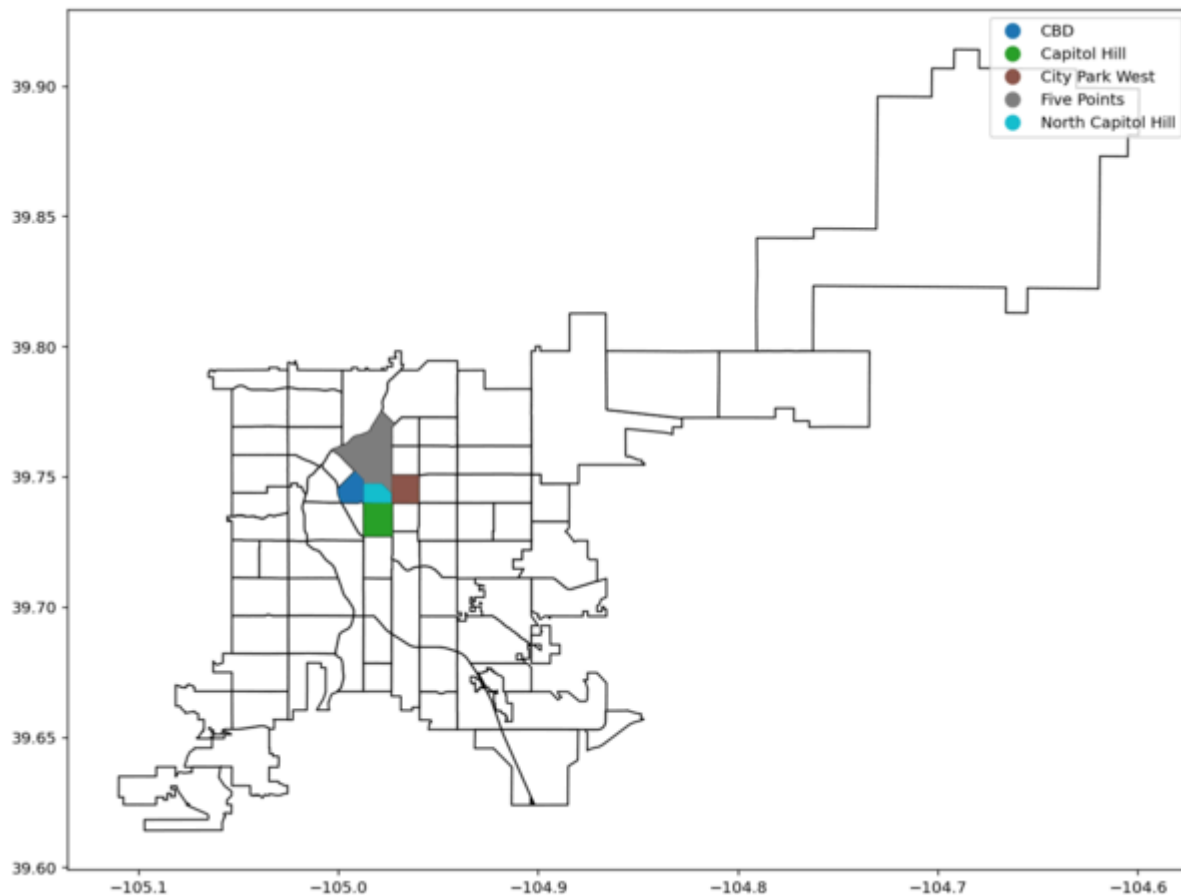
- Socioeconomic score
- Access to care score
- Morbidity score
- Mortality score
- Built environment score

Although these criteria are not the same as Boulder's, we can use Denver's equity index to inform neighborhood selection, and give higher priority to neighborhoods with lower equity index scores.

The other criteria as defined by Boulder's Plan that we use for neighborhood selection is population and employment density. The Open Data Catalog provides Pedestrian Demand Index data that "estimates the demand for walking in different areas of the City and County of Denver based on population and employment density."<sup>[8]</sup> Both Boulder and this project give higher prioritization to neighborhoods with higher population and employment density.

By sorting Denver neighborhoods (as defined by the Statistical Neighborhoods data) using the difference of their pedestrian demand index and equity index (i.e., high pedestrian demand and low equity most highly favored), we identified the following five neighborhoods as the best candidates for shared street consideration:

- CBD
- Capitol Hill
- City Park West
- Five Points
- North Capitol Hill



## Candidate Streets

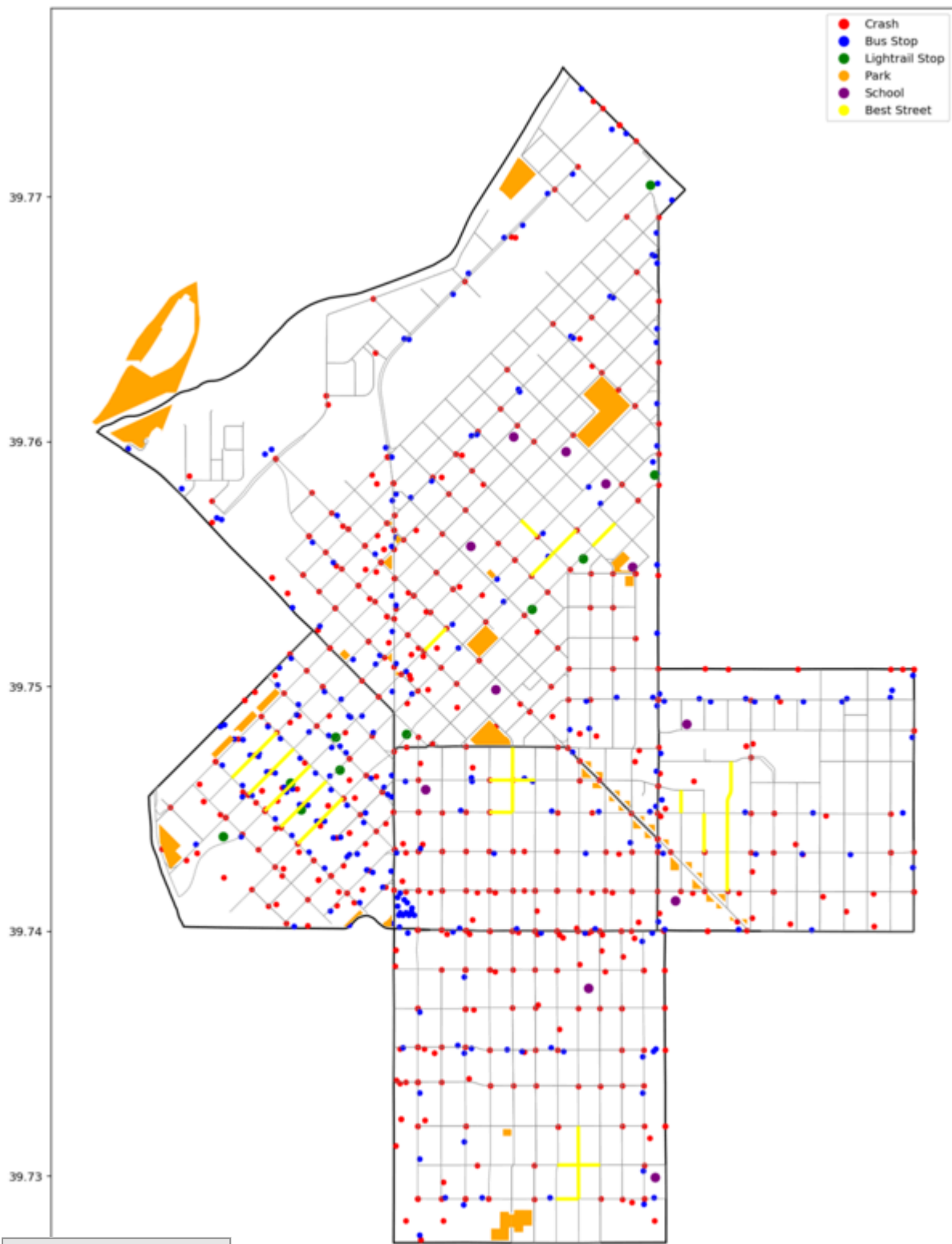
Boulder's Network Plan specifies that it gives highest priority for shared street conversion to streets who have a high frequency of "pedestrian-involved, bicyclist-involved, and killed or seriously injured (KSI) crashes." The Open Data Catalog has significant crash data,<sup>[9]</sup> and we subset this data by those that were pedestrian-involved, bicyclist-involved, or KSI crashes.

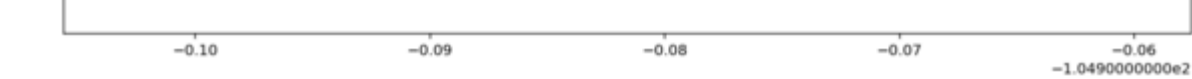
Boulder's other street selection criteria is those that are near key destinations. The Network Plan defines key destinations as including libraries, fitness centers, trailheads, restaurants, cafes, and grocery stores, but due to research and public input, defines the following three as those given consideration:

- Parks
- Schools
- Transit stops

For Denver, we collected park data from the Open Data Catalog,<sup>[10]</sup> school data from Open Street Map,<sup>[11]</sup> and bus and light rail stop data from the RTD website.<sup>[12]</sup>

With all of this data, Boulder specifies that they consider the number of crashes and key destinations within 1/4 mile of each street. By creating a 1/4 mile buffer around each street within the identified candidate neighborhood and counting the percentage of crashes and key destinations within those buffers, we can identify ideal candidate streets within each neighborhood.





With several potential candidate streets identified, we can find the maximum flow within the street network with and without those streets to find which streets have minimal traffic impact.

# Algorithms

## Assumptions

- Maximum flow is the most important quantity of interest. While transit time, time spent in congestion, and ease of the route and potentially as important as the flow rate, we assume the flow rate is the most salient factor in policy development.
- Road Segment capacity is determined by the number of lanes, maximum speed, and a function that maps the speed to a safe following distance. Even though this simplifying assumption falls short at intersections and junctions where human factors cause slowdowns, it is necessary to model flow as required by the restrictions of a network flows model. However, the assumption still allows the algorithm to set an upper bound on the max-flow.

## Determining Road Capacity

The flow rate of a roadway is determined by the density of traffic and the speed of that traffic. The density and speed have a non-linear relationship which produces a reasonably complex flow rate over combinations of speed and density. Still, they will generally produce a curve with a maximum flow rate. A simple thought experiment can understand this behavior: if there are no cars on the road, the cars may reach the maximum speed, but the flow rate will be low; if the roadway has a very high density of vehicles, the speed of traffic will be far from the maximum (potentially as low as zero); therefore, there must be some point in between these two extremes where traffic flow is maximized.

Since this behavior is dependent on a locale, local data is needed to characterize the maximum flow rate across any roadway. As such data is challenging to produce, we choose a simpler model to estimate the maximum flow rate.

Moore et al., in "Maximum flow in road networks with speed-dependent capacities-application to Bangkok traffic" (2013), suggest a model of traffic flow based on the max speed, number of lanes, and the safe following distance. Their model is expressed as  $C = \frac{s}{\delta}$  where C is the capacity, s is the maximum speed, and  $\delta$  is the separation of cars.

We use the same safe following distance data as Moore et al. from Toyota listed in their paper.

The code used to perform this operation can be found below:

```
from scipy.interpolate import UnivariateSpline

# Moore et al (2013) safe breaking distances
MOORE_AFTER_BREAK_SPLINE = UnivariateSpline(
    [20, 30, 40, 50, 60, 70, 80, 90, 100],
    [3.9, 6, 11, 18, 27, 39, 54, 58, 84],
)
MOORE_BEFORE_BREAK_SPLINE = UnivariateSpline(
    [20, 30, 40, 50, 60, 70, 80, 90, 100],
    [6, 8, 11, 14, 17, 19, 22, 25, 28],
)
```



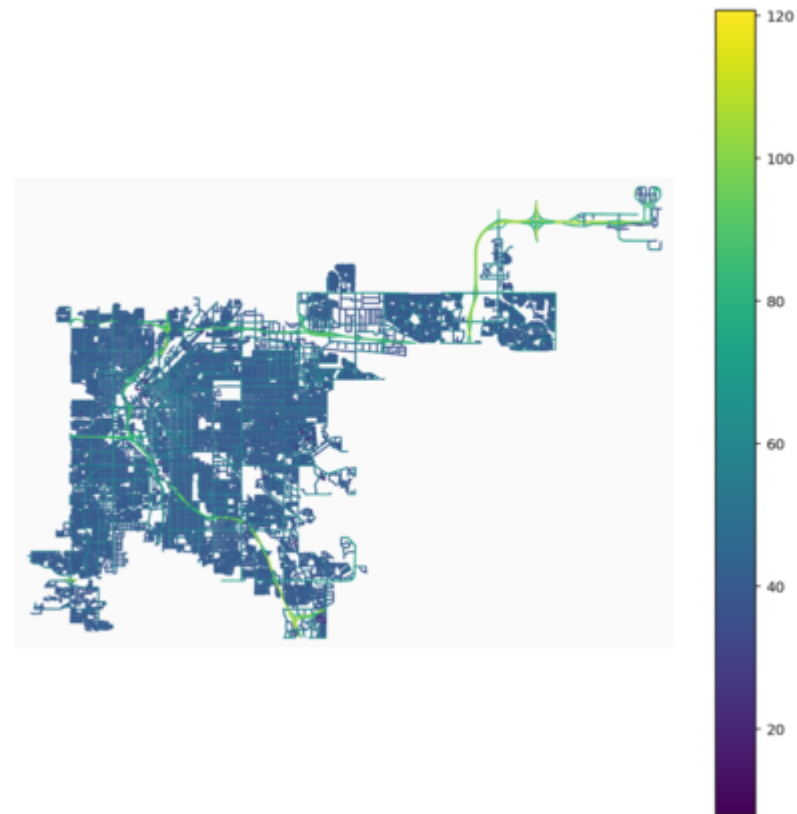
```

MOORE_SAFE_BREAKING_DISTANCE = lambda x: MOORE_AFTER_BREAK_SPLINE(
    x
) + MOORE_BEFORE_BREAK_SPLINE(x)

def capacity_moore(lanes: float, max_speed: float):
    """Maximum capacity for a road network based on
    "Maximum flow in road networks with speed-dependent capacities-application to Bangkok traffic", Moore et al, 2013
    """
    return 1000 * max_speed / (MOORE_SAFE_BREAKING_DISTANCE(max_speed) + 5) * lanes

```

The capacities for the Denver network are show below:



## Max-Flow with Preflow Push

The Generic Max-Flow Algorithm with Preflow Push is an algorithm that finds the maximum flow between a source and a sink node in a connected graph. The reason we chose this algorithm is that it has a strong polynomial time complexity, making it very efficient for large graphs. The critical step in the Generic Max-Flow Algorithm with Preflow Push is that at each iteration, a node, call it node  $i$ , with excess ( $>0$ ) flow, and distance label  $d$  is chosen. This excess is succesively discharged across arcs originating from node  $i$  in current residual graph, until the excess is 0. The arcs must be both admissible, and have positive capacity. If the excess on a node is positive, and there are no admissible arcs, then the distance label of node  $i$  is changed until an admissible arc is created.

This process allows the maximum-flow value to accumulate very quickly on the sink node. The downside of the Generic Max-Flow Algorithm with Preflow Push is that after the max-flow value has been pushed to the sink node, there may still be excess flow on other nodes in the graph. This condition violate the mass-balance constraints of the max-flow problem formulation. Using the same methodology described aboce, the remaining excess on each node is pushed back toward the source node until no excess remains, except on the sink and source node.

## Initialization

To initialize the Generic Max-Flow Preflow Push Algorithm, a capacitated network, a source node  $s$ , and a sink node  $t$ , are chosen. Starting from the sink node, distance labels,  $d(i)$ , are calculated and assigned to each node,  $i$ , in the network using a Breadth First Search Labeling Algorithm. Given a node  $i$ , and a node  $j$  in the adjacency list of node  $i$ , we define an arc  $(i,j)$  to be admissible if  $d(i) = d(j) + 1$ . Primary Step: The adjacency list,  $A(s)$ , of the source nodes is scanned, and a node  $j$  from  $A(s)$  is chosen. Notice that the distance label  $d(s) = d(j) + 1$  for all  $j$  in  $A(s)$ , so the arc  $(s,j)$  is an admissible arc. A flow value equal maximum capacity of the arc  $(s,j)$  is pushed from  $s$  to  $j$ , yielding an excess flow  $e(j) = c_{(s,j)}$  on node  $j$ . The residual graph is updated so that the capacity of arc  $(j,s) = e(j)$ . This step is repeated for all nodes in  $A(s)$ . The distance label of the source node,  $d(s)$ , is changed to equal the number of nodes in the network. This disconnects the  $s$  from the sink node in the original graph, and ensure that the algorithm does not push any flow from the source to sink node again.

## Push or Relabel

Once the network has been initialized, the algorithm either **pushes** flow from an active node along an admissible arc, or **relabels** an active node to create an admissible arc. Active nodes are nodes with a positive excess flow on the, excluding the sink and source nodes.

In the Push/Relabel step the network is scanned to see if an active node exists.

- If such a node exists, call it node  $i$ , scan the adjacency list,  $A(i)$ , for an an admissible arc.
  - If an admissible arc,  $(i,j)$ , exists, then:
    - $\text{flow}(i,j) = \min(\text{capacity}(i,j), \text{excess}(i))$  is pushed across the arc  $(i,j)$ .
    - The excess,  $\text{excess}(i)$ , is decreased by  $\text{flow}(i,j)$ .
    - The excess,  $\text{excess}(j)$ , is increased by  $\text{flow}(i,j)$ .
    - The capacity of arc  $(i,j)$  is decreased by  $\text{flow}(i,j)$ .
    - The capacity of arc  $(j,i)$  is decreased by  $\text{flow}(i,j)$ .
  - If an admissible arc,  $(i,j)$ , does not exist, then the distance label,  $d(i)$ , is relabeled:
    - $d(i) = \min\{d(j) + 1: j \text{ is in } A(i) \text{ and the capacity of arc } (i,j) \text{ is greater than } 0\}$ .

## All Pairs Impact

To understand the impact of removing a road segment (length between two intersections), we compute the max-flow between all pairs of points on the network with the segment present and the network with the segment removed. The following code performs this action:

```
import networkx as nx

def removal_impact(
    network: nx.DiGraph,
    arc: (int, int, int),
    Typesetting math: 100%
```

```

) -> float:

net = nx.DiGraph(network)
edges = ox.utils_graph.graph_to_gdfs(network, edges=True, nodes=False)
area_of_interest = edges.loc[[arc]].to_crs(epsg=util.CONFIG.local_crs) \
    .geometry.representative_point().buffer(radius) \
    .to_crs(epsg=util.CONFIG.base_crs).iloc[0]

net = nx.DiGraph(ox.truncate.truncate_graph_polygon(net, area_of_interest))

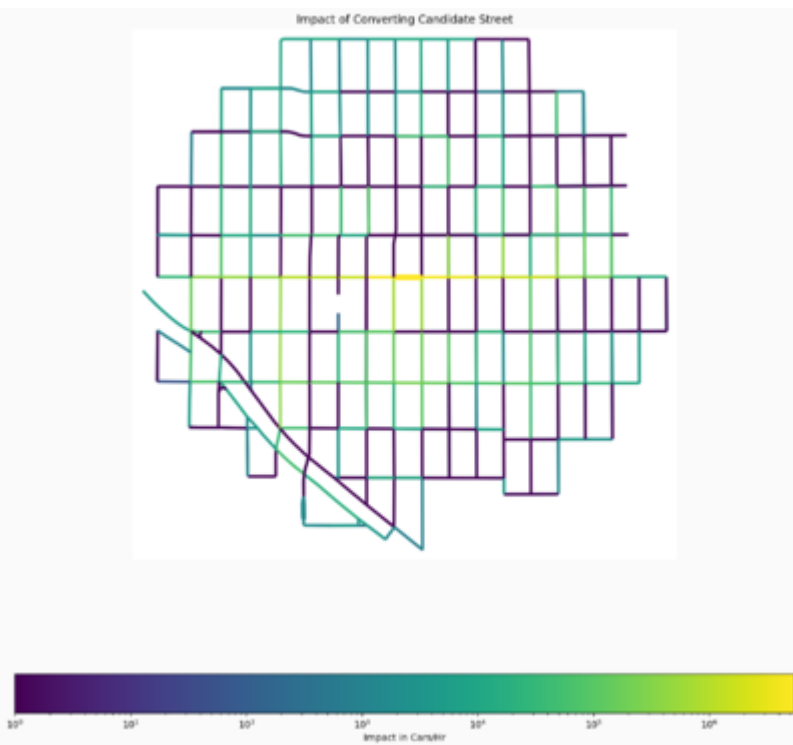
unperturbed_flow_values, _unperturbed_flows = util.all_pairs_max_flow(net, net.nodes)
net.get_edge_data(arc[0], arc[1])["capacity"] = 0
if (arc[1], arc[0]) in net.edges:
    net.get_edge_data(arc[1], arc[0])["capacity"] = 0

perturbed_flow_values, _perturbed_flows = util.all_pairs_max_flow(net, net.nodes)
return {x: perturbed_flow_values[x] - unperturbed_flow_values[x] for x in perturbed_flow_values.keys()}

```

(For the full code see: <https://github.com/schmidmt/Finding-Optional-Shared-Streets/blob/main/notebooks/StreetSelection.ipynb>)

For a given street segment, the difference in max-flow networks can be examined to determine the impact. An example of such a network is below



# Results and Policy

## Top 3 Candidates

The following represent the top three candidates for conversion to shared streets.

### Curtis Street (15th Street – 17th Street)

**Impact:** -1.8 Cars/Hr

Adjacent to 16th street mall, this area has a significantly higher occurrence of traffic accidents and could provide a protected pedestrian area with a minimal impact on traffic. There are also bus stops and lightrail stations nearby, so it is part of a major hub for commuters to Downton Denver and Auraria Campus. We recommend that the Denver Department of Transportation (DOTI) consider this street and adjacent streets while completing the feasibility study to expand the Open and Shared Streets Initiative.



### Marion Street (East 18th Ave – East 19th Ave)

**Impact:** -5.5 Cars/Hr

In a primarily residential block, this segment is adjacent to Hillcrest middle school. Converting this street, and possibly nearby streets to shared access could provide safer routes for parents to walk their kids to school. It also could provide reasonable access to a place for communities to interact and play.



### **Lafayette Street (East 17th Ave – East 18th Ave)**

**Impact:** -6 Cars/Hr

Lafayette Street sits on the other side of a residential block from an existing park. It is very close to Hillcrest Middle School. Converting this street, and possibly nearby streets to shared access could provide safer routes for parents to walk their kids to school. This closeness could extend the existing pedestrian space and provide additional safe outdoor space for the area's residents.



**Marion Street (East 18th Ave – East 19th Ave) & Lafayette Street (East 17th Ave – East 18th Ave)**

It should be noted that these streets are within 1 block of each other, indicating that the area around these streets has a high pedestrian demand, 2 different primary key destinations, a middle school and a park. Removing these streets will have a low overall impact on traffic flow through this area, so we recommend that the Denver DOTI consider these streets while completing the feasibility study to expand the Open and Shared Streets Initiative.

**All Candidates Impact By Neighborhood**

The following tables detail the impact of removing the stated road segment from the traffic network and determining the impact on all-pairs max-flow, sorted by impact and grouped by heighborhood.

CBD				
impact	road	between_left	between_right	neighborhood
-1.59	Curtis Street	15th Street	17th Street	CBD
-7.88	California Street	17th Street	15th Street	CBD
-9.93	Stout Street	15th Street	17th Street	CBD
-12.22	Welton Street	15th Street	17th Street	CBD
-15.98	Champa Street	17th Street	15th Street	CBD

Capitol Hill				
impact	road	between_left	between_right	neighborhood
-14.66	Clarkson Street	East 8th Avenue	East 9th Avenue	Capitol Hill
-19.96	Clarkson Street	East 9th Avenue	East 10th Avenue	Capitol Hill
-24.07	East 9th Avenue	Clarkson Street	Emerson Street	Capitol Hill
-27.76	East 9th Avenue	Washington Street	Clarkson Street	Capitol Hill
-45.61	East 8th Avenue	Clarkson Street	Washington Street	Capitol Hill

City Park West				
impact	road	between_left	between_right	neighborhood
-5.72	Marion Street	East 18th Avenue	East 19th Avenue	City Park West
-6.45	Lafayette Street	East 18th Avenue	East 17th Avenue	City Park West
-7.21	Humboldt Street	East 17th Avenue	East 18th Avenue	City Park West
-15.48	Humboldt Street	East 17th Avenue	East 16th Avenue	City Park West
-25.94	Humboldt Street	East 18th Avenue	East 20th Avenue	City Park West

Five Points				
impact	road	between_left	between_right	neighborhood
-11.81	California Street	27th Street	28th Street	Five Points
-11.86	California Street	26th Street	27th Street	Five Points
-12.46	27th Street	Stout Street	Champa Street	Five Points
-14.00	Stout Street		Park Avenue West	Five Points
-14.71	Welton Street	28th Street	29th Street	Five Points

North Capitol Hill				
impact	road	between_left	between_right	neighborhood
-11.57	Pennsylvania Street	East 18th Avenue	East 19th Avenue	North Capitol Hill
-12.63	East 19th Avenue	Pearl Street	Pennsylvania Street	North Capitol Hill
-13.28	East 19th Avenue	Logan Street	Pennsylvania Street	North Capitol Hill
-14.78	Pennsylvania Street	East 19th Avenue	East 20th Avenue, 22nd Street	North Capitol Hill
-31.66	East 18th Avenue	Pennsylvania Street	Logan Street	North Capitol Hill

## Further Research

- Due to the way candidate streets were selected, many of the streets were uninteresting in the actual context of the city. Additionally, many proxy variables used in the context of optimization are too general to quantify. We might relax the QoI and consider additional streets within these neighborhoods, or streets close to heavily



- trafficked paths.
- Due to the nature of neighborhood selection, only neighborhoods in downtown Denver were considered. Using different neighborhood selection criteria or manually selecting neighborhoods outside of downtown could provide interesting results.
  - Using just Pedestrian Demand and Equity as factors to select neighborhoods is not very rigorous, and additional criteria could be considered. Additionally, Denver's Equity Index was used in place of Boulder's, but they're not based on the same criteria. Creating unique neighborhood selection criteria for each city would give more meaningful results.
- Like with neighborhood selection criteria, specific criteria were used for street selection. Depending on the intention of the city converting their streets, different key destinations could be considered, like arenas, cafes, or hospitals. Cities with very considerable public transportation networks might give streets near those stops even heavier consideration.
- We used the Pedestrian Demand Index to inform the neighborhoods that we investigated, weighting neighborhoods with a high Pedestrian Demand Index Score higher than neighborhoods with a low Pedestrian Demand Index Score. This led us to investigate neighborhoods like North Capitol Hill, the Central Business District, and West City Park, which are neighborhoods with plenty of sidewalks and bike lanes. This led to the realization that basic pedestrian infrastructure, like sidewalks, bike lanes and parks, can lead to increased Pedestrian Demand, as people desire to live and move to neighborhoods that are supported such infrastructure. Light rail stations and the River North area in Denver are examples of infrastructure leading to increased pedestrian demand.
  - Considering this, it may be wise to rerun our analysis, weighting neighborhoods with a low Pedestrian Demand Index score higher than neighborhoods with a high Pedestrian Demand Index Score. Developing pedestrian infrastructure via Open and Shared Streets in these neighborhoods may lead to an increased Pedestrian Demand Index Score as these neighborhoods become more pleasant to live in, and possibly improve the overall equity score of these neighborhoods.

## Generality of Code

The code used to generate these recommendations is general to any area with streets in OpenStreetMaps and an equivalent measure of inequity (or any objective function).

## GitHub

- GitHub Repository (<https://github.com/schmidmt/Finding-Optional-Shared-Streets>)

## Tutorial

- Jupyter Notebook Tutorial (<https://github.com/schmidmt/Finding-Optional-Shared-Streets/blob/main/notebooks/Finding%20Optimal%20Shared%20Streets%20Tutorial.ipynb>)

## Resources

Moore, E.J., Kichainukon, W. and Phalavonk, U., 2013. Maximum flow in road networks with speed-dependent capacities-application to Bangkok traffic. *Songklanakarin Journal of Science & Technology*, 35(4).

Boeing, G. 2017. OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks. *Computers, Environment and Urban Systems* 65, 126-139. doi:10.1016/j.compenvurbsys.2017.05.004

Katz, Danny, Locantore, Jill, 2021, “Denver Shared and Open Streets program working...”, *DenverStreetPartnership*



“2019 And Bike Network Plan Boulder - Bouldercolorado.gov.” *2019 Boulder Low-Stress Walk & Bike Network Plan* , City of Boulder, 2019, <https://bouldercolorado.gov/media/4530/download?inline=>.

Ahuja, K. R., et al, “Network Flows: Theory Algorithms & Applications,” Prentice Hall-1993

1. ↑ <https://www.denvergov.org/Government/Agencies-Departments-Offices/Agencies-Departments-Offices-Directory/Department-of-Transportation-and-Infrastructure/Programs-Services/Shared-Streets>
2. ↑ <https://denverstreetspartnership.org/project/press-release-denverites-strongly-support-citys-shared-and-open-streets-efforts-to-expand-bicycle-network/>
3. ↑ <https://denverstreetspartnership.org/wp-content/uploads/2020/04/Media-Release-Shared-Streets-1.pdf>
4. ↑ <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-statistical-neighborhoods>
5. ↑ <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-street-centerline>
6. ↑ <https://bouldercolorado.gov/projects/low-stress-walk-and-bike-network-plan>
7. ↑ <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-equity-index-2020-neighborhood>
8. ↑ <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-pedestrian-demand-index>
9. ↑ <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-traffic-accidents>
10. ↑ <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-parks>
11. ↑ <https://www.openstreetmap.org/>
12. ↑ <https://gis-rtd-denver.opendata.arcgis.com/maps/e14366d810644a3c95a4f3770799bd54/about>

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Finding\_Optimal\_Shared\_Streets\_in\_Denver&oldid=3856"

- This page was last modified on 4 May 2022, at 08:04.
- This page has been accessed 998 times.

# Finding Your Optimal Home Subject to Personal Constraints

From CU Denver Optimization Student Wiki

This Project is by Alex Semyonov, Jacob Dunham, and Orlando Gonzalez.

## Abstract

Link to GitHub Repository: in progress

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Finding\_Your\_Optimal\_Home\_Subject\_to\_Personal\_Constraints&oldid=4447"

- 
- This page was last modified on 9 November 2023, at 20:52.
  - This page has been accessed 5 times.

# Food Deserts

From CU Denver Optimization Student Wiki

The authors of this project are Drew Horton and Rebecca Robinson.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Food\\_Deserts&oldid=3107](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Food_Deserts&oldid=3107)"

- 
- This page was last modified on 12 April 2021, at 11:15.
  - This page has been accessed 47 times.

# Gentrification

From CU Denver Optimization Student Wiki

This project is concerned with Identifying Gentrification. I will use structure age, income, and race data to inform a linear program to find out which areas are likely to be gentrified, and make suggestions on what to do with that information.

Retrieved from "<https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Gentrification&oldid=1610>"

- 
- This page was last modified on 8 November 2018, at 16:06.
  - This page has been accessed 358 times.

# Grace Truong

From CU Denver Optimization Student Wiki

## Contents

- 1 About me
- 2 Education
- 3 Projects
- 4 GitHub

## About me

Hi! My name is Grace Truong and I am currently a first year PhD student at CU Denver.

## Education

I received my Bachelor's in Mathematics at Regis University in May 2024.

## Projects

Spring 2025: Burnside's Lemma (for Applied Graph Theory)

## GitHub

Github (<https://github.com/Grace-Truong>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Grace\\_Truong&oldid=4882](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Grace_Truong&oldid=4882)"

Category: Contributors



# Gradient Descent Method in Solving Convex Optimization Problems

From CU Denver Optimization Student Wiki

During the class, Siyuan\_Lin and Gregory\_Matesi have learned various knowledge of linear programming problems. So, we take this project as a chance to learn something new. We are interested in solving the nonlinear programming problems and we take the convex optimization problems as an example.

## Contents

- 1 Project Abstract
- 2 Gradient descent
- 3 Limitations of Gradient Descent
  - 3.1 Zigzagging Issue
  - 3.2 Non-Convergence Issue
  - 3.3 Does not work for linear programming
- 4 Implementation
  - 4.1 Implementation in AMPL
  - 4.2 Other options
- 5 Newton methods
  - 5.1 Difference between newton methods and gradient descent?
- 6 Github and future work
- 7 References

## Project Abstract

The gradient descent method is a first-order iterative optimization algorithm for finding the minimum of a function. It is based on the assumption that if a function  $F(x)$  is defined and differentiable in a neighborhood of a point  $x_0$ , then  $F(x)$  decreases fastest along the negative gradient direction. It is a simple and practical method for solving optimization problems with an objective function that is well conditioned.

However, it comes with a few drawbacks. It can exhibit poor convergence if the problem is not strongly convex. Gradient descent may also exhibit zigzagging when the step size is too big and slow search when the step size is too small. Also, gradient descent may not perform well in a setting where the objective function is linear. Newton's method is more widely implemented by programmers in python and AMPL. Newton's method features use of the Hessian which corresponds to information about the curvature of the objective function. This information may help Newton's method performs better than relatively simpler gradient descent method. In particular, Newton's method is used in the Generalized Reduced Gradient (GRG) method implemented in the CONOPT solver in AMPL.

# Gradient descent

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. It originates from the Taylor Series, which represents a function by an infinite sum of terms that are calculated from the values of the function's derivatives at a single point. Let  $f(x)$  be any function infinitely differentiable around  $x = x_0$ .

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} \frac{f^n(x_0)}{n!(x-x_0)^k} \\ &= f(x_0) + f'(x_0)(x-x_0) + f'' \frac{(x_0)}{2!} (x-x_0)^2 + \dots \end{aligned}$$

So, when  $x$  is close to  $x = x_0$ , the function  $f(x)$  will have a close approximation with only the first two terms of Taylor Series.

$$f(x) \approx f(x_0) + f'(x_0)(x-x_0)$$

For a multivariable function, the approximation could be written as below:

$$f(x, y) \approx f(x_0, y_0) + \frac{\partial f(x_0, y_0)}{\partial x} (x-x_0) + \frac{\partial f(x_0, y_0)}{\partial y} (y-y_0)$$

The gradient  $\nabla f$  is a vector whose components are the first order partial derivatives of  $f$  at  $x_0$ .

$$\nabla f(x_0, y_0) = \begin{bmatrix} \frac{\partial f}{\partial x} (x_0, y_0) \\ \frac{\partial f}{\partial y} (x_0, y_0) \end{bmatrix}.$$

It can be interpreted as the "direction and rate of fastest increase". In order to find the minimum of the target function, we take the negative direction of the gradient, which will give us the max-rate descending direction.

Given initial point  $x_0$ , the gradient descent algorithm uses the following update to generate  $x_1, x_2, \dots$ , until a stopping condition is met:

from the current point  $x_k$  generate the next point  $x_{k+1}$  by

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k),$$

where  $\alpha_k$  is called the step size.

$\alpha_k$  and  $\epsilon$  are often chosen



1. as a fixed value if  $\nabla f$  is Lipschitz (rate of change is bounded) with the constant known or an upper bound of it known;
2. by line search;
3. by a method called Barzilai-Borwein with nonmonotone line search.

## Limitations of Gradient Descent

There are certain limitations of the gradient method.

### Zigzagging Issue

For poorly conditioned convex problems, gradient descent increasingly 'zigzags' as the gradients point nearly orthogonally to the shortest direction to a minimum point.

### Non-Convergence Issue

When the step size is too big, it can cause overshooting. When the step size is too small, the gradient descent may never converge because it is trying really hard to exactly find a local minimum.

### Does not work for linear programming

We examined whether or not the gradient descent method was a good option for solving linear programming problems.

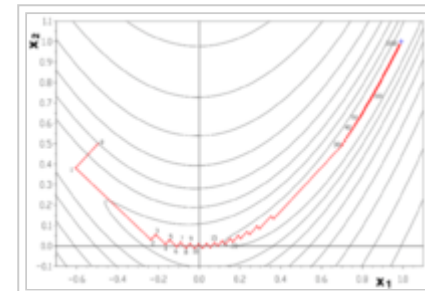
## Implementation

### Implementation in AMPL

We tried to implement gradient descent to solve a convex quadratic problem in AMPL but found that there were no readily available solvers that did so.

### Other options

We are working on modeling a convex quadratic programming problem in AMPL and then sourcing it to a gradient descent solver in python. There may not be any libraries that use gradient descent in python so we may need to write our own.



When the step size is too large, the iteration diverges.

# Newton methods

## Difference between newton methods and gradient descent?

We looked at the difference between newton methods for optimization and gradient descent. We examined whether or not newton methods were typically a better option for solving convex quadratic optimization problems. We also examined whether or not there were any available libraries that made use of newton methods.

## Github and future work

Please see our Github repository for code and a pdf/ppt of our slide show presentation.

[https://github.com/GregoryMatesi/GradientDescent\\_MATH5593](https://github.com/GregoryMatesi/GradientDescent_MATH5593)

## References

1. <sup>↑</sup> <https://ampl.com/SOLVERS/conopt3.pdf>
2. “Nonlinear Programming,” Athena Scientific, by Dimitri P. Bertsekas
3. "Gradient descent", [https://en.wikipedia.org/wiki/Gradient\\_descent#](https://en.wikipedia.org/wiki/Gradient_descent#)
3. Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Gradient\\_Descent\\_Method\\_in\\_Solving\\_Convex\\_Optimization\\_Problems&oldid=2471](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Gradient_Descent_Method_in_Solving_Convex_Optimization_Problems&oldid=2471)"

- 
- This page was last modified on 3 December 2019, at 14:02.
  - This page has been accessed 5,084 times.

# Gradient Descent Method in Solving Convex Quadratic Optimization Problems

From CU Denver Optimization Student Wiki

During the class, Siyuan\_Lin and Gregory\_Matesi have learned various knowledge of linear programming problems. So, we take this project as a chance to learn something new. We are interested in solving the nonlinear programming problems and we take the convex optimization problems as an example.

## Contents

- 1 Project Abstract
- 2 Gradient descent
- 3 Problems with Gradient Descent
  - 3.1 Convergence Analysis
  - 3.2 Zig zagging
  - 3.3 Does not work for linear programming
- 4 Implementation
  - 4.1 Implementation in AMPL
  - 4.2 Other options
- 5 Newton methods
  - 5.1 Difference between newton methods and gradient descent?
- 6 Github and future work
- 7 References

## Project Abstract

The gradient descent method is a first-order iterative optimization algorithm for finding the minimum of a function. It is based on the assumption that if a function  $F(x)$  is defined and differentiable in a neighborhood of a point  $x_0$ , then  $F(x)$  decreases fastest along the negative gradient direction. It is a simple and practical method for solving optimization problems with a quadratic objective function that is well conditioned.

However, it comes with a few drawbacks. It can exhibit poor convergence if the problem is not strongly convex. Gradient descent may also exhibits zigzagging when the step size is too big and slow search when the step size is too small. Also, gradient descent may not perform well in a setting where the objective function is linear. Newtons method is more widely implemented by programmers in python and AMPL. Newtons method features use of the hessian which corresponds to information about the curvature of the objective function. This information may help Newtonian methods perform better than relatively simpler gradient descent method. Newtons method is used in the Generalized Reduced Gradient (GRG) method implemented in the CONOPT solver in AMPL.

# Gradient descent

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. It originates from the Taylor Series, which represents a function by an infinite sum of terms that are calculated from the values of the function's derivatives at a single point. Let  $f(x)$  be any function infinitely differentiable around  $x = x_0$ .

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} \frac{f^n(x_0)}{n!(x-x_0)^k} \\ &= f(x_0) + f'(x_0)(x-x_0) + f'' \frac{(x_0)}{2!} (x-x_0)^2 + \dots \end{aligned}$$

So, when  $x$  is close to  $x = x_0$ , the function  $f(x)$  will have a close approximation with only the first two terms of Taylor Series.

$$f(x) \approx f(x_0) + f'(x_0)(x-x_0)$$

For a multivariable function, the approximation could be written as below:

$$f(x,y) \approx f(x_0,y_0) + \frac{\partial f(x_0,y_0)}{\partial x} (x-x_0) + \frac{\partial f(x_0,y_0)}{\partial y} (y-y_0)$$

The gradient  $\nabla f$  is a vector whose components are the first order partial derivatives of  $f$  at  $x_0$ .

$$\nabla f(x_0,y_0) = \begin{bmatrix} \frac{\partial f}{\partial x} (x_0,y_0) \\ \frac{\partial f}{\partial y} (x_0,y_0) \end{bmatrix}.$$

It can be interpreted as the "direction and rate of fastest increase". In order to find the minimum of the target function, we take the negative direction of the gradient, which will give us the max-rate descending direction.

## Problems with Gradient Descent

### Convergence Analysis

We examined why the gradient descent method may not converge quickly. We examined what can go wrong and cause poor convergence.

## Zig zagging

We examined how gradient descent can "zig zag" slowly towards an optimum along a flat curved valley in the objective function.

## Does not work for linear programming

We examined whether or not the gradient descent method was a good option for solving linear programming problems.

# Implementation

## Implementation in AMPL

We tried to implement gradient descent to solve a convex quadratic problem in AMPL but found that there were no readily available solvers that did so.

## Other options

We are working on modeling a convex quadratic programming problem in AMPL and then sourcing it to a gradient descent solver in python. There may not be any libraries that use gradient descent in python so we may need to write our own.

We are also looking at using the CONOPT solver in AMPL that uses newton methods<sup>[1]</sup>.

# Newton methods

## Difference between newton methods and gradient descent?

We looked at the difference between newton methods for optimization and gradient descent. We examined whether or not newton methods were typically a better option for solving convex quadratic optimization problems. We also examined whether or not there were any available libraries that made use of newton methods.

# Github and future work

Please see our Github repository.

[https://github.com/GregoryMatesi/GradientDescent\\_MATH5593](https://github.com/GregoryMatesi/GradientDescent_MATH5593)

# References

1. ↑ <https://ampl.com/SOLVERS/conopt3.pdf>

- This page was last modified on 28 November 2019, at 10:58.
- This page has been accessed 4,505 times.

# Graph Coloring Variants

From CU Denver Optimization Student Wiki

Welcome to Abigail Nix and Ari Holcombe Pomerance's project page! In this project, we will explore list coloring, a common variant of vertex coloring of graphs. We begin with an introduction to list coloring, with definitions and some examples. We then establish the list version of Brooks' Theorem and show how this implies Brooks' Theorem in the standard vertex coloring setting. Finally, we introduce correspondence coloring as a way to generalize list coloring further.

## Contents

- 1 Introduction
- 2 List Extension of Brooks' Theorem
- 3 Correspondence Coloring
- 4 References
- 5 External links

## Introduction

We start by defining list coloring, the setting we will discuss for most of the project. Just as in standard vertex coloring, we still require that any two adjacent vertices cannot receive the same color. However, in this new setting, each vertex must be colored from some subset of the colors, rather than allowing any vertex to receive any color. We now care about finding the minimum number of color options we need to provide to each vertex such that we can still color  $G$ , regardless of what color options are given to each vertex. We formally define list coloring and the list chromatic number below.

**Definition 1.**<sup>[1]</sup> For a graph  $G$ , a **list assignment**  $L$  assigns each vertex  $v \in V(G)$  a set  $L(v)$  of colors allowed at  $v$ . An  **$L$ -coloring** is a proper coloring  $\phi$  of  $G$  such that  $\phi(v) \in L(v)$  for all  $v$ . A graph  $G$  is  **$k$ -choosable** or **list  $k$ -colorable** if it has an  $L$ -coloring whenever  $|L(v)| \geq k$  for all  $v$ . The **list chromatic number** or **choice number** or **choosability**  $\chi_l(G)$  is the minimum  $k$  such that  $G$  is  $k$ -choosable.

We can see that the standard formulation of vertex coloring is simply a special case of list coloring, where every vertex receives the same list of allowable colors. If every vertex gets the same list in some graph  $G$ , then as long as these lists are at least as large as  $\chi(G)$ ,  $G$  can be colored from this list assignment. This implies that  $\chi_l(G) \geq \chi(G)$ , since we know that for this particular list assignment where every vertex gets the same list, the lists must be of size at least  $\chi(G)$ . With this observation that standard coloring is just a special case of list coloring, it may seem like list coloring should be easier than coloring, since we can have any colors in the lists of each vertex. However, finding the list chromatic number of a graph is a fundamentally different problem than finding the chromatic number, because we must consider all possible list assignments. In fact, by considering a particular list assignment, we can easily construct an example of a class of graphs where  $\chi_l(G) > \chi(G)$ . We know that all bipartite graphs are 2-colorable, since we can always assign one color to each part, but in our presentation we show that complete bipartite graphs  $K_{m,m}$  are not 2-choosable by assignment such that the graph is not colorable from these lists.

# List Extension of Brooks' Theorem

Recall that for standard coloring, we can use a greedy coloring of a graph  $G$  to get that  $\chi(G) \leq \Delta(G) + 1$ . The same argument also implies that  $\chi_l(G) \leq \Delta(G) + 1$ , since in a greedy list coloring with respect to some vertex order, each vertex will have at most  $\Delta(G)$  already colored neighbors. This result coming from greedy coloring can also be strengthened by classifying which graphs meet this bound with equality.

Brooks' Theorem (for standard coloring) states that  $\chi(G) \leq \Delta(G) + 1$ , with equality if and only if  $G$  has  $K_{\Delta(G)+1}$  (or an odd cycle when  $\Delta(G) = 2$ ) as a component. We will provide some results that enable us to establish a very similar result for list coloring. To prove this result, we first must establish a series of lemmas.

**Lemma 1.** (Vizing (1976)). Given a connected graph  $G$ , let  $L$  be a list assignment such that  $|L(v)| \geq d(v)$  for all  $v$ .

1. If  $|L(y)| > d(y)$  for some vertex  $y$ , then  $G$  is  $L$ -colorable.
2. If  $G$  is 2-connected and some two lists differ, then  $G$  is  $L$ -colorable.

For the next lemma, we need the following definition.

**Definition 2.** A graph  $G$  is  **$f$ -choosable** if it is  $L$ -colorable whenever  $|L(v)| \geq f(v)$  for each vertex  $v$ , where  $f : V(G) \rightarrow \mathbb{N}$ . The graph is **degree-choosable** if it is  $L$ -colorable whenever  $|L(v)| \geq d(v)$  for each vertex  $v$ .

**Lemma 2.** If a connected graph  $G$  has a degree-choosable induced subgraph  $H$ , then  $G$  is degree-choosable.

We will use Lemmas 1 and 2 to determine that a graph can be colored from its lists when each list is at least as large as the degree of the vertex. The next lemma is a structural result that will be helpful in the proof of Theorem 1.

**Lemma 3.** (Erdős-Rubin-Taylor (1979)). Every 2-connected graph  $G$  that is not a complete graph or odd cycle has an even cycle with at most one chord.

These three lemmas can be used to prove the following Theorem 1.

**Theorem 1.** (Borodin (1977), Erdős-Rubin-Taylor (1979)). If graph  $G$  is not degree-choosable, then every block of  $G$  is a complete graph or an odd cycle.

Finally, in our presentation, we show how this theorem implies the following list extension of Brooks' Theorem.

**Corollary 1.** If a connected graph  $G$  is not a complete graph or an odd cycle, then  $\chi_l(G) \leq \Delta(G)$ .

Since  $\chi(G) \leq \chi_l(G)$  for every graph  $G$ , note that this statement implies Brooks' Theorem. This corollary also classifies the types of connected graphs that have  $\chi_l(G) = \Delta(G) + 1$  (complete graphs and odd cycles).

## Correspondence Coloring



List coloring is just one of the many variants of graph coloring. Here we define correspondence coloring, a further generalization of the concept of list coloring. Correspondence coloring, or DP coloring (after Dvořák and Postle, 2016), was introduced as a way to allow for vertex identification. This is a common tool used in standard graph coloring, but it cannot be used with list coloring, since any two vertices might not have the same list, so might not be able to receive the same color. With correspondence coloring, we can make every list the same, which avoids this problem. In correspondence coloring, instead of two adjacent vertices not being able to receive the same color, as in list coloring, we establish a correspondence between the lists of vertices to determine what colors are forbidden at each vertex. This is formally defined below.

**Definition 3.**<sup>[2]</sup> A **correspondence assignment** for a graph  $G$  consists of a list assignment  $L$  and a function  $C$  that to every edge  $vw \in E(G)$  assigns a partial matching  $C_{vw}$  between  $L(v)$  and  $L(w)$ . An  $(L, C)$  **-coloring** of  $G$  is a function  $\varphi$  that assigns to each  $v \in V(G)$  a color  $\varphi(v) \in L(v)$  such that for every  $vw \in E(G)$  the vertices  $(v, \varphi(v))$  and  $(w, \varphi(w))$  are non-adjacent in  $C_{vw}$ . Now  $G$  is  $(L, C)$  **-colorable** if such an  $(L, C)$  -coloring exists. The **correspondence chromatic number** (or DP chromatic number),  $\chi_{corr}(G)$  is the minimum  $k$  such that  $G$  is  $(L, C)$  -colorable whenever  $|L(v)| \geq k$  for all  $v \in V(G)$ .

Note that if  $|L(v)| = k$  for each vertex  $v$ , we can find an equivalent correspondence where  $L(v) = [k]$  for all  $v \in V(G)$ . Essentially, this just means that we can relabel the colors in the lists of each vertex, as long as the underlying correspondence stays the same. This is the powerful aspect of correspondence coloring that allows for vertex identification to be used where it could not be for list coloring.

Just as we can have  $\chi_l(G) > \chi(G)$ , we can also have  $\chi_{corr}(G) > \chi_l(G)$ . In the presentation for this project, we demonstrate this on the 4-cycle, where there is one twist in the correspondence that makes the correspondence chromatic number strictly larger than 2.

## References

- ↑ Douglas B. West. *Combinatorial Mathematics*, 2021.
- ↑ Daniel W. Cranston. *Graph Coloring Methods*, 2024.

## External links

Our presentation slides can be found in the GitHub repository for this project (<https://github.com/aripom/Graph-Coloring-Variants>). Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Graph\\_Coloring\\_Variants&oldid=5013](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Graph_Coloring_Variants&oldid=5013)"

- This page was last modified on 5 May 2025, at 01:25.
- This page has been accessed 83 times.

# Gregory Matesi

From CU Denver Optimization Student Wiki

I am a fourth year undergraduate in the Applied Math Program at CU Denver. I am originally from Sandwich, IL. After spending four years in the USMC I decided to move to Denver.. After I graduate in December or 2019 I hope to continue at CU Denver in the MS in Statistics Program. I am currently working with Siyuan\_Lin on a project examining the gradient descent method for convex quadratic problems in AMPL: Gradient Descent Method in Solving Convex Quadratic Optimization Problems

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Gregory\\_Matesi&oldid=2128](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Gregory_Matesi&oldid=2128)"

Category: Contributors

- 
- This page was last modified on 10 November 2019, at 19:12.
  - This page has been accessed 1,528 times.

# Hanbyul Lee

From CU Denver Optimization Student Wiki

## About me

Hi, I am Hanbyul (Han) Lee, a first-year PhD student at CU Denver.

## Projects

Spring 2025: Degree Sequence (for Applied Graph Theory [MATH 6406])

## GitHub

Github (<https://github.com/otter275>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Hanbyul\\_Lee&oldid=4927](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Hanbyul_Lee&oldid=4927)"

Category: Contributors

- 
- This page was last modified on 21 April 2025, at 08:17.
  - This page has been accessed 58 times.

# History Of Linear Programming

From CU Denver Optimization Student Wiki

<big>**Computational LP Pioneers:**

Fourier [1826] studies the properties of systems of linear inequalities, more complex than systems of equations.

De la Vall Poussin [1911] develops an iterative procedure for linear minimax estimation which can be adjusted to solve linear optimization problems.

As early as 1930, A.N. Tolstoi described a number of solution approaches for transportation problems.

Kantorovich [1939] proposes rudimentary algorithm for linear programming applied to production planning.

George Dantzig proposes the Simplex Method in 1947.

Early works by Leontief, von Neumann and Koopsman directly influenced the theoretical development of linear programming.

From Dantzig's point of view: Not just a qualitative tool in the analysis of economic phenomena, but a method to compute actual answers.

Unfortunately, the 1975 Nobel Prize in Economics was awarded to Kantorovich and Koopsman, ignoring Dantzig's contribution. </big>

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=History\_Of\_Linear\_Programming&oldid=623"

- 
- This page was last modified on 30 November 2017, at 19:00.
  - This page has been accessed 1,487 times.

# Hope Haygood

From CU Denver Optimization Student Wiki

Hello! My name is Elizabeth Hope Haygood & I am currently a PhD student in the Mathematical & Statistical Sciences Department at the University of Colorado Denver. I obtained my Bachelor of Science in Mathematics with a focus in Professional Physics at the University of North Alabama in Florence, Alabama. When I am not enjoying Math, you can find me playing with my toddler. We enjoy biking, strolling, & hiking. I also really enjoy my share of reality television such as the Bachelorette & Keeping up with the Kardashians (so sad its cancelled!!).

During the Fall semester of 2020, Sandra Robles & I worked on Denver Fire Response Distances for a project in our Linear Programming class.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Hope\\_Haygood&oldid=2921](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Hope_Haygood&oldid=2921)"

Category: Contributors

- 
- This page was last modified on 9 November 2020, at 21:47.
  - This page has been accessed 15,801 times.

# Housing Assistance Program Allocation

From CU Denver Optimization Student Wiki

This project is by Courtney Franzen and Weston White

## Abstract

In Colorado, many homeowners face the risk of foreclosure. Losing one's home can result in economic and psychological hardship. The greater number of foreclosures within a neighborhood has rippling effects throughout our community in Denver. This project studies how housing assistance is offered throughout the Denver community and identifies policy that can be implemented to prevent a greater number of homes from being foreclosed upon. Historically, Colorado has offered assistance on a first come first served basis, which is oftentimes an inefficient and inequitable allocation of funds. To support our policy recommendations, we use a linear program that maximizes outreach in order to minimize the number of homes facing foreclosure within neighborhoods in need. We identify neighborhoods in need as those that face the largest amount of foreclosures. The number of homes that were foreclosed come from time series data that breaks down the demographics of individual neighborhoods in the Denver area. Our policy recommendations focus on promoting the availability of housing assistance to neighborhoods that are likely to use the available funds offered by the government which will assist them in paying their mortgages.

## GitHub

Housing Assistance Awareness GitHub ([https://github.com/WestonWhiteUCD/D2P\\_HAPA](https://github.com/WestonWhiteUCD/D2P_HAPA))

Click the link above to see the powerpoint slides and data/ampl files that were used during our project.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Housing\\_Assistance\\_Program\\_Allocation&oldid=4564](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Housing_Assistance_Program_Allocation&oldid=4564)"

- 
- This page was last modified on 5 December 2023, at 09:01.
  - This page has been accessed 38 times.

# Housing Distribution

From CU Denver Optimization Student Wiki

Suppose you are a developer in Denver. Building mansions may be the best way to make money, but selling all of them would be tough. On the other hand, you can fit more people into studio apartments, but not everyone will want those either. Linear Programming can be used to plan the correct distribution of types of housing to build.

The basic idea behind the linear program is to assign a variable to each type of housing you can build. You would then want to maximize the profit from selling these types of housing, subject to the constraints of what your budget is, what land is available to develop, and what people actually want to buy. We therefore end up with a linear program like:

$$\begin{aligned} &\text{Maximize} && \sum_{t \in T} p_t x_t \\ &\text{Subject to} && \sum_{t \in T} c_t x_t \leq b \\ & && \sum_{t \in T} s_t x_t \leq l \\ & && 0 \leq x_t \leq d_t \quad \forall t \end{aligned}$$

Where  $T$  is a set of the types of housing,  $p_t$ ,  $c_t$ ,  $s_t$ ,  $d_t$  are the profit, cost, area, and demand for each type,  $b$  is the budget, and  $l$  is the land available.

## Abstract

A housing developer has many different options for what to build. Apartments, or mansions? Condos, or family homes? The developer also has multiple constraints on what to build, such as, budget, avail- able land, demand for the different types of housing, etc. We use linear programming to model a typical housing developer's situation and optimize the profit for the developer. In particular, we study how changing the constraints will change the developer's optimal choices for housing. This will allow us to suggest under what conditions af- fordable housing is most likely to be built.

## Code and Poster

Here is a Github repository ([https://github.com/eric-d-culver/D2P\\_Project](https://github.com/eric-d-culver/D2P_Project)) with the AMPL code I used for my project and a PDF of the poster.

Created by Eric Culver

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Housing\\_Distribution&oldid=1776](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Housing_Distribution&oldid=1776)"

- This page was last modified on 29 November 2018, at 11:28.
- This page has been accessed 960 times.



# Http://math.ucdenver.edu/~sborgwardt/wiki/index.php/John McFarlane

From CU Denver Optimization Student Wiki

<inputbox> type=create width=100 break=no buttonlabel=Create new article default=(John McFarlane) Testing </inputbox>

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Http://math.ucdenver.edu/%7Esborgwardt/wiki/index.php/John\_McFarlane&oldid=2146"

- 
- This page was last modified on 11 November 2019, at 21:31.
  - This page has been accessed 96 times.

# Hungry for Equality: Fighting Food Deserts

From CU Denver Optimization Student Wiki

The authors of this project are Drew Horton and Rebecca Robinson.

## Contents

- 1 Abstract
- 2 Motivation
- 3 Data
- 4 The First Program
- 5 The Kolm-Pollak EDE and The Second Program
- 6 Results
  - 6.1 Comparison Between the Two Programs
- 7 Future Work and Policy Suggestions
  - 7.1 Healthy Food Financing Initiative
- 8 Links to Code and Slides
- 9 References

## Abstract

Food deserts are a form of food insecurity related to a lack of access to healthy, fresh, and affordable food. The US Department of Agriculture (USDA) defines a food desert in an urban area to be a region more than one mile away from a grocery store. According to the USDA, 13.7 million households in the U.S. experienced food insecurity in 2019, and this burden disproportionately affects marginalized communities. This problem has only be exacerbated by the ongoing COVID-19 pandemic, with an unprecedented number of people in extreme poverty, and many more at risk. To address the food insecurities in our community, we construct two integer programs that will produce an optimal distribution of grocery store locations. The first program minimizes the average distance of residents to a grocery store. For the second program, we minimize over the Kolm-Pollak EDE (Equally Distributed Equivalent), a measure of inequality. When we minimize the average distance, we ignore the worst off members in our communities, whereas in minimizing the inequality, we are ensuring we address the marginalized populations. Through comparing the resulting distributions produced from each program, we demonstrate the importance of assessing inequality within a community as a step towards addressing equity. Specifically, we hope to encourage policy makers to consider intervention strategies that prioritize relief in disproportionately affected communities.

# Motivation

A food desert\* in an urban area, as defined by the US Department of Agriculture, is a region more than one mile away from the closest grocery store.<sup>[1]</sup> Those that live in food deserts do not have easy access to health, fresh, and/or affordable food. In addition, those that require certain foods for cultural or health reasons (i.e. kosher, halal, gluten-free, etc), have even less access to foods they need. In food deserts, it is often cheaper for residents to purchase unhealthful foods. Between 1989 and 2005, the overall price of fruits and vegetables in the US increased by almost 75% while the price of fatty foods dropped by more than 26%.<sup>[2]</sup> Because of this, food deserts have greater effects on those in poverty. In addition, studies have show that "wealthy districts have three times as many supermarkets as poor ones do and white neighborhoods contain an average of four times as many supermarkets as predominantly black ones do."<sup>[3]</sup>

The ongoing COVID-19 pandemic has had a great impact on food insecurity, including food deserts. In fact, researchers found that there was a 32.3% increase in household food insecurity since the COVID-19 pandemic started, with 35.5% of food insecure households being considered as newly food insecure.<sup>[4]</sup> In addition, CDC research suggests that higher rate of COVID-19 infections among those in the Latino and Black populations could be influenced by the social determinants of food insecurity, and also by the lack of nutritional quality of food in food deserts.<sup>[4]</sup>

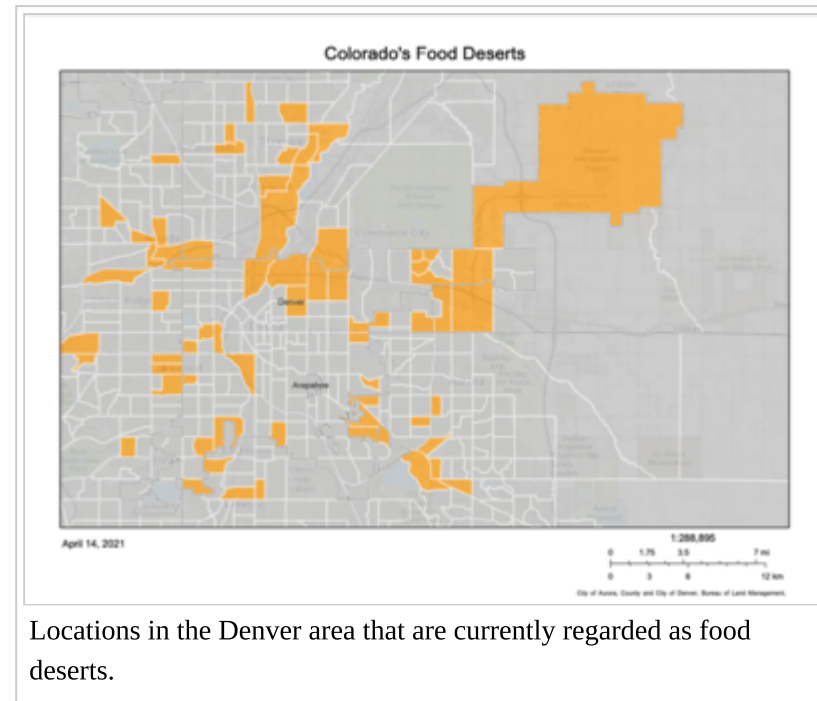
We are motivated by these consequences of food deserts, both prior to the pandemic and during it, to try to minimize the area considered to be in a food desert in our own community of Denver, Colorado.

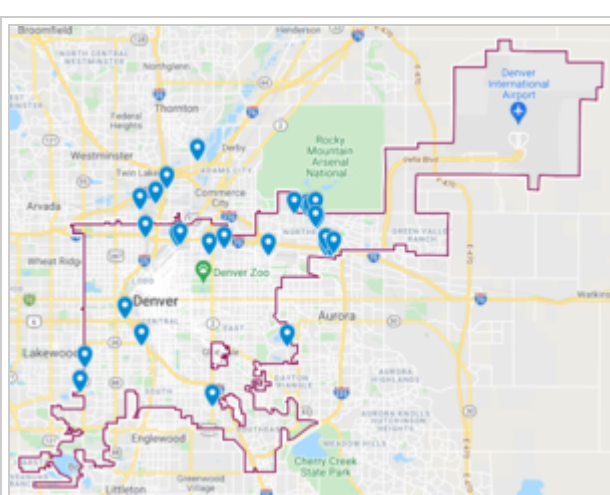
*\*The term food desert refers solely to the proximity to food providers. We recognize that this term does not consider other factors such as racism, people being time poor or cash poor, cost of living, etc. There are other terms, such as food oppression, that would be more accurate, but 'food desert' is the most widely used term, so we use it in this project.*

## Data

We choose to use census block groups, as opposed to census blocks or census tracts when determining population centers. This is because there are too many census blocks to run code efficiently, and census tracts cover too large of an area to be very meaningful in this context. Census block groups, however, provide a good middle ground. We use the centroid of each of the census block groups to determine the distance to the closest grocery store.

We are attempting to find locations where grocery stores could be built in order to minimize the number of locations that are considered to be in a food desert. To do this, we need several things. First, we need the current locations of grocery stores. Then, we need locations where a grocery store could be built at. Most grocery stores are between 20,000 and 60,000 square feet, so we compiled a list of warehouse spaces for sale or lease in the area that fit this square footage. These are the potential locations that we could build grocery stores at. In our data set for our programs, we list the current grocery stores and the potential locations in the same list, with the first 64 entries of the list being the current grocery store locations.





Locations where buildings large enough for a grocery store are located in relation to Denver county.

In the current grocery store distribution, just under a quarter of, or 115 out of 480, census block group centroids are more than one mile from the closest grocery store. We also recognize that this distance is "as the crow flies", and so is an underestimate of the actual distance to the closest grocery store. Because of this, it is likely that there are many more than just the 115 out of 480 census block centroids that are more than one mile from a grocery store.

# The First Program

The variables used for the first program are displayed below.

$R$  : Residential Areas (Here, they are census block groups)

$S$  : Stores

$p_r$  : Population of residential area  $r$

$d_{rs}$  : Distance from residential area  $r$  to store  $s$

$n$  : Number of stores

$y_{rs}$  : Indicator variable for if residential area  $r$  is assigned to store  $s$

$z_s$  : Indicator variable for if store  $s$  is used

In order to determine which store locations should be utilized, the first program minimizes the total (Manhattan) distance to the closest grocery store by each residential area multiplied by the population of that residential area. In other words, we are minimizing  $\sum_{r \in R} x_r p_r$ , where  $x_r$  is the distance to the closest grocery store, or

$$x_r = \sum_{s \in S} d_{rs} y_{rs}.$$

Our constraints are as follows:

$$(1) \quad x_r = \sum_{s \in S} d_{rs} y_{rs} \quad \forall r \in R$$

$$(2) \quad \sum_{s \in S} z_s \leq n$$

$$(3) \quad y_{rs} \leq z_s \quad \forall r \in R, s \in S$$

$$(4) \quad \sum_{s \in S} y_{rs} = 1 \quad \forall r \in R$$

$$(5) \quad \sum_{s=1}^{64} z_s = 64$$

$$(6) \quad z_s, y_{rs} \in \{0, 1\} \quad \forall r \in R, s \in S$$

Constraint (1) is the aforementioned shorthand for the distance traveled to the closest grocery store. Constraint (2) makes sure that we do not open more stores than we want to. Constraint (3) ensures that a residential area is only assigned to a store if it is open. Constraint (4) gives that every residential area gets assigned exactly one store. In order to make sure we use the current grocery stores we have, we use Constraint (5) to tell the program to utilize the first 64 stores. Lastly, Constraint (6) ensures integrality of the decision variables. To see which stores would be in the optimal distribution, we would look at the values for  $z_s$ .

For the second program, we utilize the Kolm-Pollak EDE, which is described in the next section.

## The Kolm-Pollak EDE and The Second Program

The Kolm-Pollak EDE (Equally Distributed Equivalent) is a measure of inequality that is used in urban planning to rank distributions. For the second program, instead of minimizing over population times distance, we minimize over the Kolm-Pollak EDE. Thus our objective function that is minimized is  $\min -\frac{1}{\kappa} \ln \left[ \frac{1}{|R|} \sum_{r \in R} e^{-\kappa x_r} \right]$ . This program utilizes all of the same variables as the first program, with the addition of  $\beta$ , which is the aversion parameter that represents how adverse a population is to inequality. Here, we use  $\beta < 0$  since a smaller distance is more favorable. The constraints for this second program are shown below.

$$(1) \ x_r = \sum_{s \in S} d_{rs} y_{rs} \quad \forall r \in R$$

$$(2) \ \sum_{s \in S} z_s \leq n$$

$$(3) \ y_{rs} \leq z_s \quad \forall r \in R, s \in S$$

$$(4) \ \sum_{s \in S} y_{rs} = 1 \quad \forall r \in R$$

$$(5) \ \kappa = \frac{\sum_{r \in R} x_r}{\sum_{r \in R} x_r^2} \beta$$

$$(6) \ \sum_{s=1}^{64} z_s = 64$$

$$(7) \ z_s, y_{rs} \in \{0, 1\} \quad \forall r \in R, s \in S$$

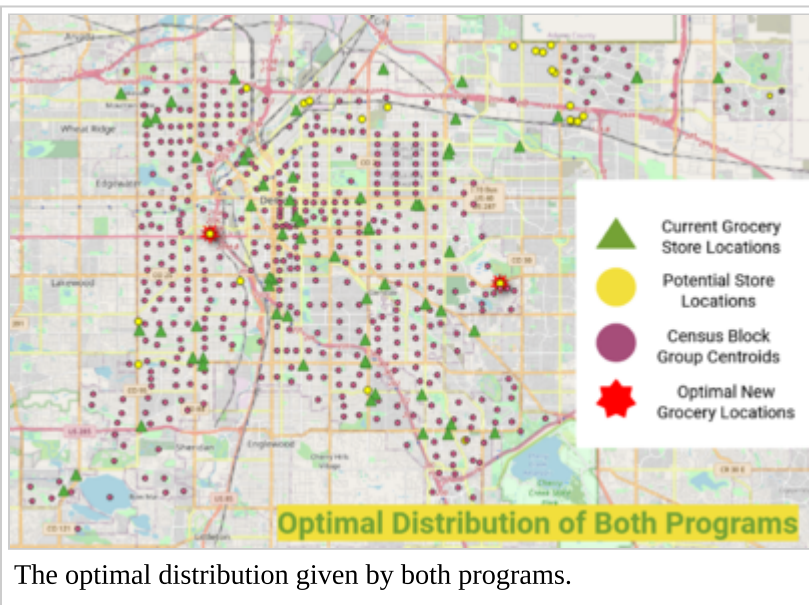
We can see that nearly all of the constraints are the same as the first program. However, Constraint (5) here does not appear in the first program. This constraint is an additional shorthand for including the aversion to inequality in the objective function. As in the first program, to see the optimal distribution, we would look at the values for  $z_s$ .

## Results

We asked both programs to produce an optimal distribution of grocery stores with opening a certain number. With our data set of potential grocery store locations, both programs gave the same optimal distribution for both opening one store and opening two stores. For opening one grocery store, the optimal location is at 9660 E Alameda Ave. If we were to open two grocery stores, the next optimal location is at 677 Alcott St. With these new added stores, 96 out of 480 of the census block group centroids would be more than 1 mile away from a grocery store, 19 less than the current distribution. Again this is "as the crow flies" distance. There could be several reasons why the two programs gave the same answer, but we suspect it is because there is a small set of potential grocery store locations to choose from. An example where the two programs gave different outputs is shown in the comparison section below.

## Comparison Between the Two Programs

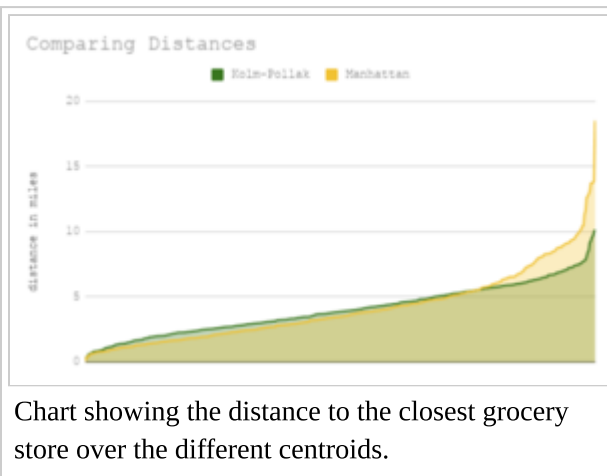
We ran both programs, instructing them to pick two stores for the entirety of Denver. The first program, which minimizes over the weighted Manhattan distance, chose the store located at 1331 N Speed Blvd and at 677 Alcott St. The second program, which minimizes over the Kolm-Pollak EDE, chose the store located



The optimal distribution given by both programs.



at 18605 Green Valley Ranch Blvd and at 825 S Colorado Blvd. Visualizations of these locations are shown on the right. We can see that the Kolm-Pollak EDE program picked a store that addresses those that would be worst off in the distribution picked by the weighted Manhattan program.



On the left is a chart depicting the distance to the closest grocery store (in the two store distribution discussed above) for each census block group centroid, organized from smallest to largest distance. We can see that the Kolm-Pollak EDE gives a 'flatter' curve, which indicates that it is addressing those that were worst off when using the weighted Manhattan program.

## Future Work and Policy Suggestions

In the future, we hope to extend this work to other cities, such as Houston and New Orleans, which have worse food access and where food insecurity disproportionately affects marginalized populations. We also hope to extend this to look at where other amenities, such as schools, parks, libraries, etc., should be placed. Eventually, we hope to

examine multiple amenities at once, for example bus stops and grocery stores would be a combination that would give valuable information for food deserts.

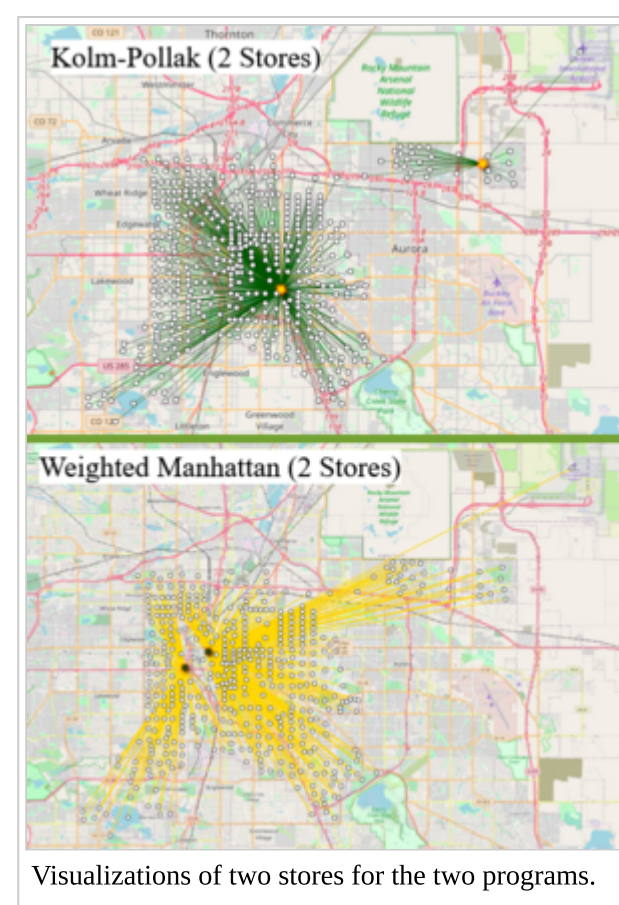
While the locations found through our programs are good locations to open grocery stores at, we can also use our programs for the distribution of food pantries or community fridges. Denver Community Fridges is a mutual aid program that is working to fight food insecurity in the Denver Metro area<sup>[5]</sup>. There are established community fridges in several locations already in Denver. These community fridges are stocked with fresh fruits and vegetables and also packaged meals. We can use our programs to find the optimal locations for additional Denver Community Fridges.

## Healthy Food Financing Initiative

The Healthy Food Financing Initiative (HFFI) is a partnership that provides grants and loans to finance construction and development of grocery stores in underserved areas. Between 2011 and 2015, it helped the development and support of over 1,000 grocery stores. Recently, it was reauthorized in the 2018 Farm Bill. Much of the Denver area is considered to be eligible (based off 2010 Census data), including the area where the location of the first store the two programs suggested to open.<sup>[6]</sup> With this, there could be grants and loans given to those that open a store in that location.

## Links to Code and Slides

Our AMPL Model and Data files, as well as our final slide deck, can be accessed through GitHub here: <https://github.com/rebrobin/HungryForEquality>



Visualizations of two stores for the two programs.

# References

1. ↑ [1] (<https://www.medicalnewstoday.com/articles/what-are-food-deserts#definition>)<https://www.medicalnewstoday.com/articles/what-are-food-deserts#definition>
2. ↑ [2] (<https://foodispower.org/access-health/food-deserts/>)<https://foodispower.org/access-health/food-deserts/>
3. ↑ [3] ([https://sites.duke.edu/lit290s-1\\_02\\_s2017/2017/03/04/health-and-socioeconomic-disparities-of-food-deserts/](https://sites.duke.edu/lit290s-1_02_s2017/2017/03/04/health-and-socioeconomic-disparities-of-food-deserts/))[https://sites.duke.edu/lit290s-1\\_02\\_s2017/2017/03/04/health-and-socioeconomic-disparities-of-food-deserts/](https://sites.duke.edu/lit290s-1_02_s2017/2017/03/04/health-and-socioeconomic-disparities-of-food-deserts/)
4. ↑ <sup>4.0</sup> <sup>4.1</sup> [4] (<https://www.canr.msu.edu/news/how-the-covid-19-pandemic-affects-food-deserts>), <https://www.canr.msu.edu/news/how-the-covid-19-pandemic-affects-food-deserts>
5. ↑ [5] (<https://www.denvercommunityfridge.com>)<https://www.denvercommunityfridge.com>
6. ↑ [6] (<https://www.investinginfood.com/eligibility/>)<https://www.investinginfood.com/eligibility/>

T.M. Logan, M.J. Anderson, T.G. Williams, L. Conrow, Measuring inequalities in urban systems: An approach for evaluating the distribution of amenities and burdens, Computers, Environment and Urban Systems, Volume 86, 2021, 101590, ISSN 0198-9715, <https://doi.org/10.1016/j.compenvurbsys.2020.101590>. Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Hungry\\_for\\_Equality:\\_Fighting\\_Food\\_Deserts&oldid=3334](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Hungry_for_Equality:_Fighting_Food_Deserts&oldid=3334)"

- This page was last modified on 5 May 2021, at 18:44.
- This page has been accessed 2,546 times.



# Identifying Gentrification

From CU Denver Optimization Student Wiki

This project is concerned with Identifying Gentrification. I will use structure age, income, and race data to inform a linear program to find out which areas are likely to be gentrified, and make suggestions on what to do with that

## Contents

- 1 Abstract
- 2 At-risk Neighborhoods
- 3 How can we help?
- 4 Linear Programming
- 5 Conclusions and Future Work
- 6 Links



A home in Five Points

## Abstract

In recent years, gentrification has become a major issue impacting low income families. This happens to neighborhoods which have older buildings, declining incomes, and businesses moving out. We can identify at-risk neighborhoods by evaluating these conditions with data from the Denver Open Data Catalog. Once the neighborhoods are identified, targeted policies can be informed with linear programming to help current residents continue to afford their homes.

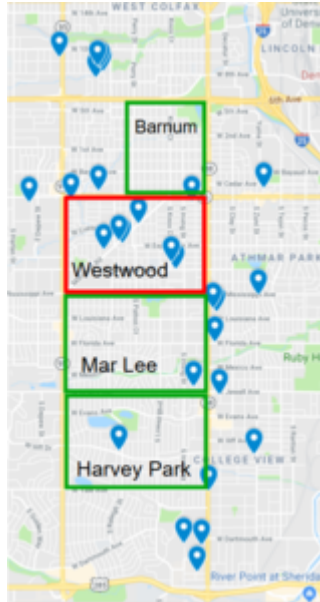
## At-risk Neighborhoods

Three things happen before gentrification takes place, income declines, businesses move out, and buildings grow older (no new development). The Denver Open Data Catalog has structure age, and income over a few years, organized by neighborhood. We'll consider these together and find neighborhoods that are potentially at risk. Below are three neighborhoods which stood out.

Neighborhood	Med. Struct. Age	Income growth
Barnum	79 years	−8.449%
Mar Lee	63 years	−4.950%
Harvey Park	61 years	−3.435%

# How can we help?

One of many ways to combat gentrification is to push for more affordable housing. If we map these neighborhoods, along with all nearby affordable housing, there actually seems to be something going on. In green are the at-risk neighborhoods, and the red square is Westwood.



Interestingly, has nearly identical income (not growth) to the surrounding neighborhoods, but seems to be facing nearly no risk of gentrification, with very different structure age and income growth statistics. This certainly suggests that the affordable housing projects have something to with the neighborhoods potential to be gentrified.

Neighborhood	Med. Struct. Age	Income growth
Barnum	79 years	−8.449%
Mar Lee	63 years	−4.950%
Harvey Park	61 years	−3.435%
Westwood	39 years	+10.656%

## Linear Programming

Our approach is to suggest areas which are far away from current housing projects in these areas as candidates for future housing projects. To this end, we'll maximize the distance to the nearest project (to make it as far as possible), and stay within the bounding box of the neighborhoods. The program is as follows:

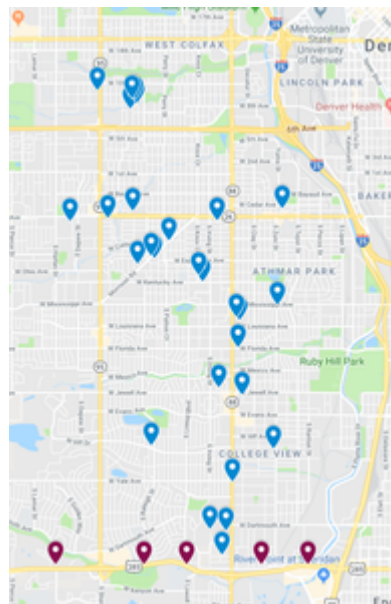
maximize  $\min_i d(x, y_i)$

subject to

$$-105.063312 \leq x_1 \leq -105.009367$$

$$39.653957 \leq x_2 \leq 39.732583$$

Clearly, this program is not linear, and there were quite a bit of struggles to get it to output something reasonable (as nonlinear programs are prone to do). Initial conditions were attempted, though the program seemed to always gravitate towards the southern border of the region. Below, in red, are five outputs of the program.



## Conclusions and Future Work

One thing that I would really like to see is a more comprehensive case study on the three at-risk neighborhoods and Westwood, to see what other potential differences there are in those neighborhoods that could be causing such a drastic difference. Another thing I would have liked to do with the program is consider property prices and the budget for affordable housing projects, something like this could have led to a much more interesting Linear Programming approach. Two other approaches I considered for maximizing the minimum distance were the "Largest Empty Circle" algorithm, and doing some clustering algorithm, finding the least dense cluster, and adding to it, though neither seemed to fit the Linear Program bill. One unfortunate thing about Denver County specifically is that we are running out of places to gentrify, currently there are more neighborhoods with median incomes exceeding 100,000\$ per year, than there are neighborhoods with less than 50,000\$ a year.

## Links

github (<http://github.com/dmtburke/d2p-2018>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Identifying\\_Gentrification&oldid=1897](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Identifying_Gentrification&oldid=1897)"



# Improving Denver Traffic

From CU Denver Optimization Student Wiki

## Links

GitHub Repository (<https://www.github.com/jakeat555/TrafficProject>)

## Abstract

In this project, Jacob Johns used linear regression techniques to determine the influence of one-way streets on traffic flow in Denver. Using this, a subset of streets in downtown Denver was selected, and a linear programming model was run to model the potential traffic flow if those streets were converted into one-ways.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Improving\\_Denver\\_Traffic&oldid=4418](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Improving_Denver_Traffic&oldid=4418)"

- 
- This page was last modified on 7 November 2023, at 11:43.
  - This page has been accessed 7 times.

# In-N-Out Of Kilter

From CU Denver Optimization Student Wiki

Welcome to Abigail Nix's project page! This project is on the Out of Kilter Algorithm, used to find a minimum cost flow of a network.

## Contents

- 1 Abstract
- 2 Introduction
- 3 Optimality Conditions
- 4 The Algorithm
  - 4.1 Kilter Numbers
  - 4.2 Pseudocode
  - 4.3 Correctness and Complexity
- 5 GitHub
- 6 References

## Abstract

In this project, I explored the Out-Of-Kilter Algorithm for solving a minimum cost flow problem. This algorithm works by successively augmenting flow along shortest paths in order to decrease the kilter numbers of each arc. At each iteration of the algorithm, the mass balance constraints are satisfied, and the algorithm incrementally increases and optimality. We begin by establishing the background information on optimality conditions and node potentials necessary to understand the algorithm. Then, in the next section, we introduce the algorithm itself, justify correctness, and analyze complexity.

## Introduction

The Out-of-Kilter algorithm works by satisfying the mass balance constraints of the network in every iteration, but flow bounds and optimality conditions do not need to be satisfied. However, in this project, we focus on a simplified version of the algorithm where flow bounds are respected, and we start with a feasible flow. Each iteration of the algorithm decreases the kilter number of arcs in the network by updating node potentials and augmenting flow along some shortest path, and the algorithm terminates when every arc is in-kilter. At this point, the flow found by the algorithm satisfies the complementary slackness optimality conditions, and is thus a minimum cost flow for the network. The description of the algorithm in this project is based on <sup>[1]</sup>, as well as a little from <sup>[2]</sup>.

Before explaining exactly how the algorithm works, we will explain a couple optimality conditions that can be used as termination criterion for algorithms solving minimum cost flow problems. Recall that for shortest path problems, we have the following optimality conditions:  $d(j) \leq d(i) + c_{ij}$  for every edge  $(i, j)$  in the network. When

Typesetting math: 100%

satisfied, we know that the distance labels  $d$  define true shortest path distances. We can define different optimality conditions for the minimum cost flow

problem instead of the shortest path problem.

## Optimality Conditions

The first optimality conditions we will introduce are the negative cycle optimality conditions. They are defined as follows:

**Theorem:** A feasible flow  $\mathbf{x}^*$  is optimal for a minimum cost flow problem if and only if the residual network  $G(\mathbf{x}^*)$  contains no negative cycle.

**Proof:** We will not explicitly prove these optimality conditions, but instead will give the proof idea. The first direction is proven by contrapositive. If  $G(\mathbf{x})$  contains a negative cycle, then flow can be augmented along this negative cycle and decrease the overall objective function value, so the original flow  $\mathbf{x}$  was not optimal. The other direction starts with a suboptimal flow, and decomposes the difference between this and an optimal flow into cycles. This then implies that the original flow is optimal.

For the next optimality conditions we will describe, we first establish a few definitions.

**Definition:** Let  $\pi(i)$ , the node potential for node  $i$ , be some real number associated with node  $i$ .

**Definition:** For a given set of node potentials  $\pi$ , we define the reduced cost of an arc  $(i, j)$  as  $c_{ij}^\pi = c_{ij} - \pi(i) + \pi(j)$ .

With reduced costs defined, we note a couple properties that we will use to prove the reduced cost optimality conditions. First, consider a directed path  $P$  from node  $k$  to node  $l$ . Then, the reduced cost of the whole path forms a telescoping sum, so we can rewrite it as

$$\sum_{(i,j) \in P} c_{ij}^\pi = \sum_{(i,j) \in P} (c_{ij} - \pi(i) + \pi(j)) = \left( \sum_{(i,j) \in P} c_{ij} \right) - \pi(k) + \pi(l).$$

For a directed cycle  $C$ , using the same argument, the reduced cost of the cycle becomes

$$\sum_{(i,j) \in C} c_{ij}^\pi = \sum_{(i,j) \in C} c_{ij}.$$

Using these properties, we can prove the following optimality conditions by showing that they are equivalent to the negative cycle optimality conditions.

**Theorem:** A feasible flow  $\mathbf{x}^*$  is optimal for a minimum cost flow problem if and only if there exists a set of node potentials  $\pi$  that satisfy  $c_{ij}^\pi \geq 0$  for every arc  $(i, j)$  in the residual network  $G(\mathbf{x}^*)$ .

**Proof:** Again, we only give the proof idea. For the first direction, we assume the negative cycle optimality conditions, and find shortest path distances  $d$  between node 1 and each node  $k$ . Then, by defining  $\pi = -d$ , the shortest path optimality conditions imply the reduced cost optimality conditions. For the other direction, assume the reduced cost optimality conditions, and let  $C$  be a directed cycle in the residual network. Then, since  $\sum_{(i,j) \in C} c_{ij}^\pi = \sum_{(i,j) \in C} c_{ij}$ , the negative cycle optimality conditions hold.

The third and final optimality conditions that we will explore can be proven using the two sets of conditions we have already introduced. While the negative cycle and reduced cost optimality conditions are good certificates to use to check if a solution is optimal, the Out-of-Kilter algorithm uses the complementary slackness conditions as its termination criteria. In particular, the complementary slackness conditions define whether an arc is "in-kilter" or "out-of-kilter" for a given residual network.

The complementary slackness conditions are as follows:

**Theorem:** A feasible flow  $x^*$  is optimal for a minimum cost flow problem if and only if there exists a set of node potentials  $\pi$  such that:

- If  $c_{ij}^\pi > 0$ , then  $x_{ij}^* = 0$ ,
- If  $0 < x_{ij}^* < u_{ij}$ , then  $c_{ij}^\pi = 0$ ,
- If  $c_{ij}^\pi < 0$ , then  $x_{ij}^* = u_{ij}$ .

**Proof:** For the first direction, suppose  $x^*$  satisfies the reduced cost optimality conditions, i.e.  $c_{ij}^\pi \geq 0$  for all  $(i, j)$  in  $G(x^*)$ . Then consider the following three cases.

- $c_{ij}^\pi > 0$ . In this case, suppose that arc  $(j, i)$  is in  $G(x^*)$ , i.e.,  $x_{ij}^* > 0$ . Then,

$$\begin{aligned} c_{ij}^\pi &= c_{ij} - \pi(i) + \pi(j) \\ &= -(-c_{ij} + \pi(i) - \pi(j)) \\ &= -(c_{ji} - \pi(j) + \pi(i)) = -c_{ji}^\pi \leq 0, \end{aligned}$$

since  $c_{ji}^\pi \geq 0$ . This is a contradiction, since  $c_{ij}^\pi > 0$ . Thus,  $x_{ij}^* = 0$ .

- $0 < x_{ij} < u_{ij}$ . Here, both arcs  $(i, j)$  and  $(j, i)$  are in  $G(x^*)$ . But, both  $c_{ij}^\pi \geq 0$  and  $c_{ji}^\pi = -c_{ij}^\pi \geq 0$ . This implies that  $c_{ij}^\pi = c_{ji}^\pi = 0$ .
- $c_{ij}^\pi < 0$ . Suppose arc  $(i, j) \in G(x^*)$ , i.e.,  $x_{ij}^* < u_{ij}$ . Then, for this arc, since  $c_{ij}^\pi < 0$ , the reduced cost optimality conditions do not hold. So,  $(i, j)$  cannot be in  $G(x^*)$ , so we must have  $x_{ij}^* = u_{ij}$ .

For the other direction of this proof, assume the complementary slackness conditions hold for  $(x, \pi)$ . Let  $(i, j) \in G(x)$  and suppose for contradiction that  $c_{ij}^\pi < 0$ . Then,  $x_{ij} = u_{ij}$ . However, this implies that  $(i, j) \notin G(x)$ , a contradiction. This finishes the proof.

In the next section, we will discuss the actual Out-of-Kilter algorithm, starting off by defining the kilter number of an arc using the complementary slackness optimality conditions.

## The Algorithm

As we have previously stated, the Out-of-Kilter algorithm iteratively augments flow in a network along shortest paths found in the current residual network. Each augmentation's goal is to decrease the kilter number of some arc, which increases both the feasibility and optimality of the new flow. Throughout the algorithm, intermediate flows may not satisfy the upper and lower arc flow bounds or the complementary slackness optimality conditions. However, at each step, mass balance constraints are

Typesetting math: 100%  
 laying and explaining pseudocode for the algorithm, we describe what a kilter number for an arc is.



# Kilter Numbers

Essentially, the kilter number of an arc measures how suboptimal that arc flow is. The complementary slackness optimality conditions are used to define the kilter number of an arc, and to form its kilter diagram, shown below.

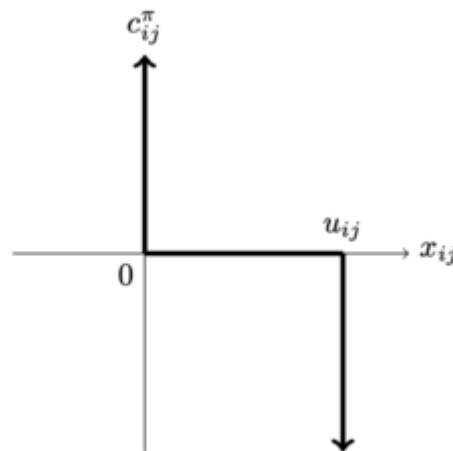


Figure 1: The kilter diagram for arc  $(i, j)$ .

The kilter diagram tells you which  $(x_{ij}, c_{ij}^\pi)$  pair corresponds to the arc being "in-kilter" or "out-of-kilter". A point that is on the thick line in the diagram corresponds to an in-kilter arc, i.e., it satisfies the complementary slackness optimality conditions, while a point anywhere else on the grid corresponds to an out-of-kilter arc. As a reminder, the complementary slackness optimality conditions are:

- If  $c_{ij}^\pi > 0$ , then  $x_{ij}^* = 0$ ,
- If  $0 < x_{ij}^* < u_{ij}$ , then  $c_{ij}^\pi = 0$ ,
- If  $c_{ij}^\pi < 0$ , then  $x_{ij}^* = u_{ij}$ .

From the kilter diagram, we can see each of these three conditions. The first condition corresponds to the leftmost line segment, where  $c_{ij}^\pi > 0$  and  $x_{ij} = 0$ . The second condition corresponds to the center line segment, where  $c_{ij}^\pi = 0$  and  $0 < x_{ij} < u_{ij}$ . Finally, the third condition corresponds to the rightmost line segment, where  $c_{ij}^\pi < 0$  and  $x_{ij} = u_{ij}$ . Using this diagram, we can define the kilter number of an arc.

**Definition:** The kilter number of an arc  $(i, j)$ , denoted  $k_{ij}$ , with current flow  $x_{ij}$  and reduced cost  $c_{ij}^\pi$ , is defined as how much  $x_{ij}$  needs to be changed without changing  $c_{ij}^\pi$  in order to make  $(i, j)$  in-kilter. In the diagram this is just how far the point would need to move in the  $x_{ij}$  direction in order to be on an in-kilter line segment.

Note that the kilter number can also be described based on where the point  $(x_{ij}, c_{ij}^\pi)$  lies:

- If  $c_{ij}^\pi > 0$ , then  $k_{ij} = |x_{ij}|$ .

Typesetting math: 100%  $u_{ij}$  and  $x_{ij} < 0$ , then  $k_{ij} = -x_{ij}$ . If  $c_{ij}^\pi = 0$  and  $x_{ij} > u_{ij}$ , then  $k_{ij} = x_{ij} - u_{ij}$ .

- If  $c_{ij}^\pi < 0$ , then  $k_{ij} = |u_{ij} - x_{ij}|$ .

Note we can also define the kilter number using residual capacities in  $G(x)$ ,  $r_{ij} = u_{ij} - x_{ij}$ . This is useful since the Out-of-Kilter algorithm works on residual networks. This definition is as follows:

$$k_{ij} = \begin{cases} 0 & \text{if } c_{ij}^\pi \geq 0 \\ r_{ij} & \text{if } c_{ij}^\pi < 0. \end{cases}$$

With the kilter numbers defined, we can describe the Out-of-Kilter algorithm.

## Pseudocode

The Out-of-Kilter algorithm starts with  $\pi(i) = 0$  for every node  $i$ , and a feasible flow, and then iteratively chooses an out-of-kilter arc  $(p, q)$  in the current residual network. In each iteration, the algorithm updates the node potentials  $\pi$  and augments flow along a shortest path from  $q$  to  $p$  if  $(p, q)$  is still an out-of-kilter arc ( $c_{pq}^{\pi'} < 0$ ). This process does not increase the kilter number of any arc, and strictly decreases the kilter number of  $(p, q)$ . Once all kilter numbers are zero, the current flow satisfies the complementary slackness optimality conditions, and thus, is optimal for the minimum cost flow problem. We provide pseudocode below.

---

**Algorithm 1** Out-Of-Kilter Algorithm for finding a Minimum Cost Flow

---

```
1: procedure OUTOFKILTER( $G$ )
2:    $\pi := 0$ 
3:   find feasible flow  $x$  ▷ using a Max Flow Algorithm
4:   define residual network  $G(x)$ 
5:   for  $(i, j) \in G(x)$  do
6:      $k_{ij} := \text{Kilter}((i, j))$ 
7:   end for
8:   while there exists  $(i, j) \in G(x)$  with  $k_{ij} > 0$  do ▷ there is an out-of-kilter arc
9:     select out-of-kilter arc  $(p, q) \in G(x)$ 
10:    define length of each arc  $(i, j) \in G(x)$  as  $\max\{0, c_{ij}^\pi\}$ 
11:    let  $d(i)$  denote the shortest path distance from  $q$  to  $i$  for each  $i \in G(x) - \{(q, p)\}$ 
12:    let  $P$  denote the shortest path from  $q$  to  $p$ 
13:     $\pi'(i) := \pi(i) - d(i)$  for each node  $i$  in  $G(x)$ 
14:    if  $c_{pq}^{\pi'} < 0$  then
15:       $W := P \cup \{(p, q)\}$ 
16:       $\delta := \min\{r_{ij} : (i, j) \in W\}$ 
17:      augment  $\delta$  units of flow along  $W$ 
18:      update  $x$  with new feasible flow, residual network  $G(x)$ , and reduced costs  $c_{ij}^{\pi'}$ 
19:    end if
20:  end while
21:  return  $x$  ▷  $x$  is the minimum cost flow
22: end procedure
```

---

## Correctness and Complexity

The correctness of the Out-of-Kilter algorithm relies on the following two lemmas about the two parts of the algorithm where the kilter number of some arc could change: when updating node potentials or when augmenting flow.

**Lemma:** Updating node potentials, i.e., setting  $\pi'(i) := \pi(i) - d(i)$  for all nodes  $i$ , does not increase the kilter number of any arc in  $G(x)$ .

**Proof:** Consistent with notation in the algorithm, let  $\pi$  be the node potentials before updating, and  $\pi'$  be the updated node potentials. Suppose for contradiction that this update increases the kilter number of some arc  $(i, j) \in G(x)$ . Since changing  $\pi$  does not change  $r_{ij}$ , the final definition of kilter number implies that the only way this can happen is if  $c_{ij}^\pi > 0$  and  $c_{ij}^{\pi'} < 0$ . Note that since  $c_{pq}^{\pi'} < 0$ , and  $c_{ij}^\pi \geq 0$ , we have  $(p, q) \neq (i, j)$ . Recall that in the algorithm, we defined  $d$  as shortest path distances,

with edge lengths  $\max\{0, c_{ij}^\pi\}$ . Therefore,  $d$  satisfies the shortest path optimality constraints, so

$$d(j) \leq d(i) + \max\{0, c_{ij}^\pi\} = d(i) + c_{ij}^\pi,$$

since  $c_{ij}^\pi \geq 0$ . Thus,

$$c_{ij}^{\pi'} = c_{ij} - (\pi(i) - d(i)) + (\pi(j) - d(j)) = c_{ij}^\pi + d(i) - d(j) \geq 0.$$

This is a contradiction, so updating node potentials does not increase the kilter number of any arc.

**Lemma:** Augmenting flow along  $W = P \cup \{(q, p)\}$  does not increase the kilter number of any arc in  $G(x)$ , and strictly decreases  $k_{pq}$ , the kilter number of arc  $(p, q)$ .

**Proof:** Since we only augment flow along  $W$ , the only arcs whose kilter numbers can change are those along this cycle (or their reversed arcs). Since  $P$  is a shortest path with arc lengths  $\max\{0, c_{ij}^\pi\}$ , then for  $(i, j) \in P$ ,

$$d(j) = d(i) + \max\{0, c_{ij}^\pi\} \geq d(i) + c_{ij}^\pi.$$

We defined  $\pi' = \pi - d$ , so we have

$$c_{ij}^{\pi'} = c_{ij} - (\pi(i) - d(i)) + (\pi(j) - d(j)) = c_{ij}^\pi + d(i) - d(j) \leq 0.$$

Now, since we augment flow along  $W$  by  $\delta = \min\{r_{ij} : (i, j) \in W\}$  units, the new flow does not go over the upper bound on any arc. Thus, if  $c_{ij}^{\pi'} = 0$ , the kilter number of arc  $(i, j)$  stays 0, and if  $c_{ij}^{\pi'} < 0$ , then  $k_{ij}$  will decrease because  $r_{ij}$  decreases when the flow on that arc  $(i, j)$  increases. Also note that augmenting flow along  $(i, j)$  may introduce the arc  $(j, i)$  into the new residual network, but since  $c_{ij}^{\pi'} \leq 0$ , then  $c_{ji}^{\pi'} \geq 0$ , meaning  $k_{ji} = 0$ , and  $(j, i)$  is in-kilter.

Finally, consider the arc  $(p, q)$ . Note that we only augment flow along this arc if  $c_{pq}^{\pi'} < 0$ . Since  $c_{pq}^{\pi'} < 0$ , we know that  $k_{pq} = r_{pq}$ . Thus, increasing the flow value on this arc strictly decreases its kilter number because it decreases the arc's residual capacity. Note that  $c_{qp}^{\pi'} = -c_{pq}^{\pi'} > 0$ , so  $(q, p)$  is an in-kilter arc.

Therefore, the flow augmentation step of the algorithm does not increase the kilter number of any arc, strictly decreases the kilter number of  $(p, q)$ , and only adds in-kilter arcs to the updated residual network.

With these two lemmas proven, we can show both correctness and finite termination of this algorithm. By our final definition of the kilter number, the maximum possible kilter number is  $U = \max\{u_{ij}\}$ . So, the sum of the kilter numbers at the start of the algorithm is at most  $mU$ , where  $m$  is the number of arcs in the network. As proven in the previous two lemmas, each iteration of the algorithm does not increase the kilter number of any arc, and strictly decreases the kilter number of one arc (by at least 1). Thus, the algorithm takes at most  $mU$  iterations. In each iteration, the main operation is solving a shortest path problem. Using Dijkstra's algorithm, this takes  $O(n^2)$ , but the running time depends on the algorithm used. So, the Out-of-Kilter algorithm takes at most  $O(mUn^2)$  time, i.e., runs in pseudopolynomial time.

Note that in this project, we focus on a simpler version of the Out-of-Kilter algorithm, where we start off with a feasible flow. However, this algorithm can start with a flow of  $x = 0$ , and in this case, feasibility is not maintained at each step. Here, we form the residual network arcs differently for arcs that have flow below the lower bound or above the upper bound. More information on this form of the algorithm can be found in <sup>[3]</sup>.

## GitHub

In the following GitHub link, both the slide deck for this presentation, as well as a LaTeX version of this write up, can be found.

<https://github.com/abigail-nix/In-N-Out-Of-Kilter>

## References

1. ↑ Ravinda K. Ahuja, Thomas L. Magnanti, James B. Orlin. Network Flows. 1993.
  2. ↑ Durbin, E. P. and David Kroenke. The Out-of-Kilter Algorithm: A Primer. 1967. RAND Corporation.
  3. ↑ Ravinda K. Ahuja, Thomas L. Magnanti, James B. Orlin. Network Flows. 1993.
- Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=In-N-Out\_Of\_Kilter&oldid=4823"

- 
- This page was last modified on 2 May 2024, at 11:08.
  - This page has been accessed 62 times.

# Increasing Network Robustness

From CU Denver Optimization Student Wiki

Network Robustness is a measurement of how fault tolerant a network is. Research in this area focuses on making a network more tolerant to perturbations. It is a very active area of research mainly focusing on making network more robust against various failure patterns. The patterns come from removing nodes or arcs and when the removal is targeted it starts being classified as an attack. This project is focused on cascading failures, which is when a single failure causes other nodes and arcs to fail which means that no matter how small the initial issue was, the damage is magnified by the fact this single failure of a node or arc affected more than just the one arc or node in the network.

## Contents

- 1 Problem Set Up
- 2 Evaluating Robustness
- 3 Methods
  - 3.1 High Betweenness Linking Strategy
  - 3.2 Low Polarization Linking Strategy
  - 3.3 Real World Issues
- 4 Wrapping Up
- 5 References

## Problem Set Up

The motivation for this is installing Sensors on a network. There are two problems that need to be solved here. The first is how the sensors are deployed, and this will not be focused on here. The other is making the systems of sensors fault tolerant.

Deploying the sensors effectively becomes picking a starting point and intelligently growing the network of sensors from there that can take advantage of position of location on the system to validate sensor input and correct configuration. While interesting this becomes finding a minimum spanning tree and is covered extensively elsewhere on this wiki.

Since they are deployed with the network hardware if a sensor fails or no longer functions, it is difficult to access it quickly and replace the dead sensor. The robustness of the sensor coverage becomes paramount to avoid any gaps in coverage or blind spots on the network in the event of a sensor failure. Failure of the network being monitored is a real possibility but outside the scope of this project, and we instead focus on ensuring enough redundancy that a single sensor failing will not create blind spots in our coverage while the hardware is being fixed.

Network details will not be included in this project since it is not important to the problem itself, so we will be using a simplified network to illustrate the problem. In the illustration each node represents some grouping of users or machines. We assume that each arc has a high capacity and the number of arcs connected to a note can represent the is to the network and can represent how much traffic that node can generate.

# Evaluating Robustness

With the basic minimum spanning tree there is just a single sensor covering each node, so the traffic between each node has two sensors covering it, but any traffic internal to the node is lost and the validation of the data for each sensor's nput is also gone for data between those paths. This sort of failure drives how we can compute the robustness of a network. This is called the Critical Node Method<sup>[1]</sup>. Here the critical node method serves as a way to make sure that any arc added to to produce redundancy in our network actually makes our network harder to disrupt.

This can be treated as a linear program formulated in the following way

$$\min \sum_{C_i} \binom{|C_i|}{2}$$

The equation is attempting to minimize the size of the connected components that remain after some number of nodes are removed from the network. Solving this when the network has added arcs and comparing the solution shows if the arc addition matters because if the solution has changed then the arc affected the robustness of the network. Each arc probably will not require additional nodes to be removed, but changing which nodes are removed show that the arc changed the structure of the network. Though as the network gets denser it is suspected that the number of nodes that have to fail to break the graph will also increase, which was not a focus of this project, but could warrant further study.

## Methods

The most common focus that was found reviewing the literature for cascading failures was a focus on betweenenss<sup>[2]</sup>. Betweeneness is a measurement of much traffic passes between two nodes. If you recall our basic map above Each nodes was adjacent to the nodes it talked to more frequently is an assumption, but it is reasonable to expect that each node communicates with every other node to some degree.

Betweeneness itself is computed by determining how many shortest paths a node is involved in. the formula looks something like this for some node v

$$b(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

## High Betweenness Linking Strategy

The first method that really uses this method is called the high betweenness linking strategy<sup>[3]</sup>, and the method is to just go through pairs of arcs and compute the traffic between them, and if they are not already adjacent in the network connect the two of them with an arc. For our problem this means that the two sensors will be set up to receive the traffic from both nodes as a primary and a back up so in case the primary fails the traffic will go to the back up and still be analyzed by the system and this heavily paired traffic will be analyzed by a single system.



Our representative network

This methodology is very good at reinforcing more heavily used components in a network, however it can unintentionally isolate labs and team specific infrastructure to a small subset of heavily connected components which could lead to very underutilized hardware depending on team activity and usage patterns.

The reinforcing of specific labs with themselves does not seem like a bad thing, but some labs are so isolated or so specialized that most of their traffic can be ignored for sensor usage so the sensors will be very underutilized.

## Low Polarization Linking Strategy

The low polarization linking strategy attempts to homogenize the network<sup>[4]</sup>. This makes a great deal of sense from a network load perspective because if a high traffic node goes down its back ups will be hit by all that traffic and it is possible for back ups to be overwhelmed by the traffic they were already handling when combined with the new network data from the failed node which leads to a cascading failure because the new traffic amount will definitely cause the next back up node to fail.

However, if the sensor network is homogenized it means that each sensor handles roughly the same amount of traffic, so all primaries and back up settings equal the same values. The way we can homogenize the network is to first compute the polarity of a network by taking the maximum betweenness of the network and the average betweenness of the network to compute the polarity and then balance the traffic between sensors to get the polarity value as low as possible.

Polarization of a network is also relatively easy to compute with the formula being

$$p = \frac{b_{max} - b_{avg}}{b_{avg}}$$

This may sound like the ideal case, but it does run into some real world issues, the network we are placing sensors on is not very uniform in the betweenness of the nodes which means that it will be hard to balance traffic between the various sensors. Next, traffic patterns change over time to reflect funding and growth. This sensor rollout is actually part of some future proofing upgrades, so eventually the traffic of the system will exceed how much traffic the sensors can properly handle.

## Real World Issues

After looking at these methods betweenness appears to be a great way to balance a network effectively, even with the drawbacks pointed out about both algorithms. However, an assumption made in both of these is that the person running the analysis on the network owns all the nodes. This causes an issue for the sensor rollout because a large portion of the network traffic leaves the owned nodes through a small subset of nodes on the network this skews out betweenness values since all network traffic with a destination outside the owned network goes through these nodes and also means that a small subset of nodes see a large percentage of the traffic.

Currently the sensors have a great deal of capacity to keep pace with traffic within the nodes and between nodes, but while a network can be expanded relatively easily by adding more hardware each sensor has a very finite capacity and once it is exceeded the sensor falls behind during periods of high traffic and has to clear its buffer slowing down analysis. This state of affairs is tolerable in bursts, but being permanently behind is not good. Especially since filling the buffer could cause issues and cause a sensor outage on its own. This will fall under issues to be future proofed against, but it is reasonable to expect that sensor capacity will go up over time much like how network capacity will go up.



# Wrapping Up

Network robustness is still an active area of research, currently there is no best way to solve these types of problems and real world applications do not always line up with the assumptions necessary to make these algorithms work. Similar to irrational capacities in a network flow operate the problem is avoidable from a practical standpoint, but is an important theoretical construct. Both of the methods discussed can provide reasonable solutions that will operate well, but both have issues with the longevity of the solutions provided and struggle with the network set up that exists currently with how to measure betweenness on the controlled network.

With those limitations in mind these algorithms can help to develop deployment and installation guidelines the help with set up and maintenance patterns that can help with the problem that this paper started with. Time just has to be budgeted and people with relevant experience should provide feedback to ensure that any solution that these algorithms reach is relevant and meets the needs of the teams involved.

Finally, the analysis being performed would also be improved over time, so the network should be reanalyzed on a somewhat regular schedule to make sure that the solutions that were found are still reasonable with the growth and changes the network experienced, but time should also be spent doing another literature search to see if better tools have been developed to tackle this problem.

## References

1. ↑ [1] (<https://computationalsocialnetworks.springeropen.com/articles/10.1186/s40649-015-0010-y>) Mario Ventresca and Dionne Aleman "Efficiently identifying critical nodes in large complex networks"
2. ↑ [2] ([https://en.wikipedia.org/wiki/Betweenness\\_centrality](https://en.wikipedia.org/wiki/Betweenness_centrality)) Betweenness Centrality
3. ↑ [3] (<https://www.sciencedirect.com/science/article/pii/S0960077913001604>) Xian-Bin Cao,Chen Hong,Wen-Bo Du,Jun Zhang "Improving the network robustness against cascading by adding links"
4. ↑ [4] (<https://www.sciencedirect.com/science/article/pii/S0960077913001604>) Xian-Bin Cao,Chen Hong,Wen-Bo Du,Jun Zhang "Improving the network robustness against cascading by adding links"

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Increasing\\_Network\\_Robustness&oldid=1424](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Increasing_Network_Robustness&oldid=1424)"

- This page was last modified on 30 April 2018, at 12:53.
- This page has been accessed 11,292 times.

# Irrational Input to Integer Programs

From CU Denver Optimization Student Wiki

Let  $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ . The Fundamental Theorem of Integer Programming requires rational matrices  $A$  and  $b$  in the formulation of the polyhedron  $P$  in order to guarantee that if  $S = P \cap \{\mathbb{Z}^p \times \mathbb{R}^{n-p}\}$ ,  $\text{conv}(S)$  will be a polyhedron. Suppose this requirement is relaxed, such as in the following set of inequalities.

$$\begin{aligned} -\sqrt{2}x_1 + x_2 &\leq 0 \\ x_1 &\geq 1 \\ x_2 &\geq 0 \end{aligned}$$

The integer hull for this set of inequalities is not a polyhedron. To see this, first note that the first equation gives

$$x_2 \leq \sqrt{2}x_1.$$

Any integer value of  $x_2$  will therefore satisfy  $x_2 \leq \lfloor \sqrt{2}x_1 \rfloor$ , so the inequality can be rewritten in terms of  $x_1$  only.

$$-\sqrt{2}x_1 + \lfloor \sqrt{2}x_1 \rfloor \leq 0.$$

The first number in this sum must be an irrational number; noting that  $x_1 = 0$  is excluded by another inequality,  $\sqrt{2}$ , times a positive integer,  $x_1$ , results in an irrational number. Therefore the sum with another integer can never equal zero exactly, making the inequality strict. This inequality is the primary cause of the integer hull not being polyhedral, as we will see.

$$-\sqrt{2}x_1 + \lfloor \sqrt{2}x_1 \rfloor < 0$$

## Contents

- 1 Forming the Integer Hull
- 2 Optimization
- 3 Rational Requirement of  $b$
- 4 Matlab Code
  - 4.1 Sources

# Forming the Integer Hull

When forming the integer hull, it is only the top face of the original  $P$  that needs adjusted. For this specific polyhedron, we can work from the left along the  $x_1$  axis. For each  $x_1$ , we need only consider the integer point with the largest  $x_2$  value. We can form a valid inequality by connecting the point with the current maximum  $(x_1, x_2)$  to the next integer point to its right that is closer to the strict inequality. So for the first step, we do not connect (1,1) to (2, 3), but instead to (3,4). Since the slope between (1,1) and (3,4) is greater than  $\sqrt{2}$ , this is guaranteed not to cut off any other integer points, and will also exit  $P$  where further inequalities are required.

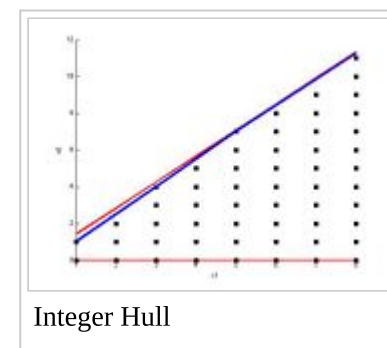
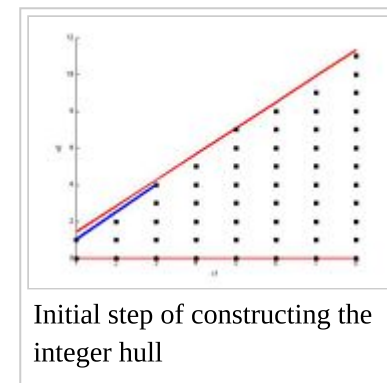
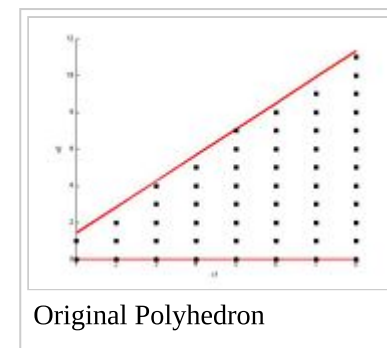
This process resumes from the right point on the segment, (3,4), which we can see connects to (5,7). There is not a specific pattern to detect where the next connection will occur; However, by a density argument, there will always be another integer point that is closer to the inequality than the current point. This leads to an infinite number of inequalities defining the upper boundary of this integer hull, and the conclusion that the integer hull is not polyhedral, as any polyhedron can be finitely generated.

The first ten vertices on these segments are:

$x_1$	$x_2$	distance
1	1	0.414213562373
3	4	0.242640687119
5	7	0.071067811865
17	24	0.041630560343
29	41	0.012193308820
99	140	0.007142674936
169	239	0.002092041053
577	816	0.001225489276
985	1393	0.000358937499
3363	4756	0.000210260719

## Optimization

Attempting to optimize over this can lead to no optimal value even when bounded in the optimal direction. For instance, allow the inequality to directly translate to the objective:



$$\text{maximize } -\sqrt{2}x_1 + x_2$$

subject to:

$$\begin{aligned} -\sqrt{2}x_1 + x_2 &\leq 0 \\ x_1 &\geq 1 \\ x_2 &\geq 0 \end{aligned}$$

With the only integer solution to  $-\sqrt{2}x_1 + x_2 \leq 0$  excluded by the other inequalities, there is no optimal objective value.

## Rational Requirement of $b$

For Mixed Integer Linear Programs, similar issues happen when  $b$  is allowed to be irrational. But in the case of Pure Integer Programs, only  $A$  is required to be rational. To see this, let  $S = P \cap \mathbb{Z}^n$ . Scale  $Ax \leq b$  so that  $A$  is integer. Choose a row  $a_i^T x \leq b_i$  where  $b_i$  is irrational. Then for  $x \in S$ ,  $a_i^T x$  is integer, so  $a_i^T x \leq \lfloor b_i \rfloor$  is a valid inequality for  $\text{conv}(S)$ . Repeat for all irrational  $b_i$ . (Essentially, just round all right hand sides).

## Matlab Code

```
%This code generates figures and tables for the polyhedron
%-1*sqrt(2)*x_1+x_2 <= 0, x_1 >= 1, x_2 >= 0, examining the integer hull

clear all;
close all;

%Adjustable input
x = [0:8];%will need to adjust axis in figures if changing this
maxnum = 10;%How many vertices to find along the upper side of the polyhedron
%maxnum-1 is the number of segments drawn in the final figure

%Declarations
n = length(x);
A = zeros(n^2,2);%This will store all integer points
%n^2 is not true size, overshoots (55 vs. 64).
%Will not access extra 0's
%Actual number of integer points is n + sum_j=1^n floor(sqrt(2)*j)
ymax = zeros(maxnum,3);
currentrow = 1;
maxfound = 1;

for i = 2:length(x)
    ytemp = 0;
    while ytemp < sqrt(2)*x(i) %generate all integer points, roughly 0(n^2)
        A(currentrow,1) = x(i);
        A(currentrow,2) = ytemp;
        ytemp = ytemp+1;
        currentrow = currentrow+1;
    end
```

```

%ymax(maxnum,2) = A(currentrow-1,2);
%ymax(maxnum,1) = A(currentrow-1,1);
%ymax(maxnum,3) = sqrt(2)*x(i)-A(currentrow-1,2);
%
%else
%    if (sqrt(2)*x(i)-A(currentrow-1,2) < ymax(maxnum,3))
%        maxnum = maxnum+1;
%        ymax(maxnum,2) = A(currentrow-1,2);
%        ymax(maxnum,3) = sqrt(2)*x(i)-A(currentrow-1,2);
%        ymax(maxnum,1) = A(currentrow-1,1);
%    end
%end
end
end
%size(A)
%ymax

%Separate way to find vertices
tempx = 2;
olddist = sqrt(2) - floor(sqrt(2));
ymax(1,1) = 1;
ymax(1,2) = floor(sqrt(2));
ymax(1,3) = olddist;
while maxfound < maxnum
    newdist = sqrt(2)*tempx - floor(sqrt(2)*tempx);
    if newdist < olddist
        maxfound = maxfound+1;
        ymax(maxfound,1) = tempx;
        ymax(maxfound,2) = floor(sqrt(2)*tempx);
        ymax(maxfound,3) = newdist;
        olddist = newdist;
    end
    tempx = tempx+1;
end
disp(ymax);
%break;%If you don't want the figures

figure(1)
hold on;
plot(x,sqrt(2)*x, 'r', 'LineWidth', 3)
plot([1 1], [0 sqrt(2)], 'r', 'LineWidth', 2);
plot([1 8], [0 0], 'r', 'LineWidth', 2);
axis([1 8 0 ceil(8*sqrt(2))]);
plot(A(1:currentrow-1,1), A(1:currentrow-1,2), 'ks','markerfacecolor','k')
xlabel('x1');
ylabel('x2');
hold off;

for i = 2:maxnum

figure(i)
hold on;
plot(x,sqrt(2)*x, 'r', 'LineWidth', 3)
plot([1 1], [0 sqrt(2)], 'r', 'LineWidth', 2);
plot([1 8], [0 0], 'r', 'LineWidth', 2);
axis([1 8 0 ceil(8*sqrt(2))]);
plot(A(1:currentrow-1,1), A(1:currentrow-1,2), 'ks','markerfacecolor','k')
plot(ymax(1:i,1), ymax(1:i,2), 'LineWidth', 4);
xlabel('x1');
ylabel('x2');

```

```

        fignum = i+1;
        break;
    end
end

figure(fignum)
hold on;
%plot(x,sqrt(2)*x, 'r', 'LineWidth', 3)
%plot([1 1], [0 sqrt(2)], 'r', 'LineWidth', 2);
%plot([1 ymax(maxnum,1)], [0 0], 'r', 'LineWidth', 2);
axis([1 ymax(maxnum,1) 0 ceil(ymax(maxnum,1)*sqrt(2))]);
%plot(A(1:currentrow-1,1), A(1:currentrow-1,2), 'ks','markerfacecolor','k')
plot(ymax(:,1), ymax(:,2), 'LineWidth', 2);
xlabel('x1');
ylabel('x2');
plot(ymax(:,1), ymax(:,2), 'ks','markerfacecolor','k' );

```

## Sources

Sven O. Krumke. *Integer Programming: Polyhedra and Algorithms*. Technische Universität Kaiserslautern. November 16, 2016. Integer Programming Lecture Notes ([http://www.mathematik.uni-kl.de/fileadmin/AGs/opt/Lehre/WS1617/ip\\_ws1617/IP1-3.pdf](http://www.mathematik.uni-kl.de/fileadmin/AGs/opt/Lehre/WS1617/ip_ws1617/IP1-3.pdf))  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Irrational\\_Input\\_to\\_Integer\\_Programs&oldid=539](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Irrational_Input_to_Integer_Programs&oldid=539)"

- This page was last modified on 3 May 2017, at 12:39.
- This page has been accessed 8,586 times.

# Jacob Dunham

From CU Denver Optimization Student Wiki

## Contents

- 1 Contact Links
- 2 Early Life and Background
  - 2.1 Education
- 3 Professional Life
  - 3.1 Career
  - 3.2 Programming Languages/Experience
  - 3.3 Publications
  - 3.4 Projects
- 4 Github repositories



self portrait

## Contact Links

- Email (<mailto:jaocb.2.dunham@ucdenver.edu>)

## Early Life and Background

Jacob Dunham was born in Bakersfield, California.

## Education

1. California State University Bakersfield, BS applied mathematics 2017
2. University of South Alabama, MS applied mathematics 2020

# Professional Life

## Career

- Triage Specialist - TuSimple(2020-2022)

## Programming Languages/Experience

- Python
- SageMath
- C++
- SQL
- JQL

## Publications

Metrizability of Mahavier Products Indexed by Partial Orders (<http://topology.nipissingu.ca/tp/reprints/v62/tp62002p1.pdf>)

## Projects

Clustering Neighborhoods in Order to Analyze Policy Needs  
([http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Clustering\\_Neighborhoods\\_in\\_Order\\_to\\_Analyze\\_Policy\\_Needs](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Clustering_Neighborhoods_in_Order_to_Analyze_Policy_Needs))

Preflow-Push Algorithm ([https://math.ucdenver.edu/~sborgwardt/wiki/index.php/Preflow-Push\\_Algorithm](https://math.ucdenver.edu/~sborgwardt/wiki/index.php/Preflow-Push_Algorithm))

## Github repositories

Preflow-Push Project Repository (<https://github.com/JacobDun/Preflow-Push-alg>)  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Jacob\\_Dunham&oldid=4637](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Jacob_Dunham&oldid=4637)"  
Category: Contributors

- 
- This page was last modified on 15 April 2024, at 14:47.
  - This page has been accessed 92 times.



# Jacob Johns

From CU Denver Optimization Student Wiki

## Contents

- 1 Contact Links
- 2 Early Life and Background
  - 2.1 Education
- 3 Professional Life
  - 3.1 Career
  - 3.2 Programming Languages/Experience
  - 3.3 Projects

## Contact Links

- Email (<mailto:jacob.johns@ucdenver.edu>)
- LinkedIn (<https://www.linkedin.com/in/jacob-j-6a2a12136/>)
- GitHub (<https://github.com/jakeat555>)

## Early Life and Background

Jacob Johns was born in Denver, Colorado. He is the 4th of 6 children. At the age of 13 he took 126th place in the 2013 Colorado State MathCounts competition. After relegation, he learned how to solve a Dodecahedron Rubik's Cube; a skill he no longer maintains.

## Education

1. Utah State University, B.S. in Computational Mathematics with a minor in Psychology

## Professional Life

### Career

- Office Cleaner - Weatherstone Captial Mangement



Jacob Johns at 23

- Swim Instructor Aid - Westminster Recreation Center
- Tile Installer - Interior Resource Group
- Facility Attendant - Thornton Recreation Center
- Dive Instructor - Logan Recreation Center
- Shelf Stocker - Walmart
- College Algebra Grader - Utah State University
- Window Well Technician - Mountainland Covers
- Barista - USU Dining Services
- Sales Associate - DEFY
- Missionary - Church of Jesus Christ of Latter-Day Saints
- IT Intern - Schreiber
- Graduate Teaching Assistant - University of Colorado Denver

## Programming Languages/Experience

- Java
- C++
- C
- Matlab
- Python
- R

## Projects

In the fall of 2020 for Linear Programming, he worked on One-way: To Improve Denver Traffic. In this project, we used linear regression techniques to determine the influence of one-way streets on traffic flow in Denver. Using this, a subset of streets in downtown Denver was selected, and a linear programming model was run to model the potential traffic flow if those streets were converted into one-ways.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Jacob\\_Johns&oldid=4449](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Jacob_Johns&oldid=4449)"

Category: Contributors

- 
- This page was last modified on 13 November 2023, at 13:11.
  - This page has been accessed 51 times.

# Jacob's page

From CU Denver Optimization Student Wiki

Jacob Dunham

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Jacob%27s\\_page&oldid=4385](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Jacob%27s_page&oldid=4385)"

- 
- This page was last modified on 7 November 2023, at 01:36.
  - This page has been accessed 5 times.

# John McFarlane

From CU Denver Optimization Student Wiki

I am a Electrical Engineering Master's student specializing in Communications at CU Denver. I am taking linear programming because it applies directly to optimization problems in communications. My project partner is Sajjad Nassirpour. Our project this semester is based on an IEEE-published paper entitled "A New Wireless Multicast Queuing Design Using Network Coding and Data-Flow Model" by Nadieh Moghadam and Hongxiang Li.

My contribution: Queue-Based Strategy to Achieve Maximum Stable Rate in Multi-user Network

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=John\\_McFarlane&oldid=2214](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=John_McFarlane&oldid=2214)"

Category: Contributors

- 
- This page was last modified on 15 November 2019, at 11:58.
  - This page has been accessed 5,745 times.

# Johnathan Rhyne

From CU Denver Optimization Student Wiki

## Contents

- 1 Contact Links
- 2 About Me
- 3 Programming Languages
- 4 Education
- 5 Projects

## Contact Links

- Email (<mailto:johnathan.rhyne@ucdenver.edu>)
- Github (<https://github.com/jprhyne>)

## About Me

Hey y'all, I am Johnathan Rhyne, and I am pursuing a PhD in Applied Mathematics here at CU Denver. My research interests are in Numerical Linear Algebra and general Math Software.

My personal interests are in open source software, solving puzzles, and general tinkering with computers.

## Programming Languages

1. Java
2. C/C++
3. Python

## Education

1. NC State, B.S. In Mathematics with minors in Computer Programming, Statistics, and Political Science

# Projects

1. In Spring of 2023, I worked on a project for using AMPL from inside C as a part of the Integer Programming course Embedding AMPL In C
2. In Spring of 2023, I worked on a project showing the equitability of fire hydrant distribution in the city of Denver and some surrounding areas  
Equitable\_Fire\_Hydrant\_Placement

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Johnathan\_Rhyne&oldid=4302"

Category: Contributors

- 
- This page was last modified on 16 April 2023, at 10:48.
  - This page has been accessed 79 times.

# Jordan Perr-Sauer

From CU Denver Optimization Student Wiki



Jordan Perr-Sauer graduated with a bachelors in Mathematics, magna cum laude - with distinction, from the University of Colorado, Denver in 2017. Jordan completed his honors project and directed research with professor Steffen Borgwardt on the topic of elementary cellular automata. He also installed, managed, and provided user support for this wiki.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Jordan\\_Perr-Sauer&oldid=550](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Jordan_Perr-Sauer&oldid=550)"

Category: Contributors

- 
- This page was last modified on 1 August 2017, at 17:19.
  - This page has been accessed 3,815 times.

# Kathleen Gatliffe

From CU Denver Optimization Student Wiki

Kathleen Gatliffe is a graduate student at the University of Colorado Denver. She holds bachelor's degrees in both electrical and mechanical engineering from CU Denver as well. She is passionate about mathematical communication and exploring social issues through statistics. Her master's project will be in service to the CU Denver learning assistants program.

She currently works as a mathematics and physics tutor for the Community College of Denver. In her free time she is an accomplished enamelist and metalsmith and is known for her skill as a cat whisperer.

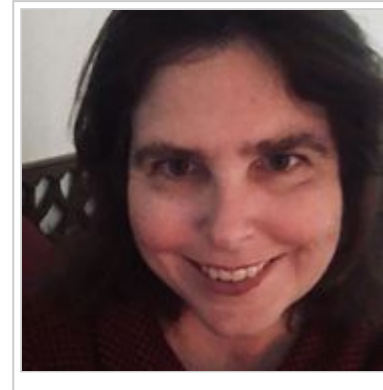
She is scheduled to teach MATH 2411-004, Calculus II, in the spring.

## Contributions

In Fall 2018, Kathleen contributed to the wiki with her project for MATH 5593: Linear Programming, Location Fluctuations in Denver Area Bias Motivated Crime Explored Through Linear Programming. In addition, she created two general interest pages documenting interesting aspects of her project, Data Visualization Using QGIS and Denver Government Coordinate Systems.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Kathleen\\_Gatliffe&oldid=1940](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Kathleen_Gatliffe&oldid=1940)"

Category: Contributors



- 
- This page was last modified on 18 December 2018, at 16:05.
  - This page has been accessed 2,817 times.



# Keegan Schmitt

From CU Denver Optimization Student Wiki

Keegan Schmitt is an undergraduate student at the University of Colorado Denver. He has studied business and engineering and holds an associates in business from Arapahoe Community College. He was working on a bachelors in mechanical engineering with a focus in motorsports engineering when he fell in love with pure and applied mathematics. He is now pursuing a bachelors degree in applied mathematics. He tutors all levels of mathematics from middle school pre-algebra to college calculus and is passionate about furthering mathematics and science education in any way he can. When he isn't staring at a whiteboard or typesetting a document in LaTeX, he can be found trail running in the Rocky Mountains or at any local gym picking up heavy things and then putting them back down. His favorite color is orange and favorite variable is  $\lambda$  because he thinks it is fun to say.

Here are Keegan's contributions to the Optimization Wiki:

- Linear Programming

- Duality: Bounding the Primal

- Duality: Economic Example

- Shadow price

- Complementary slackness

- Programming Files

- Computing Using Mathematical Programs

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Keegan\_Schmitt&oldid=280"

Category: Contributors

- 
- This page was last modified on 30 March 2017, at 15:56.
  - This page has been accessed 6,302 times.

# Kevin Bloom

From CU Denver Optimization Student Wiki

After growing up in Houston, TX, Kevin earned his Bachelor of Science degree in Applied and Computational Mathematics from the University of Southern California in Los Angeles, CA. After graduating, he continued his mathematics education by completing 3 Society of Actuary Exams and fulfilled the Validation by Educational Experience requirements in Economics, Corporate Finance, Time Series, and Regression Analysis while owning and operating a music production company.

After getting burnt out on the Los Angeles music scene, he relocated to Denver and earned an Alternative License to teach high school mathematics. Most recently, he began studying for a Master of Science degree in Applied Mathematics with a concentration in Operations Research from University of Colorado, Denver.

Kevin created the A Multi-Objective Linear Program for Nutrition wiki page.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Kevin\\_Bloom&oldid=919](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Kevin_Bloom&oldid=919)"

Category: Contributors

- 
- This page was last modified on 7 December 2017, at 08:54.
  - This page has been accessed 2,571 times.

# Knapsack Problem Algorithms

From CU Denver Optimization Student Wiki

The Knapsack Problem is a classic combinatorial optimization problem that has been studied for over a century. The premise of the problem is simple: given a set  $\mathcal{S} = \{a_1, \dots, a_n\}$  of  $n$  objects, where each object  $a_i$  has an integer size  $s_i$  and profit  $p_i$ , we wish to pack a knapsack with capacity  $B \in \mathbb{Z}$  in such a way that the profit of the packed items is maximized without violating the knapsack's capacity. By introducing a binary variable  $x_i$  for each  $i \in \{1, \dots, n\}$  that indicates whether or not object  $a_i$  should be included in the knapsack, we can formulate this problem as an integer program:

$$\begin{aligned} Z^* = \max \quad & \sum_{i=1}^n p_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^n s_i x_i \leq B \\ & x \in \{0, 1\}^n. \end{aligned}$$

This is a relatively simple integer program: it requires only a single constraint other than the domain restriction  $x \in \{0, 1\}^n$ . However, the Knapsack Problem is an example of an NP-hard optimization problem, which means we do not have a polynomial time algorithm that finds a solution. However, several algorithms have been developed which approximate the optimal objective  $Z^*$  in polynomial time, and others even find an optimal solution in pseudo-polynomial time.

# Contents

- 1 Variations
- 2 Applications
- 3 Approximation Schemes
- 4 PTAS for Knapsack
  - 4.1 Greedy Approximation Algorithm
  - 4.2 The PTAS Algorithm
- 5 Pseudo-polynomial Time Algorithm
  - 5.1 Dynamic Programming
  - 5.2 Dynamic Programming for Knapsack
- 6 FPTAS for Knapsack
  - 6.1 Proof of Approximation
- 7 Implementation and Experimentation
- 8 Conclusions
- 9 References

## Variations

The Knapsack Problem as formulated above, where each individual object is either included in the knapsack or left out of the knapsack, is known as the 0-1 Knapsack Problem. However, several variations of the problem have also been formulated. For instance, in the Bounded Knapsack Problem, we assume that there are multiple copies of each object in  $\mathcal{S}$ , and up to  $c \in \mathbb{Z}$  copies of each object may be included in the knapsack. In the Unbounded Knapsack Problem, we again have copies for each object, but there are no restrictions on the number of copies that may be placed in the knapsack. Even more complicated variations include the Multiple Knapsack Problem, in which we have  $m$  different knapsacks that we are trying to fill with elements of  $\mathcal{S}$  in order to maximize profit. Nevertheless, the simple 0-1 Knapsack Problem has been studied the most and appears most frequently in practice, so it will be the variation we refer to throughout the remainder of this page.

## Applications

Other than preparing for a backpacking trip, when might the Knapsack Problem be useful? This combinatorial optimization problem actually comes up quite frequently in various resource allocation situations. For instance, an investor may wish to determine which assets to purchase given some maximum budget, or a computer program may need to determine how to allocate memory for segments of data with varying sizes. In fact, in a 1998 study of the Stony Brook University Algorithm Repository, the Knapsack Problem was the fourth most needed out of 75 algorithmic problems.

Furthermore, the Knapsack Problem often appears as a subproblem or a special case of other important optimization problems such as the Cutting Stock Problem or the Bin Packing Problem. Therefore, efficient algorithms for the Knapsack Problem allow for effective algorithms for a variety of other problems.

# Approximation Schemes

The Knapsack Problem is an NP-Hard optimization problem, which means it is unlikely that a polynomial time algorithm exists that will solve any instance of the problem. However, algorithms known as approximation schemes offer a viable alternative. Such an algorithm is guaranteed to provide a solution whose corresponding objective value is within a certain percentage of the optimal objective value. In other words, if  $Z^*$  is the optimal objective value, then given an error parameter  $\epsilon > 0$ , an approximation scheme results in an objective value that is at least  $(1 - \epsilon)Z^*$ . (If we were concerned with a minimization problem rather than a maximization problem, the resulting objective value would be at most  $(1 + \epsilon)Z^*$ .)

If the run time of such an algorithm is bounded by a polynomial in  $n$ , the size of the problem, then we say it is a Polynomial Time Approximation Scheme (PTAS). However, this allows for the possibility that the run time is exponential in the error term  $1/\epsilon$ , which means the algorithm may not be a viable choice if we desire a very small error.

To avoid this issue, we may desire a Fully Polynomial Time Approximation Scheme (FPTAS): an approximation scheme whose run time is bounded polynomially in both  $n$  and  $1/\epsilon$ .

## PTAS for Knapsack

A Polynomial Time Approximation Scheme for the Knapsack Problem can be achieved by extending partial, small-size solutions via a greedy algorithm.

### Greedy Approximation Algorithm

A heuristic technique proposed by George Dantzig is a naive but fast approach to the Knapsack Problem. First, sort the objects of  $\mathcal{S}$  in decreasing order according to their unit profit  $p_i/s_i$ , which can be done in  $O(n \log n)$  time. Next, examine the objects in this order, adding the current object to the knapsack if and only if there is room for it. The run time of this algorithm is bounded polynomially in  $n$  as it terminates after only  $O(n \log n + n) = O(n \log n)$  operations. Unfortunately, a solution obtained via this algorithm may be arbitrarily bad compared to an optimal solution. Nevertheless, we can modify this algorithm to obtain a PTAS.

### The PTAS Algorithm

The PTAS described here is an exhaustive search that employs the previously described greedy algorithm. Let some constant  $k \in \{1, \dots, n\}$  be given. For each subset  $S \subset \mathcal{S}$  that has size at most  $k$ , place the objects of  $S$  in the knapsack (if they fit) and use the Greedy Approximation Scheme to fill up the remainder of the knapsack in  $O(n)$  time. After this has been repeated for all such subsets  $S$ , keep the most profitable solution.

There are  $O(n^k)$  subsets of  $\mathcal{S}$  with size at most  $k$ , so the algorithm will terminate in  $O(n^{k+1})$  time. Furthermore, it is guaranteed to produce an objective value at least  $\left(1 - \frac{1}{k+1}\right)Z^*$ , which means it is a PTAS! However, the run time is still exponential in  $k$ , so this algorithm is not ideal if a high level of accuracy is required.

# Pseudo-polynomial Time Algorithm

Although we do not have a polynomial time algorithm for the Knapsack Problem, we do have an algorithm that uses dynamic programming in order to find an optimal solution in pseudo-polynomial time.

## Dynamic Programming

Dynamic programming is a computational technique for optimization problems that recursively breaks down the original problem into many subproblems. The smallest such subproblems are easy to solve, and their solutions are then used to solve the second stage of subproblems. Next, the solutions of these subproblems can be used to solve the third stage of subproblems. This process is continued until an optimal solution to the original problem can be obtained. This technique can be memory intensive due to a potentially large number of subproblems, but by storing the solutions of these subproblems, the computation time needed to solve the next stage of subproblems is minimal.

## Dynamic Programming for Knapsack

Dynamic programming offers an efficient approach to the Knapsack Problem. Let  $P$  denote the maximum profit across all objects of  $\mathcal{S}$ ; that is,  $P = \max_{i \in \{1, \dots, n\}} \{p_i\}$ . Then the optimal objective value of the problem is clearly bounded above by  $nP$ . Next, for each  $i \in \{1, \dots, n\}$  and  $p \in \{1, \dots, nP\}$ , let  $S_{i,p}$  denote a subset of  $\{a_1, \dots, a_i\}$  (the first  $i$  objects of  $\mathcal{S}$ ) whose profit is precisely  $p$  and whose size  $\left(\sum_{a_j \in S_{i,p}} s_j\right)$  is as small as possible. Finally, let  $A(i, p)$  denote the size of  $S_{i,p}$  if such a subset exists; otherwise let  $A(i, p) = \infty$ . Based on these definitions, an optimal solution to the original Knapsack Problem is the set  $S_{n,p}$  for which  $p$  is maximized and  $A(i, p) \leq B$ .

We can determine the values  $A(i, p)$  via dynamic programming in which the different values of  $i$  correspond to different stages of subproblems. The first stage of subproblems is to determine to values of all  $A(1, p)$ . Clearly, the only way to do this is to set  $A(1, p_1) = s_1$  and then  $A(1, p) = \infty$  for all  $p \neq p_1$ .

To solve subsequent stages of subproblems, we use the following recursive definition for each  $p \in \{1, \dots, nP\}$ :

$$A(i+1, p) = \begin{cases} \min\{A(i, p), s_{i+1} + A(i, p - p_{i+1})\} & \text{if } p_{i+1} \leq p \\ A(i, p) & \text{if } p_{i+1} > p, \end{cases}$$

where  $A(i, 0)$  is defined to be 0 for each  $i \in \{1, \dots, n\}$ . In other words, depending on the values of  $p_{i+1}$  and  $s_{i+1}$ , either  $S_{i+1,p}$  can be formed by adding the object  $a_{i+1}$  to the set  $S_{i,p-p_{i+1}}$ , or it must hold that  $a_{i+1} \notin S_{i+1,p}$  (in which case  $S_{i+1,p} = S_{i,p}$  and  $A_{i+1,p} = A_{i,p}$ ).

Since each of the  $n$  stages of subproblems involves evaluating  $nP$  values  $A(i, p)$ , this algorithm terminates after  $O(n^2 P)$  operations. Therefore, this dynamic programming method is a viable pseudo-polynomial time algorithm for the Knapsack Problem. The following pseudocode gives a basic description of the algorithm:

```
1. // Input:
2. // Number of objects (n)
   ed in array p)
Typesetting math: 100%
```

```

4. // Sizes (stored in array s)
5. // Knapsack capacity (B)
6. // Maximum Profit (P)
7. //
8. // Output:
9. // Optimal profit (Z)
10.
11. for i from 1 to n do:
12.     A[i, 0] := 0
13.
14. //solve first stage subproblems
15. for p from 1 to n*P do:
16.     if p == p[1] then:
17.         A[1, p] = p[1]
18.     else:
19.         A[1, p] = infity
20.
21. //solve subsequent subproblems
22. for i from 2 to n do:
23.     for p from 1 to n*P do:
24.         if p[i] <= p then:
25.             A[i, p] = min( A[i-1, p], s[i] + A[i-1, p-p[i]] )
26.         else:
27.             A[i, p] = A[i-1, p]
28.
29. //return optimal objective
30. Z = 0
31. for p from 1 to n*P do:
32.     if A[n, p] <= B then:
33.         Z = p
34. return Z

```

## FPTAS for Knapsack

From the previously described dynamic programming algorithm, if the the profits of the objects in  $\mathcal{S}$  are polynomially bounded in  $n$ , we have a polynomial time algorithm that solves the Knapsack Problem. However, we must still consider cases in which the profits can be arbitrarily large. In these situations, we may employ an adjusted form of the dynamic programming algorithm as a Fully Polynomial Time Approximation Scheme.

As before, let  $P$  denote the maximum profit over all objects in  $\mathcal{S}$ . Then for any  $\epsilon > 0$ , we scale down the profits by a factor of  $K$ , where  $K = \frac{\epsilon P}{n}$ . Specifically, we introduce adjusted profits  $p'$  by setting  $p'_i = \left\lfloor \frac{p_i}{K} \right\rfloor$  for all  $i \in \{1, \dots, n\}$ . Using these adjusted profits, we can apply the dynamic programming algorithm to obtain a solution whose objective is at least  $(1 - \epsilon)Z^*$ . Furthermore, this algorithm completes in  $O(n^2 \left\lceil \frac{P}{K} \right\rceil) = O(n^2 \left\lceil \frac{n}{\epsilon} \right\rceil)$  time, which is polynomially bounded in both  $n$  and  $\frac{1}{\epsilon}$ . Therefore, we have a FPTAS for the Knapsack Problem.

## Proof of Approximation

To see why the solution returned by the adjusted price dynamic programming algorithm has an objective value of at least  $(1 - \epsilon)Z^*$ , note that due to the scaling and rounding down of the profits, each adjusted profit  $p'_i$  satisfies  $p_i - K \leq Kp'_i \leq p_i$ . Hence, if  $S^*$  denotes an optimal set for the original problem, we have since  $|S^*| \leq n$  that:

$$\sum_{a_i \in S^*} p_i - K \sum_{a_i \in S^*} p'_i \leq nK.$$

Therefore, if  $S'$  denotes an optimal set for the adjusted profit problem, we must have

$$\begin{aligned} \sum_{a_i \in S'} p_i &\geq K \sum_{a_i \in S^*} p'_i \\ &\geq \sum_{a_i \in S^*} p_i - nK \\ &= Z^* - \epsilon P \\ &\geq Z^* - \epsilon Z^* \quad (\text{assuming } Z^* \geq P) \\ &= (1 - \epsilon)Z^*. \end{aligned}$$

## Implementation and Experimentation

Implementations of dynamic programming for knapsack and FPTAS for knapsack can be found on the Code for Knapsack Problem Algorithms page. The KnapsackTest program can be run to randomly generate and solve/approximate an instance of the Knapsack Problem with a specified number of objects and a maximum profit. Otherwise, "hard" instances of the problem generated by David Pisinger can be tested. The .csv files for these instances may be downloaded from <http://www.diku.dk/~pisinger/codes.html>.

It may be noticed that in most typical instances of the Knapsack Problem where neither  $n$  nor  $P$  is very large, the dynamic programming algorithm finds an optimal solution very quickly and FPTAS for Knapsack is not needed. In fact, since the approximation scheme is only useful when the scaling factor  $K$  is greater than 1, FPTAS for Knapsack should only be considered as an option when  $P > n/\epsilon$ . In these situations, although the dynamic programming algorithm may still have a satisfactory run time, it must allocate a very large portion of memory for the subproblem arrays. Hence, the approximation scheme is a desirable alternative since it not only reduces run time but also drastically reduces the amount of memory required. Additionally, it is not uncommon for FPTAS for Knapsack to find an optimal solution to the problem instead of just a  $(1 - \epsilon)$ -approximation. In other words, if a solution optimizes the modified problem with scaled profits, it is likely to optimize the original problem as well. Of course, there is no guarantee that this will happen, but it further suggests that the approximation scheme is a viable option whenever the profits are simply too large for the dynamic programming algorithm to perform adequately.



# Conclusions

The Knapsack Problem is a frequently encountered combinatorial optimization problem, and although it is NP-hard, several efficient algorithms have been developed to find or approximate an optimal solution. The combinatorial structure of this problem permits a pseudo-polynomial time dynamic programming algorithm that efficiently solves nearly every reasonable instance of the problem. Furthermore, in the rare cases where the profits of the objects are exponentially large, a fully polynomial time approximation scheme may be used. However, an approximation is generally not necessary. In his article "Where are the hard knapsack problems," David Pisinger explains that as a result of decades of algorithmic improvement, virtually all instances of the Knapsack Problem encountered in practice can be efficiently solved. Nevertheless, the algorithms presented here to solve the Knapsack Problem demonstrate that seemingly difficult problems may often be approached using pseudo-polynomial time algorithms or polynomial time approximations.

## References

- Lai, Katherine. "The Knapsack Problem and Fully Polynomial Time Approximation Schemes (FPTAS)" (<http://math.mit.edu/~goemans/18434S06/knapsack-katherine.pdf>), Massachusetts Institute of Technology, 10 March 2006. Retrieved on 3 May 2017.
- Pisinger, David. "Where are the hard knapsack problems?" (<http://www.sciencedirect.com/science/article/pii/S030505480400036X>), University of Copenhagen, Computers & Operations Research 32 (September 2005). Retrieved on 3 May 2017.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Knapsack\\_Problem\\_Algorithms&oldid=521](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Knapsack_Problem_Algorithms&oldid=521)"

- 
- This page was last modified on 3 May 2017, at 11:37.
  - This page has been accessed 53,497 times.

# Kushmakar Baral

From CU Denver Optimization Student Wiki

I am a first year graduate student at CU Denver. I received my undergraduate degree in Mathematics here in CU Denver. I am currently teaching at Community College of Denver & Front Range Community College (Westminster). When I have spare time, I spend my time watching all sorts of documentaries. I fluently speak four different languages excluding English. I often go for hiking, that is one of the best part of living in Colorado. I am working with Dongdong Lu.The link to the project page is: Crime & Temperature: Scheduling Awareness Programs

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Kushmakar\_Baral&oldid=1739"

Category: Contributors

- 
- This page was last modified on 28 November 2018, at 00:40.
  - This page has been accessed 1,805 times.

# Kushmakarb1

From CU Denver Optimization Student Wiki

This is a master's project on Tutorial on calling optimization solvers in five different programming languages: C++, AMPL, Python, MATLAB, and R : Optimization Solvers in Five Different Programming Languages

Retrieved from "<https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Kushmakarb1&oldid=2079>"

Category: Contributors

- 
- This page was last modified on 24 October 2019, at 13:47.
  - This page has been accessed 170,745 times.

# Lagrange Multipliers

From CU Denver Optimization Student Wiki

The Lagrange Multipliers are a powerful solving method for a certain class of optimization problems. But if we look beyond a solving technique we can see that Lagrange Multipliers (and solving the Lagrangian Dual problem itself) yields information not readily available through solving the original optimization problem alone.

In general, given a program:

$$\begin{aligned} & \textit{optimize } f(\mathbf{x}) \\ & s.t. \quad g_i(\mathbf{x}) \leq 0 \quad i = 1, 2, \dots, k \\ & \quad \mathbf{x} \in X \end{aligned}$$

where  $X$  is the domain of definition for  $\mathbf{x}$ .

If  $f(\mathbf{x})$  and  $g_i(\mathbf{x})$  follow the "rules" of The General Lagrange Multiplier Theorem (see below), then to find an optimal solution to the program we can consider the gradient of  $f(\mathbf{x})$  in relation to the gradient of any constraints that may be active (i.e. the constraints that have a direct affect on our program at a certain point). We do this by considering any  $g_i(\mathbf{x})$  such that  $g_i(\mathbf{x}) = 0$ . A point at which the objective function's gradient and the active  $g_i(\mathbf{x})$ 's gradients are pointed in the same direction indicates an optimal solution  $\mathbf{x}^*$ . So we seek to solve the equation  $\nabla f(\mathbf{x}) = \lambda_i \nabla g_i(\mathbf{x})$  for  $\mathbf{x}^*$ . But these gradient vectors are not typically of equal magnitude, so in addition to solving for  $\mathbf{x}^*$ , we would also like to solve for  $\lambda_i$  to study how much we can scale the gradient vectors in relation to each other. These scaling factors,  $\lambda_i$ , are called Lagrange Multipliers. The theory behind the Lagrange Multipliers solving method comes from a fascinating field of study in optimization research called Lagrangian Duality. The value of  $\lambda_i$  will yield one of the following pieces of information about the original program:

$$\begin{bmatrix} \lambda_i = 0 \\ \lambda_i > 0 \\ \lambda_i < 0 \end{bmatrix} \implies \begin{bmatrix} \text{violated constraint } g_i(x) \\ \text{active constraint } g_i(x) \\ \text{constraint } g_i(x) \text{ not a binding in the original program} \end{bmatrix}$$

Note: Lagrangian Multipliers are also known as shadow prices and offer a rich economic interpretation of mathematical programming.

## Contents

- 1 The General Lagrange Multiplier Theorem <sup>[1]</sup>
- 2 Corollary to the Lagrange Multiplier Theorem <sup>[1]</sup>
- 3 Examples
  - 3.1 Example 1
  - 3.2 Example 2

Typesetting math: 100%

## The General Lagrange Multiplier Theorem <sup>[1]</sup>

Let  $O$  be an open subset of  $\mathbb{R}^n$ .

Suppose that the function  $f : O \rightarrow \mathbb{R}$  is continuously differentiable.

Let  $k \in \mathbb{N}$  such that  $k < n$ .

Suppose that the mapping  $\mathbf{G} : O \rightarrow \mathbb{R}^k$  is continuously differentiable.

Define  $S = \{\mathbf{x} \in O : \mathbf{G} = \mathbf{0}\}$ .

Suppose that  $\mathbf{x}^* \in S$  is an extrema of the function  $f : S \rightarrow \mathbb{R}$ .

Suppose that the  $k \times n$  matrix  $\mathbf{DG}(\mathbf{x}^*)$  has full row rank.

Where  $\mathbf{DG}(\mathbf{x}^*)$  is the Jacobian (or derivative matrix) of  $\mathbf{G}$  at the point  $\mathbf{x}^* \in \mathbb{R}^n$ .

Then  $\exists \lambda_1, \lambda_2, \dots, \lambda_k \in \mathbb{R}$

such that  $\nabla f(\mathbf{x}^*) = \sum_{i=1}^k \lambda_i \nabla g_i(\mathbf{x}^*)$ .

While the Theorem is quite technical, the rules are as follows:

1)  $f(\mathbf{x})$  and  $g_i(\mathbf{x})$  are continuously differentiable for  $\forall i = 1, 2, \dots, k$ .

2) We only consider equality constraints, that is any  $g_i(\mathbf{x})$  such that  $g_i(\mathbf{x}) = 0$ .

3) The derivative matrix  $\mathbf{DG}(\mathbf{x}^*)$  is defined to be the matrix

$$\begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \dots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_k}{\partial x_1} & \frac{\partial g_k}{\partial x_2} & \dots & \frac{\partial g_k}{\partial x_n} \end{bmatrix}.$$

The matrix  $\mathbf{DG}(\mathbf{x}^*)$  has to be of full row rank, meaning that the  $\mathbf{DG}(\mathbf{x}^*)$  is square and consistent. If  $\mathbf{DG}(\mathbf{x}^*)$  is not of full row rank, we can often eliminate trivial constraints to obtain a sub-matrix that does have full row rank.

So given some strict assumptions, we can solve the system resulting from the equation  $\nabla f(\mathbf{x}^*) = \sum_{i=1}^k \lambda_i \nabla g_i(\mathbf{x}^*)$  to find optimal solutions to our original system.

The following corollary is for the case that there is only one constraint.

## Corollary to the Lagrange Multiplier Theorem <sup>[1]</sup>

Let  $O$  be an open subset of  $\mathbb{R}^n$ .

Suppose that the functions  $f : O \rightarrow \mathbb{R}$  and  $g : O \rightarrow \mathbb{R}$  are both continuously differentiable.

Define  $S = \{\mathbf{x} \in O : g(\mathbf{x}) = 0\}$ .

Suppose that  $\mathbf{x}^* \in S$  is an extrema of the function  $f : S \rightarrow \mathbb{R}$ .

Suppose that  $\nabla g(\mathbf{x}^*) \neq \mathbf{0}$ .

Then  $\exists \lambda \in \mathbb{R}$

such that  $\nabla f(\mathbf{x}^*) = \lambda \nabla g(\mathbf{x}^*)$ .

Note that for both the General Lagrange Multiplier theorem and the Corollary,  $g_i(\mathbf{x}) = 0$ .

## Examples

### Example 1

Consider the primal program, P, from the Lagrangian Duality example:

$$\begin{aligned} \min x_1^2 + x_2^2 \\ \text{s.t. } -x_1 - x_2 + 4 &\leq 0 \\ x_1, x_2 &\geq 0 \end{aligned}$$

$$\text{Let } f(\mathbf{x}) = x_1^2 + x_2^2$$

$$\text{Let } g(\mathbf{x}) = -x_1 - x_2 + 4$$

Then to minimize  $f(\mathbf{x})$  subject to  $g(\mathbf{x}) = 0$  we can use the equation  $\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x})$  to solve for  $\lambda$ ,  $x_1$ , and  $x_2$

$$\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x}) \implies \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} = \lambda \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

So we seek to solve the system:

$$2x_1 = -\lambda$$

$$2x_2 = -\lambda$$

Now we have two equations and three unknowns.

To solve this system we incorporate another equation we have had all along,  $g(\mathbf{x}) = 0$ . (Recall that the Lagrangian Multiplier Theorem requires  $g(\mathbf{x}) = 0$ ).

So we solve the system:

$$2x_1 = -\lambda$$

$$2x_2 = -\lambda$$

$$-x_1 - x_2 + 4 = 0$$

This is not a hard system to solve.

Solving the system gives us the values  $\lambda = -4, x_1 = x_2 = 2 \implies \mathbf{x}^* = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$  and  $f(\mathbf{x}^*) = 8$ .

## Example 2

Consider the program:

$$\min x_1 + x_2 + 2x_3$$

$$s.t. x_1^2 + x_2^2 + x_3^2 \leq 1$$

$$\text{Let } f(\mathbf{x}) = x_1 + x_2 + 2x_3$$

$$\text{Let } g(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2 - 1 = 0$$

Then to minimize  $f(\mathbf{x})$  subject to  $g(\mathbf{x}) = 0$  we can use the equation  $\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x})$  to solve for  $\lambda, x_1$ , and  $x_2$ .

$$\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x}) \implies \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} = \lambda \begin{bmatrix} 2x_1 \\ 2x_2 \\ 2x_3 \end{bmatrix}$$

So we seek to solve the system:

$$2\lambda x_1 = 1$$

$$2\lambda x_2 = 1$$

$$2\lambda x_3 = 2$$

Now, again we have less equations than unknowns, so we must incorporate an equation that has already been given to us,  $x_1^2 + x_2^2 + x_3^2 - 1 = 0$ . (Recall that the theorem requires  $g(x) = 0$ ).

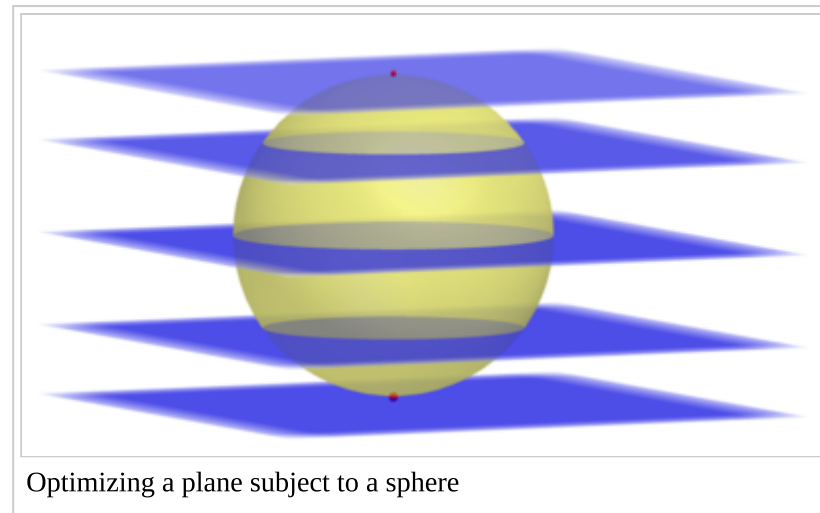
So our new system to solve becomes:

$$\begin{aligned}2\lambda x_1 &= 1 \\2\lambda x_2 &= 1 \\2\lambda x_3 &= 2 \\x_1^2 + x_2^2 + x_3^2 - 1 &= 0\end{aligned}$$

Where we solve for  $\lambda$ ,  $x_1$ ,  $x_2$ , and  $x_3$   
And again, this is not a hard system to solve.

Our resulting optimal solution for minimizing  $f(\mathbf{x})$ , occurs at  $\mathbf{x}^* = \begin{bmatrix} \frac{-\sqrt{6}}{6} \\ \frac{-\sqrt{6}}{6} \\ \frac{-\sqrt{6}}{3} \end{bmatrix}$

and our minimum value is  $f(\mathbf{x}^*) = -\sqrt{6}$



## References

1. <sup>↑</sup> <sup>1.0</sup> <sup>1.1</sup> Fitzpatrick, Patrick. Advanced Calculus. Providence, RI: American Mathematical Society, 2009. Print.  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Lagrange\\_Multipliers&oldid=348](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Lagrange_Multipliers&oldid=348)"

- This page was last modified on 15 April 2017, at 19:32.
- This page has been accessed 21,242 times.



# Lagrangian Duality

From CU Denver Optimization Student Wiki

Among the various dual program formulations, the Lagrangian Dual formulation in particular lends to a rich understanding of Duality Theory in Optimization Research. The general idea is that with the right multipliers, the Lagrangian is an auxiliary function that penalizes constraint violations linearly<sup>[1]</sup>. The original program to solve can be linear, nonlinear, quadratic, etc., and can often be quite messy. But using the Lagrangian allows for us to take messy, hard to handle, constraints and treat them as variables in a newly formed unconstrained problem. As mentioned, we incorporate non-negative multipliers, called Lagrange Multipliers, that (from the perspective of the formulating the Lagrangian itself) penalize violated constraints linearly. If we can find a solution such that none of the constraints are violated, the Lagrangian will yield a feasible solution to our original program. Since the form of the program requires minimization (see definitions below), we consider the minimum value of the Lagrangian in what is called the Lagrangian Primal problem. We can also consider the Lagrangian Dual problem that, like all Dual programs, is a closely related problem that uses the same information as the primal program but in different ways to solve an optimization problem. Since the Lagrangian Primal problem seeks to minimize, and we know from Calculus that  $MaxMin \leq MinMax$ , the Lagrangian Dual seeks to maximize the minimum value of the Lagrangian Primal. Formulating the Lagrangian Dual introduces the constraint that some of the Lagrange Multipliers must be non-negative, this allows us to not only solve for an optimal value, but also allows us to solve for the Lagrange Multipliers themselves which gives us an opportunity to study the structure and behavior of the original program. The Lagrange Multipliers are a powerful solving method for a certain class of optimization problems, but if we look beyond a solving technique we can see that Lagrange Multipliers and solving the Lagrangian Dual problem itself yields information not readily available through solving the Lagrangian Primal problem alone. Solving the Lagrangian Dual problem also illuminates new areas of study and offers theory behind rich topics like complementary slackness conditions, step sizes, and the boundaries of the feasible region.

## Contents

- 1 Lagrangian Duality in Nonlinear Programming
  - 1.1 Geometric Interpretation
    - 1.1.1 Geometric Interpretation Example<sup>[3]</sup>
  - 1.2 Duality Gap
- 2 Lagrangian Duality in Linear Programming
- 3 Notes and Additional Methods
- 4 References

## Lagrangian Duality in Nonlinear Programming

Understanding the Lagrangian Dual Problem for nonlinear programming is the foundation for understanding the theory behind Duality in Optimization Research and the ways that the dual program can be used to find optimal solutions to the primal program. Lagrangian Duality for nonlinear programming leads to a discussion of other types of programs, including linear programs and quadratic programs.

$$\begin{aligned}
& \min f(\mathbf{x}) \\
& s.t. \quad g_i(\mathbf{x}) \leq 0 \quad i = 1, 2, \dots, k \\
& \quad \quad h_i(\mathbf{x}) = 0 \quad i = 1, 2, \dots, l \\
& \quad \quad \mathbf{x} \in X \\
& \quad \text{where } X \text{ is the domain of definition for } \mathbf{x}.
\end{aligned}$$

\* The Lagrangian is defined to be:

$$L(\mathbf{x}, \mathbf{u}, \mathbf{v}) := f(\mathbf{x}) + \sum_{i=1}^k u_i g_i(\mathbf{x}) + \sum_{i=1}^l v_i h_i(\mathbf{x})$$

\* The Lagrangian Dual function,  $\theta$ , is defined to be:

$$\theta(\mathbf{u}, \mathbf{v}) := \inf \left\{ f(\mathbf{x}) + \sum_{i=1}^k u_i g_i(\mathbf{x}) + \sum_{i=1}^l v_i h_i(\mathbf{x}) : \mathbf{x} \in X \right\}$$

\* And the Lagrangian Dual Problem to solve is defined to be:

$$\begin{aligned}
& \max \theta(\mathbf{u}, \mathbf{v}) \\
& \forall \mathbf{u} \geq 0
\end{aligned}$$

Note: The Lagrangian Dual function  $\theta$  may assume the value of  $-\infty$  for some vector  $(\mathbf{u}, \mathbf{v})$ .

Note: If you are unfamiliar with the concept of the "inf" in the  $\theta(\mathbf{u}, \mathbf{v})$  function, you can consider  $\theta(\mathbf{u}, \mathbf{v})$  to be

$$\min \{ f(\mathbf{x}) + \sum_{i=1}^k u_i g_i(\mathbf{x}) + \sum_{i=1}^l v_i h_i(\mathbf{x}) : \mathbf{x} \in X \}$$

In this case, please look into the concept of an infimum, but for now consider  $\theta$  in terms of a minimum.

Consider the example:

Given the primal program, P :

$$\begin{aligned} \min \quad & x_1^2 + 4x_2^2 + \frac{x_1}{2} + 9x_2 \\ \text{s.t.} \quad & g(x_1, x_2) := -x_1 - 3x_2 + 24 \leq 0 \\ & x_1, x_2 \geq 0 \end{aligned}$$

The Lagrangian is  $L(x_1, x_2, u) := x_1^2 + 4x_2^2 + \frac{x_1}{2} + 9x_2 + u(-x_1 - 3x_2 + 24)$

The Lagrangian Dual function is  $\theta(u) := \inf\{x_1^2 + 4x_2^2 + \frac{x_1}{2} + 9x_2 + u(-x_1 - 3x_2 + 24) : x_1, x_2 \geq 0\}$

Therefore the Lagrangian Dual Problem is:

$$\begin{aligned} \max \quad & \theta(u) \\ \forall u \quad & u \geq 0 \end{aligned}$$

The variables  $\mathbf{u}$  and  $\mathbf{v}$  of the Lagrangian Dual function,  $\theta(\mathbf{u}, \mathbf{v})$ , are what incorporate the primal constraints,  $g(\mathbf{x})$  and  $h(\mathbf{x})$  into the dual program. The variables  $\mathbf{u}$  and  $\mathbf{v}$  are known as Lagrange Multipliers, and are often represented as  $\lambda$  or  $\alpha$ . In general, the punchline of the Lagrangian Dual Problem is finding non negative values of the Lagrange Multipliers,  $u_i$  and  $v_i$ , such that  $\nabla f(\mathbf{x}) = \mathbf{u}^T \nabla g(\mathbf{x}) + \mathbf{v}^T \nabla h(\mathbf{x})$ , but the theory behind the equation offers a rich understanding of the theory of Duality.

## Geometric Interpretation

To better understand Lagrangian Duality, consider a simple program with one inequality constraint and no equality constraints.

Consider the primal problem, P:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g(\mathbf{x}) \leq 0 \end{aligned}$$

The Lagrangian is  $L(\mathbf{x}, u) := f(\mathbf{x}) + ug(\mathbf{x})$

The Lagrangian function is  $\theta(u) := \inf\{f(\mathbf{x}) + ug(\mathbf{x})\}$

The dual problem is  $\max \theta(u)$

Suppose that  $u \geq 0$  is given.

Let  $z_1 = g(\mathbf{x}), z_2 = f(\mathbf{x})$  for  $\mathbf{x} \in X$ .

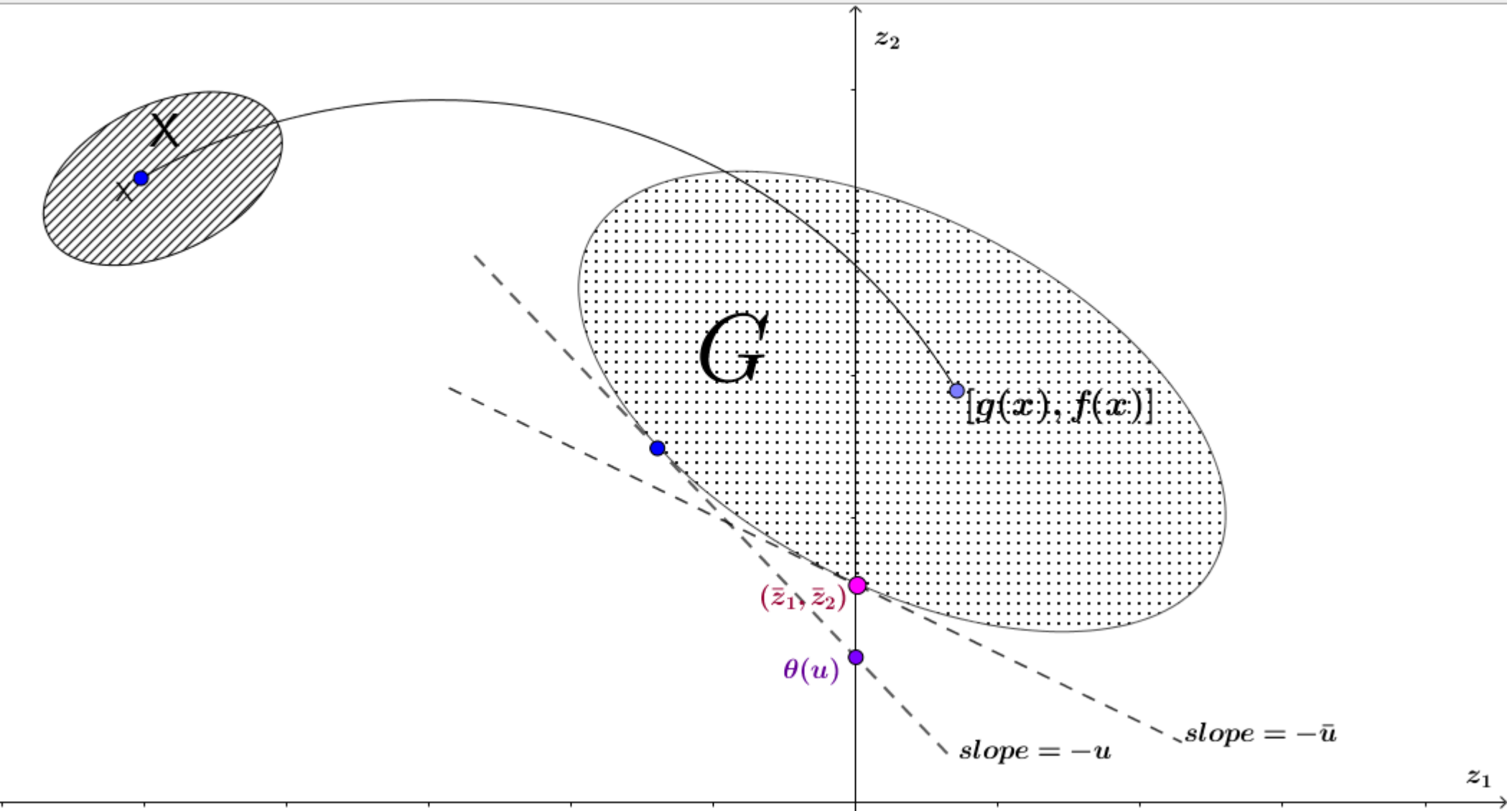
Let  $G :=$  be the image of  $X$  under the  $(g, f)$  map.

Then  $G = \{(z_1, z_2) : z_1 = g(\mathbf{x}), z_2 = f(\mathbf{x}) \text{ for some } \mathbf{x} \in X\}$ .

Note that  $z_2 + uz_1 = \alpha$  is just the equation of a line with *slope*  $= -u$  and  $z_2$  *intercept*  $= \alpha$ .

So to minimize  $f(\mathbf{x})$  we just want to "move" the line  $z_2 + uz_1 = \alpha$  down, parallel to itself, until it supports  $G$ . That is, until the line is underneath  $G$  and is just touching it.

The dual problem is equivalent to finding the slope of the supporting hyperplane such that the  $z_2$  *intercept* is maximized.



### Geometric Interpretation Example <sup>[3]</sup>

Consider the primal program P:

$$\min x_1^2 + x_2^2$$

$$-x_1 - x_2 + 4 \leq 0$$

$$x_1, x_2 \geq 0$$

Note that the optimal solution occurs at  $\mathbf{x}^* = (2, 2)$  which gives an optimal value of  $f(\mathbf{x}^*) = 8$

$$\text{Let } g(\mathbf{x}) = -x_1 - x_2 + 4$$

$$\text{Let } X = \{(x_1, x_2) : x_1, x_2 \geq 0\}$$

The dual function is:

$$\theta(u) = \inf\{x_1^2 + x_2^2 + u(-x_1 - x_2 + 4) : x_1, x_2 \geq 0\}$$

$$\theta(u) = \inf\{x_1^2 - ux_1 : x_1 \geq 0\} + \inf\{x_2^2 - ux_2 : x_2 \geq 0\} + 4u$$

$$\text{Note that the infimum is achieved at } \begin{cases} x_1 = x_2 = \frac{u}{2} & \text{if } u \geq 0 \\ x_1 = x_2 = 0 & \text{if } u < 0 \end{cases}$$

$$\text{This implies that } \theta(u) = \begin{cases} -\frac{u^2}{2} + 4u & \text{if } u \geq 0 \\ 4u & \text{if } u < 0 \end{cases}$$

Note that  $\theta(u)$  is a concave function, and that  $\max u \geq 0$  occurs at  $\bar{u} = 4 \implies \theta(\bar{u}) = 8$

Now let's consider the problem in the  $(z_1, z_2)$  plane.

$$\text{Let } z_1 = g(\mathbf{x}) \text{ and } z_2 = f(\mathbf{x})$$

$G := \text{the image of } X = \{(x_1, x_2) : x_1, x_2 \geq 0\} \text{ under the } (g, f) \text{ map} .$

And to do this, we derive explicit expressions for the lower and upper envelopes of  $G$

Let  $\alpha = \text{lower envelope of } G$

Let  $\beta = \text{upper envelope of } G$

Note that given  $z_1 = g(\mathbf{x})$ ,  $\alpha(z_1)$  and  $\beta(z_1)$  are the optimal objective values of the problems  $P_1$  and  $P_2$  respectively:

*Problem  $P_1$  :*

$$\min x_1^2 + x_2^2$$

$$\text{s.t. } -x_1 - x_2 + 4 = z_1$$

$$x_1, x_2 \geq 0$$

$$\therefore \alpha(z_1) = \frac{(4-z_1)^2}{2}$$

*for  $z_1 \leq 4$*

*Problem  $P_2$  :*

$$\max x_1^2 + x_2^2$$

$$\text{s.t. } -x_1 - x_2 + 4 = z_1$$

$$x_1, x_2 \geq 0$$

$$\therefore \beta(z_1) = (4 - z_1)^2$$

*for  $z_1 \leq 4$*

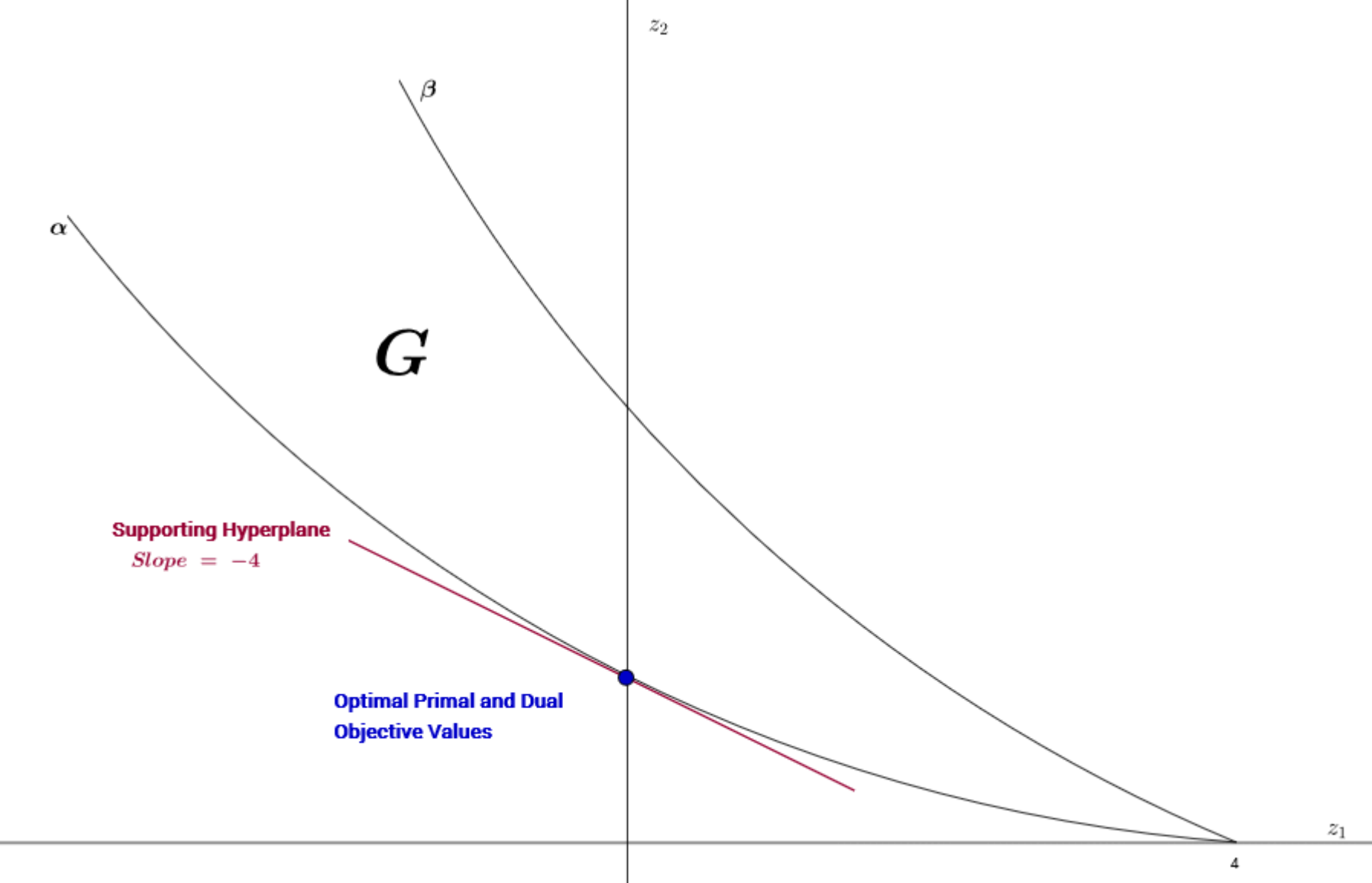
Note:  $\mathbf{x} \in X$  which implies that  $x_1, x_2 \geq 0$  such that  $-x_1 - x_2 + 4 \leq 4$

$\therefore \forall \mathbf{x} \in X$  corresponds directly to  $z_1 \leq 4$  because  $g(\mathbf{x}) \leq 4$ .

The optimal dual solution  $\bar{u} = 4$  is the negative slope of the supporting hyperplane.

And the optimal dual objective value is  $\alpha(0) = 8$ .

Recall that  $f(\mathbf{x}^*) = 8$ .



## Duality Gap

Because of the  $\text{MaxMin} \leq \text{MinMax}$  properties of the Lagrangian Primal and the Lagrangian Dual, the optimal value of the primal program will be less than or equal to the optimal value of the dual program. If the values are strictly different (that is they are not equal) then a Duality Gap is said to exist. This result is known as Weak Duality. But under certain assumptions regarding convexity, the primal and dual problems have equal objective values, thus solving the dual problem solves the



primal problem indirectly. When these values are equal, and no Duality Gap exists, the result is known as *Strong Duality*. The concepts of Weak Duality and Strong Duality are elaborated in the section on Lagrangian Duality in Linear Programming.

## Lagrangian Duality in Linear Programming

Given the linear program (LP for short):

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

We can consider the LP component-wisely and rewrite the LP as:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & b_i - a_i x_i \leq 0 \quad i = 1, 2, \dots, n . \\ & x \geq 0 \end{aligned}$$

Then the *Lagrangian formulation* is:

$$L(\mathbf{x}, \mathbf{y}) := c^T x + \sum_{i=1}^n y_i (b_i - a_i x_i)$$

Translating this back to a set of linear inequalities (where the summation of the linear constraints is implied),

the *Lagrangian formulation* is:

$$L(\mathbf{x}, \mathbf{y}) := c^T x + \mathbf{y}^T (\mathbf{b} - \mathbf{A}x) .$$

And this equation can be rearranged as follows:

$$\begin{aligned}
c^T x + y^T (b - Ax) &= c^T x + y^T b - y^T Ax \\
&= c^T x + - y^T Ax + y^T b \\
&= (c^T - y^T A)x + y^T b. \\
&= - (y^T A - c^T)x + y^T b
\end{aligned}$$

So for  $x \geq 0$ , we want  $y^T A - c \leq 0$ . That is, for  $x \geq 0$ , we want  $A^T y \leq c$ , where we seek to maximize  $y^T b$ .

Therefore, the Lagrangian Dual Problem is:

$$\begin{aligned}
&\max y^T b \\
&s.t. \ A^T y \leq c \\
&\quad y \geq 0
\end{aligned}$$

Note: In linear programming, the dual program does not involve the primal variables. And the dual problem is itself a linear program, and the dual of the dual is the primal.

And as the Lagrangian is an auxiliary function that bounds the set of feasible solutions of the primal program linearly, we can see that the formulation of the Lagrangian Dual program leads to the following result:

$$\begin{aligned}
\text{minimum of primal LP} &\geq \text{maximum of dual LP} \quad . \\
\min c^T x &\geq \max y^T b .
\end{aligned}$$

This result is known as Weak Duality

Furthermore, for a finite optimal solution to the primal LP,  $\bar{\mathbf{x}}$ , with optimal value  $z = c^T \bar{\mathbf{x}}$ , the dual LP has a finite optimal solution  $\bar{\mathbf{y}}$  with an optimum value  $w = \bar{\mathbf{y}}^T b$ , and these values are equal!

$$\text{That is, } z = w \implies c^T \bar{\mathbf{x}} = \bar{\mathbf{y}}^T b .$$

This result is known as Strong Duality.

Given that for a finite optimal solution,  $c^T x = y^T b$ , we can obtain:

$$\begin{aligned}
c^T x &= y^T b \\
c^T x &= b^T y \\
0 &= c^T x - b^T y \\
0 &= c^T x - y^T A x + y^T A x - b^T y \\
0 &= (c^T - y^T A) x + y^T (A x - b^T)
\end{aligned}$$

Therefore, component wise,  $(c^T - y^T A)_i \neq 0 \implies x_i = 0$  and  $(A x - b^T)_j \neq 0 \implies y_j = 0$ .

These two results together are known as Complementary Slackness Conditions, and offer a rich interpretation of the behavior of the constraints in both the primal and dual programs.

Considering the linear primal program, P, and linear dual program, D, as presented above, one of the following mutually exclusive cases will occur:<sup>[3]</sup>

- 1) P admits a feasible solution, but has an unbounded optimal value.  
 $\implies$  D is infeasible.
- 2) D admits a feasible solution, but has an unbounded optimal solution.  
 $\implies$  P is infeasible.
- 3) P and D admit feasible solutions  
 $\implies$  both P and D have optimal solutions  $\bar{x}$  and  $\bar{y}$  such that  $c^T \bar{x} = \bar{y}^T b$  and  $(c^T - \bar{y}^T A) \bar{x} = \mathbf{0}$  respectively.
- 4) P and D are both infeasible.

## Notes and Additional Methods

In general, to find an optimal solution to our primal program, P, we can consider the gradient of our primal objective function,  $f(\mathbf{x})$ , in relation to the gradient of any constraints that may be active. A point at which the objective function's and the active constraints' gradients are pointed in the same direction indicates an optimal solution  $\mathbf{x}^*$ . So we seek to solve the equation  $\nabla f(\mathbf{x}) = u \nabla g(\mathbf{x}) + v \nabla h(\mathbf{x})$  for  $\mathbf{x}^*$ ,  $u$ , and  $v$ . But the gradient vectors are not typically of equal magnitude, so in addition to solving our original system, we would like to study how much we can scale the gradient vectors. The scaling factors associated with the active constraints of each of the systems P and D, are known as Lagrange Multipliers and are sometimes referred to as shadow prices. The economic interpretation of Lagrange Multipliers as shadow prices lends to a rich economic application of the relationship between the primal and dual programs.

## References

1. ↑ Freitas, Guilherma. "Crash Course on Convex Optimization." California Institute of Technology. HSS Division. Fall 2011-12.
2. ↑ Geogebra
3. ↑ 3.0 3.1 Bazarara, Mokhtar S., and C. M. Shetty. Nonlinear Programming: Theory and Algorithms. New York: Wiley, 1979. Print.

- This page was last modified on 17 April 2017, at 19:35.
- This page has been accessed 32,731 times.

# Lauren Hearn

From CU Denver Optimization Student Wiki

Lauren Hearn recieved her MS in Applied Mathematics from the University of Colorado Denver in May of 2019. She completed her undergraduate education at Metropolitan State University, earning a BS in Physics. Past research includes analysis of the visual spectrum of AGN emissions and sound wave analysis of non-traditional fret material.

Her current research includes data assimilation for wildfire modeling and L1 estimation of fire arrival time. Current and past work can be found on github (<https://github.com/mandlebrot>).

Lauren and her family are enthusiastic supporters of the arts and public media.

Here are Lauren's contributions to the Optimization Wiki:  
Mapping Accident Prone Regions

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Lauren\\_Hearn&oldid=2066](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Lauren_Hearn&oldid=2066)"

Category: Contributors



RobotLauren

- This page was last modified on 13 June 2019, at 15:42.
- This page has been accessed 1,938 times.

# Lillian makhoul

From CU Denver Optimization Student Wiki

## Contents

- 1 About Me
- 2 Previous Education
- 3 Current Projects
- 4 Previous Projects
- 5 Previous and Current Research Topics and Experience

## About Me

I am currently a second-year Applied Mathematics PhD student at the University of Colorado, Denver.

## Previous Education

- Lehigh University, B.S. in Statistics, minor in Computer Science

## Current Projects

- Computing the volume of the convex hull of a trilinear monomial over a general domain box - with Dr. Emily Speakman
- Optimizing Water Allocation of the Colorado River
- An Overview of Dijkstra's Algorithm

## Previous Projects

- A Stochastic Approach to the Burrito Problem - with Abigail Nix
- Data to Policy Symposium 2023 - Optimizing SAR Paths in the Rocky Mountain Region with Nicholas Rogers and Matthew Knodell

# Previous and Current Research Topics and Experience

In 2022 I was exposed to my first research experience at the Markov Chains REU at the University of Connecticut, run by Iddo Ben-Ari. I presented the results from this program at the 2023 Joint Mathematics Meetings with colleague Bram Silbert. From there I focused quite a lot on social and opinion dynamics, more specifically the voter model, but also explored a bit of the work done by Mason Porter on the bounded confidence model.

Most recently, I've been working closely with Emily Speakman on a number of projects. We're currently wrapping up my work with her on computing volume formulae (the results for this were presented at a poster session at JMM and in a talk at FRAMSC), and are shifting gears into studying Stochastic Programming and its applications, specifically in relation to water storage and allocation.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Lillian\\_makhoul&oldid=4866](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Lillian_makhoul&oldid=4866)"

- 
- This page was last modified on 16 April 2025, at 11:55.
  - This page has been accessed 37 times.

# Linear Algebra

From CU Denver Optimization Student Wiki

## Basic Definitions

A vector is a magnitude with a direction. In general, a vector is given by the directional components. For example, a 2-dimensional vector in the cartesian plane that goes 3 units in the x direction and 4 units in the y direction is denoted as  $\begin{bmatrix} 3 \\ 4 \end{bmatrix}$ . A 3-dimensional vector in with the same components and no height in the z direction looks like  $\begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}$ . It is important to note that even though these vectors appear to be the same, they are not. They operate in different **vector spaces**. A vector space is a collection of vectors of the same dimension. A vector space is closed under addition and scalar multiplication. That is to say, the addition of any vectors in the vector space with any other vectors in the space, will result in a vector in the space. Similarly, the scaling of any vector by a constant will result in a vector in the space. A matrix is a collection of vectors that form a vector space. Entries in a matrix  $M$  are denoted by  $M_{ij}$ , which is the  $i^{th}$  entry in the  $j^{th}$  row. The main diagonal entries in a matrix are all of the entries  $M_{ij}$  such that  $i = j$ .

## Basic Operations

A matrix transpose, denoted  $M^T$  is a matrix that has been "flipped" along the main diagonal. For example,  $\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}^T = \begin{vmatrix} 1 & 3 \\ 2 & 4 \end{vmatrix}$ . A symmetric matrix is a matrix  $M$  where  $M = M^T$ . In other words,  $M_{ij} = M_{ji} \forall i, j$ . The conjugate transpose of a matrix, denoted  $M^*$ , with complex number entries is obtained by taking the transpose of the matrix and then taking the complex conjugate of each entry. A Hermitian matrix is a matrix such that  $M = M^*$ .

## Products

An inner product of two vectors is a scalar quantity relating the two vectors. For any vectors  $x, y, z$  and scalar  $a$ , the following hold for inner products:

Conjugate symmetry: The vector product of two vectors is equal to the conjugate of their inverse product  $\langle x, y \rangle = \overline{\langle y, x \rangle}$ . When dealing with the reals, this reduces to symmetry  $\langle x, y \rangle = \langle y, x \rangle$ .

Linearity in the first argument: The first argument is closed under addition and scalar multiplication.

$$\begin{aligned} \langle ax, y \rangle &= a \langle x, y \rangle \\ \langle x + y, z \rangle &= \langle x, z \rangle + \langle y, z \rangle \end{aligned}$$



And positive-definiteness:  $\langle x, x \rangle \geq 0$   
 $\langle x, x \rangle = 0 \Leftrightarrow x = \mathbf{0}$ .

The dot product of two vectors is a special case of the inner product where the domain is real-valued vectors.

Matrix multiplication:

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Linear\_Algebra&oldid=735"

- 
- This page was last modified on 5 December 2017, at 15:47.
  - This page has been accessed 666 times.

# Linear Programming Methods for Radiation Therapy Treatment Planning

From CU Denver Optimization Student Wiki

## Contents

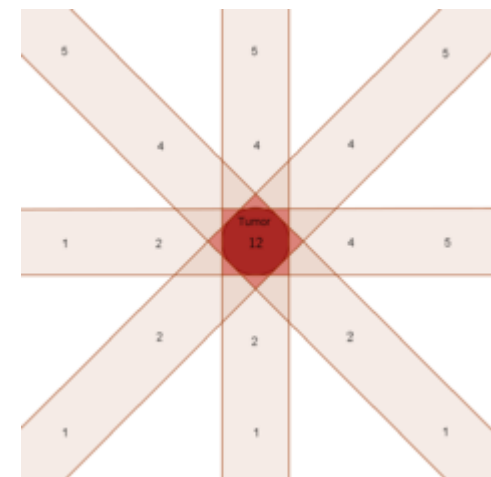
- 1 Introduction
- 2 Radiation Therapy
  - 2.1 Treatment Area
- 3 Dose Matrix
- 4 Linear Program Formulation
- 5 Simplified Program
- 6 Dual Program Formulation
  - 6.1 Simplified Dual Program
- 7 Conclusions
- 8 References

## Introduction

In 2017, the American Cancer Society estimates nearly 1.7 million new instances of cancer in the United States <sup>[1]</sup>. There are five different types of cancer treatment, one of which is radiation therapy. Linear programming methods for radiation therapy treatment planning is a fascinating and practical application of optimization research and given changing technologies in the field of radiation oncology, the linear programming methods continue to grow and the application can be extended to other medical fields. We will explore here the foundations of the application including the procedure, the treatment area, the initial linear program formulation, it's components, and the dual program.

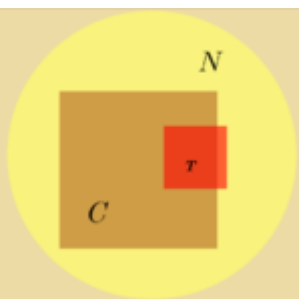
## Radiation Therapy

There are three types of radiation therapy: the use of radioactive drugs (either ingested or administered through an IV), internal beam radiation therapy (where a radioactive source is placed inside of the patient's body near the cancerous cite), and **external beam radiation therapy**. We explore here external beam radiation therapy, where prescribed amounts of radiation (decided by a physician) are delivered to the patient by a machine called a linear accelerator. The patient lies on a treatment bed, and the linear accelerator <sup>[2]</sup> moves around the patient delivering beams of radiation with different intensities from various angles to target the cancerous cite, called a tumor. There are multiple methods for external beam radiation therapy, but here we consider what is know as "pencil beam radiation therapy" <sup>[3]</sup> in that the beams of radiation delivered to the patient are of uniform width and shape.



# Treatment Area

To formulate a linear program for radiation therapy treatment planning, we first need to understand the treatment area and translate information about physical tissue into mathematical terms. The treatment area<sup>[4]</sup> under consideration consists of normal tissue, target tissue (the tumor), and possibly critical tissue (like tissue of the spinal chord or an organ where dosages of radiation can be especially harmful). We impose a two or three dimensional grid over the treatment area and classify the corresponding pixels or voxels as belonging to one of the sets:



$N$  := is the normal tissue, consisting of  $n_N$  pixels/voxels.

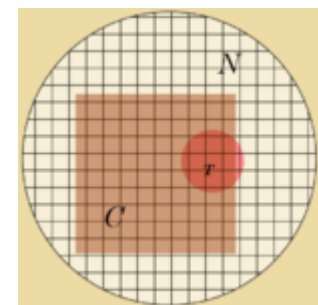
$C$  := is the critical tissue, consisting of  $n_C$  pixels/voxels.

$T$  := is the target tissue, consisting of  $n_T$  pixels/voxels.

Thus, the entire treatment area under consideration is  $S := N \cup C \cup T$ .

For the discussion of a foundational and simple linear program formulation, we consider any pixel/voxel to belong to only one of the sets  $N$ ,  $C$ , or  $T$ . There are more advanced linear programs that accommodate for a pixel/voxel belonging to two sets at the same time, say  $T$  and  $C$ , though we do not explore those programs here. In fact, to understand the simplest model, we consider critical tissue to belong to  $N$  and only consider the sets

$T$  of target area voxels, and  $N$  of "non-target" area voxels. We can accommodate for dosage restrictions on critical tissue in the constraints of our model.



# Dose Matrix

Now that we have established a way to classify physical tissue mathematically, we can translate this information into matrix notation. Recall that in external beam radiation therapy, a linear accelerator delivers prescribed dosages of radiation to the tumor via beams of radiation with different intensities. We call the intensity of each beam at a particular pixel/voxel, it's **weight**. The total sum of the beam weights to any given pixel/voxel is called the **dosage**.

We construct a **dose matrix**,  $A$ , such that each entry in the matrix corresponds to the dosage delivered to a single voxel in the set  $S$ .

0	0	0	0	5
0	0	0	4	0
1	2	6	4	5
0	2	0	0	0
1	0	0	0	0

And we consider  $A_T$  as the submatrix whose nonzero entries correspond to only the target voxels, and likewise  $A_N$  as the submatrix whose nonzero entries correspond to the "non-target" voxels, and  $A_T + A_N = A$ . The examples given here are very small compared to how large the dose matrix is in practice. A typical dose matrix ranges from a dimension of 16 to a dimension of 96<sup>[5]</sup>.

0	0	0	0	5
0	0	0	4	0
1	2	6	4	5
0	2	0	0	0
1	0	0	0	0

# Linear Program Formulation

Define:

$w$  := vector of beam weights  
 $x_T$  := vector of dosage to target voxels  
 $x_N$  := vector of dosage to normal voxels  
 $c_N$  := cost vector of dosage to normal voxels, whose components are beam weights  
And consider the submatrices  $A_T$  and  $A_N$  as defined above.

Thus we can formulate the following linear program <sup>[6]</sup> for treatment planning:

$$\begin{aligned} \min \quad & c_N^T x_N \\ \text{s.t.} \quad & x_T = A_T w \\ & x_N = A_N w \\ & x_N \leq x_N^U \\ & x_T^L \leq x_T \leq x_T^U \\ & w \geq 0 \end{aligned}$$

The beam weights are the decision variables in our linear program, and we seek to **minimize the dosage delivered to the non-target tissue** subject to lower and upper constraints on dosages on the target voxels,  $x_T^L$  and  $x_T^U$  respectively, and upper dosage restrictions,  $x_N^U$ , on the voxels of the non-target area. Through understanding the components of this linear programming and it's formulation, we have gained insight into the foundations of all linear programming methods in radiation therapy treatment planning. Now that we have an understanding of the basic linear program, we can move forward and look at a few variations including the dual.

## Simplified Program

The initial linear program can be simplified using substitution and basic linear algebra to obtain:

$$\begin{aligned} \min \quad & c_N^T A_N w \\ \text{s.t.} \quad & A_N w \leq x_N^U \\ & x_T^L \leq A_T w \\ & A_T w \leq x_T^U \\ & w \geq 0 \end{aligned}$$

And can be rewritten to obtain the form:

$$\begin{aligned}
 \min \quad & (A_N^T c_N)^T w \\
 \text{s.t.} \quad & -A_N w \geq -x_N^U \\
 & A_T w \geq x_T^L \\
 & -A_T w \geq -x_T^U \\
 & w \geq 0
 \end{aligned}$$

The simplified linear program is often less computationally expensive to solve than the original, but much of the information pertaining to the treatment area and dosage prescription are lost, and thus it is harder to translate the results back to the physician.

## Dual Program Formulation

We know from duality theory in linear programming,

If the primal is of the form:

$$\begin{aligned}
 \min \quad & c^T x \\
 \text{s.t.} \quad & Ax \geq b \\
 & x \geq 0
 \end{aligned}$$

The Dual takes the form:

$$\begin{aligned}
 \max \quad & b^T y \\
 \text{s.t.} \quad & A^T y \leq c \\
 & y \geq 0
 \end{aligned}$$

So given the simplified program:

$$\begin{aligned}
 \min \quad & (A_N^T c_N)^T w \\
 \text{s.t.} \quad & -A_N w \geq -x_N^U \\
 & A_T w \geq x_T^L \\
 & -A_T w \geq -x_T^U \\
 & w \geq 0
 \end{aligned}$$

The Dual can be formulated as:

$$\begin{aligned}
max \quad & (-x_N^U)^T y_B + (x_T^L)^T y_L + (-x_T^U)^T y_U \\
s.t. \quad & y_N = c_N + y_B \\
& y_T = y_U - y_L \\
& -A_T^T y_T \leq A_N^T y_N \\
& y_B, y_L, y_U \geq 0
\end{aligned}$$

The theory of duality in linear programming is a rich topic that allows us to interpret the model in various ways. For a more in depth look at how to compute and interpret the dual of a linear program please see: Duality: Bounding the Primal, Shadow Prices, Duality: Economic Example, and Lagrangian Duality. For our simplified linear program for radiation therapy, we can interpret the dual as a new mathematical program that considers the bounds on the dosage constraints. Unlike in the original program that explicitly describes dosage to the target and non-target voxels, in the dual we are describing the boundaries on the dosage restrictions. So, we discuss the upper and lower dosage boundaries on the target voxels with the vectors  $y_U$  and  $y_N$  and define  $y_T = y_U - y_L$ . In our original program we considered upper dosage restrictions on the normal voxels, so we define the vector  $y_B$  corresponding to the upper bounds of dosage to the normal region. But given that in our original program the objective function minimizes the total amount of radiation delivered to the normal voxels, we consider  $c_N$  corresponding to the lower bounds of dosage to the normal voxels and thus we define  $y_N = c_N + y_B$  to describe the bounds on the dosage delivered to the normal tissue.

## Simplified Dual Program

And we can further simplify the Dual to the form:

$$\begin{aligned}
max \quad & (-x_N^U)^T y_B + (x_T^L)^T y_L + (-x_T^U)^T y_U \\
s.t. \quad & A_T^T (y_L - y_U) \leq A_N^T (c_N + y_B) \\
& y_B, y_L, y_U \geq 0
\end{aligned}$$

The simplified dual is not only compact, it also displays more clearly that we are discussing the bounds of dosage requirements on the target tissue relative to the bounds on the dosage restrictions for the non-target tissues.

## Conclusions

There is a vast amount of research done on the topic of linear programming methods in radiation therapy treatment planning. The technologies in radiation oncology and the math itself are constantly developing to improve and introduce new models for cancer treatment. The linear programs presented here serve as an introduction to the topic, and are in no way comprehensive of the models being used in practice. The models discussed here each have multiple variations, even in their simple forms, and serve as the foundations for understanding more complicated and more advanced linear programming methods in radiation therapy treatment planning.

## References

2. ↑ Cherry, Pam, and Angela Duxbury. Practical Radiotherapy: Physics and Equipment. Blackwell Pub., 2009.
3. ↑ Bansal, Nikhil, et al. “Shape Rectangularization Problems in Intensity-Modulated Radiation Therapy.” *Algorithmica*, vol. 60, no. 2, 2009, pp. 421–450., doi:10.1007/s00453-009-9354-8
4. ↑ Epelman, Marina. 2013. Examples of LP models: Radiation Therapy, lecture notes. Linear Programming II, IOE 610. University of Michigan, September 2013.
5. ↑ Schlegel, Wolfgang. The Use of Computers in Radiation Therapy: XIIIth International Conference, Heidelberg, Germany May 22-25, 2000. Springer, 2000.
6. ↑ Ólafsson, Arinbjörn and Stephen J. Wright. “Linear Programing Formulations and Algorithms for Radiotherapy Treatment Planning.” *Optimization Methods and Software*, vol. 21, no. 2, 2006, pp. 201–231., doi:10.1080/10556780500134725

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Linear\\_Programming\\_Methods\\_for\\_Radiation\\_Therapy\\_Treatment\\_Planning&oldid=1098](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Linear_Programming_Methods_for_Radiation_Therapy_Treatment_Planning&oldid=1098)"

---

- This page was last modified on 20 February 2018, at 15:20.
- This page has been accessed 9,714 times.

# Linear Programming for Diet and Nutrition

From CU Denver Optimization Student Wiki

## Contents

- 1 An Unhealthy America: A High Need for Accurate Nutrition
  - 1.1 The Inadequacies of Other Diets
  - 1.2 A Review On Linear Program Diets
  - 1.3 Summary of all present Diets (LP and non LP Related)
- 2 Proposed Linear Program for our Functional Diet: A Multi-Objective Linear Program
  - 2.1 Test Subject (Client)
  - 2.2 Meal Plan with 2 Options and Additional Recommendations:
- 3 Multi-Objective Linear Program for a Functional Diet
  - 3.1 Dictionary of Variables
  - 3.2 Meal Codes
  - 3.3 Objective Function 1: Minimize Cost
  - 3.4 Objective Function 2: Maximize Profit
  - 3.5 Multi-Objective Function: The Difference of Minimizing Cost and Maximizing Profit
  - 3.6 References

## An Unhealthy America: A High Need for Accurate Nutrition

Given that almost 3 out of 4 Americans are overweight, 1.3 out of 3 Americans are obese, 1 in 5 men (**women**) 45 (**55**) years and older will have some kind of cardiac disease/dysfunction and that the United States of America is one of the most medicated countries in the world[1], there is a very high need for better nutrition.

### The Inadequacies of Other Diets

Given that there are so many diets in the world, but there seems to be so much controversy and risks with many of these diets, if not all of them. Many of these diets may have the person lose weight too fast only to gain it back just as fast. Some diets may have the person eat too much protein which can also cause increased uric levels and negatively affect your kidneys in the long run. Many of these diets may have the person lose weight, but they end up losing the wrong kind of weight (For example if they lost 40 lbs, but 37 lbs of this weight loss was muscle mass and only 3 lbs of fat). A lot of these diets are also 'a one size fits all' and they don't consider your unique basal metabolic rate (BMR), daily lifestyle and the amount of your physical activity/inactivity. There are other flaws in some of these diets such as the Weight Watchers Diet have you eat anything you want but you just eat smaller portions. This may not be necessarily a bad thing, however, this diet does not consider if you are still getting all of your essential vitamins and minerals as you can eat foods with the same amount of calories but their nutrient density is lacking.



# A Review On Linear Program Diets

There are only several articles and publications with **Linear Programs** that consider the cost of these diets. In 2002, Darmon and Ferguson discussed when introducing a cost constraint during the selection of foods that it reduced the proportion of energy contributed by fruits and vegetables, meat and dairy products and increased the proportion from cereals, sweets and added fats, a pattern similar to that observed among low socioeconomic groups[6]. This implied that less nutrient dense foods having the same amount of calories were associated with lower socioeconomic groups and that in order for our diets to be better, our economy will have to be better.

In 2015, Drewnowski found that many diets that consider calories with nutrient dense food can be more expensive compared to foods that are high in calories but have less nutrient density. He also found out there are still foods that are not as expensive and they do have the right caloric amount combined with having high nutrient density and that coming up with a less expensive diet is quite feasible[7]. He also noted that many of these diets can also be pleasing to one's palate while still considering 'social norms' acceptance.

There were many other good and relevant publications regarding Linear Programming and Nutrition. Some considered high contaminants in nutrient dense foods[4] or minimizing cost for malnutrition-ed children in Africa using therapeutic type foods[5] whilst others considered that food choices for high nutrient dense foods given the right amount of caloric intake were correlated with people who were categorized to be in an above average socioeconomic class. However, none of these articles considered the person's daily activities (caloric expenditure), occupation, lifestyle and BMR combined with their age, sex, height, and type of build (mesomorph, endomorph, and ectomorph).

## Summary of all present Diets (LP and non LP Related)

These diets may have some good points, but they are lacking and not considering many important factors (as mentioned above) that can positively negate our health and well being. They might possibly be a step up from the person's regular regime, but they are still most likely to be considered the lesser of two evils, which is nowhere close to being OPTIMAL.

## Proposed Linear Program for our Functional Diet: A Multi-Objective Linear Program

We are hoping to design a LP functional diet based on one's age, sex, body weight, type of build, BMR, Lifestyle and physical activity/inactivity that will then prescribe the correct daily amount of carbohydrates(50%), fats(20%) and proteins(30%) while minimizing cost and maximizing profit given all of the appropriate constraints. A Multi-Objective program is a LP with more than one objective function. This LP solves how to minimize cost (based on the dollar value of meals) and maximize profit (based on the grams of protein) while meeting specific requirements for both the consumer and supplier. This multi-objective LP will ultimately be developed by combining the separate objectives. In order to execute this program, we will need a Test Subject and a Meal Plan.

### Test Subject (Client)

We will consider a Male who is 45 years old, and he is a Computer Programmer (A very sedentary job). He works out three times a week with moderate intensity and each work out he is estimated to burn about 500 calories per workout (350 carbohydrate and 150 fat calories). His basal metabolic rate is 1765 (We used the Harrison Benedict Formula for men to equate this) calories per day. Other than working out three times a week (This is an additional 1500 calories in one week.) and being a Computer Programmer and walking his dog every day about 15 to 20 minutes per day at about 3 miles per hour (This is about 100 calories per day which is equated into his BMR as the Benedict Formula can overestimate Caloric BMR by about 5 to 10%), the rest of his lifestyle is sedentary. He is 6 feet tall, 175 lbs heavy (mesomorph) and would like to maintain his present physical status but he is worried about if his diet contains the correct amount of nutrients combined with the correct prescription of carbohydrates, proteins and fats, especially as he is getting older. Given his lifestyle, he would like to eat at the most optimal level to ensure good health and well being with minimal cost.

## Meal Plan with 2 Options and Additional Recommendations:

**Meal 1:** Price: Shrimp Paella \$10.00, 260 calories, protein: 20g, carbs: 24g, fats: 10g

**Meal 2:** Price: Spicy Dan Noodles \$7.00 , 430 calories, protein: 9g, carbs: 58g, fats: 19g

**Meal 3:**Price: Turkey Chili with beans, \$5.50, 290 calories, protein: 29g, carbs: 16g, fats: 13g

**Meal 4:** Price: Supreme Pasta, \$8.50, 340 calories, protein: 23g, carbs: 30g, fats: 17g

**Meal 5:** Price: Over Easy Burger with Sweet Potato, \$10.00, 470 calories, protein: 27g, carbs: 31g, fats: 26g

**Meal 6:** Price: Sweet Chili Glazed Salmon, \$13.00, 360 Calories, protein: 11g, carbs: 48g, fats: 15g

**Meal 7:** Price: Tex Mex Chicken Bowl, \$9.50, 340 Calories, protein: 32g, carbs: 21g, fats: 16g

**Meal 8:** Price: Ab&J Oatmeal Bowl. \$7.00, 420 Calories, protein: 14g, carbs:40g, fats: 24g

**Meal 9:** Price: B Platter. \$9.00, 410 Calories, protein: 34g, carbs: 9g, fats: 27g

**Meal 10:** Price: Buffalo Style Chicken Bowl. \$9.50, 430 Calories, protein: 32g, carbs: 41g, fats: 16g

**Meal 11:** Price: Chicken Teriyaki with Rice. \$8.50, 360 Calories, protein: 11g, carbs: 48g, fats: 15g

**Meal 12:** Price: Turkey Meat Loaf, \$8.00, 330 calories, protein: 43g, carbs: 31g, fats: 4.5g

**We will also give the client two options of choosing the most economical meal plan which will have less variety, and the most economical meal plan which will have more variety.**

**We are also recommending our client to do the following with this diet:** Drink 2 to 4 Liters of water per day, depending on one’s activities and the condition of the external environment Don’t drink a gallon of water or a very large amount all at once (Water Intoxication) Eat Higher Carbohydrate meals in the morning and about 2 hours before working out Eat Higher Protein meals after working out (Ideally 30 to 40 minutes after working out) and closer to the end of the day (Evening Time).

# Multi-Objective Linear Program for a Functional Diet

## Dictionary of Variables

$x_j$	<i>number of meal type <math>j</math> purchased in a week</i>
$\beta_j$	<i>minimum number of meal type <math>j</math> purchased in a week</i>
$\mu_j$	<i>maximum number of meal type <math>j</math> purchased in a week</i>
$c_j$	<i>cost of meal type <math>j</math></i>
$a_{ij}$	<i>amount of nutrient type <math>i</math> in meal <math>j</math></i>
$\zeta_i$	<i>minimum number of nutrient type <math>i</math> needed in a week</i>
$\eta_i$	<i>maximum number of nutrient type <math>i</math> needed in a week</i>
$p_j$	<i>amount of protein in meal <math>j</math></i>
$\gamma$	<i>the maximum number of meals made in a week</i>
$\lambda_1 \in (0, 1]$	<i>scalar weight of minimize cost objective</i>
$\lambda_2 \in (0, 1]$	<i>scalar weight of maximize profit objective</i>

## Meal Codes

<i>SP</i>	<i>Shrimp Paella</i>
<i>SDN</i>	<i>Spicy Dan Noodles</i>
<i>TC</i>	<i>Turkey Chili with Beans</i>
<i>PAS</i>	<i>Supreme Pasta</i>
<i>OEB</i>	<i>Over Easy Burger</i>
<i>SCS</i>	<i>Sweet Chili Glazed Salmon</i>
<i>TMC</i>	<i>Tex Mex Chicken Bowl</i>
<i>OAT</i>	<i>ABJ Oatmeal Bowl</i>
<i>BP</i>	<i>B. Platter</i>
<i>BSC</i>	<i>Buf falo Style Chicken Bowl</i>
<i>CT</i>	<i>Chicken Teriyaki with Rice</i>
<i>TML</i>	<i>Turkey Meat Loaf</i>

## Objective Function 1: Minimize Cost

The nutritional data to minimize the cost of a week's worth of meals was designed for a male, 45 years old, 6 ft tall, 175 lbs heavy and of mesomorphic build. We calculated his Basal Metabolic Rage to be 1765 calories per day (This included him walking his dog for 15-20 minutes every day) and his expected expenditure of 1500 calories per week (A total of 3 workouts, burning about 500 calories per workout.). It is therefore prescribed that the consumer has a minimum weekly nutritional intake ( $\zeta_i$ ) of the following: 12355 calories, 1239 grams of protein, 1393 grams of carbohydrates, and 203 grams of fat. To promote variety in the diet, each meal must be consumed at least once and may not be consumed more than 5 times (i.e.  $1 = \beta_j \leq x_j \leq \mu_j = 5$ ). Based on the cost and nutritional information of each of the meals, the objective function can be written as:

$$\begin{aligned}
\min \quad & \sum_{j=1}^m c_j \cdot x_j \\
s.t. \quad & \zeta_i \leq \min \sum_{j=1}^m a_{ij} \cdot x_j \leq \eta_i \\
& i = 1, 2, \dots n \\
& j = 1, 2, \dots m
\end{aligned}$$

### Interpretation of Results

The optimal solution is \$354, which can be interpreted as the minimum cost to buy a weeks worth of food while meeting all nutritional requirements and sufficiently varying the meals. The meals that were least expensive but was higher in nutrient density were purchased in higher quantities. The following diet is the optimal number of each meal that should be purchased:

<i>BP</i>	1
<i>BSC</i>	5
<i>CT</i>	5
<i>OAT</i>	5
<i>OEB</i>	1
<i>PAS</i>	5
<i>SCS</i>	3
<i>SDN</i>	5
<i>SP</i>	1
<i>TC</i>	4
<i>TMC</i>	1
<i>TML</i>	5

It is important to note that B. Platter (BP), Over-Easy Burger (OEB), Shrimp Paella (SP), and Tex-Mex Chicken Bowl (TMC) have the least nutritional benefits for this diet per dollar.

## Objective Function 2: Maximize Profit

The definition of profit in this model is to sell as much protein as possible. Each meal may only be made 5 (i.e.  $0 = \beta_j \leq x_j \leq \mu_j = 5$ ) times to promote variety, and only 50 meals may be made in a given week (i.e.  $\gamma = 50$ ) . The objective function for this model can be written as:

$$\begin{aligned} \max \quad & \sum_{j=1}^m p_j \cdot x_j \\ \text{s.t.} \quad & \sum_{j=1}^m x_j \leq \gamma \\ & j = 1, 2, \dots m \end{aligned}$$

### Interpretation of Results

The optimal solution is 1325 grams of protein. The results of this objective function are intuitive -- make 5 of the meal with the most protein and then move on the to the meal with the next most protein. The following is the optimal number of each meal that should be purchased:

<i>BP</i>	5
<i>BSC</i>	5
<i>CT</i>	0
<i>OAT</i>	5
<i>OEB</i>	5
<i>PAS</i>	5
<i>SCS</i>	5
<i>SDN</i>	0
<i>SP</i>	5
<i>TC</i>	5
<i>TMC</i>	5
<i>TML</i>	5

It is important to note that Spicy Dan Noodles (SDN) and Chicken Teriyaki (CT) have the least amount of protein, so neither were made. It is preferred to make 5 of each of the other 10 meals.

## Multi-Objective Function: The Difference of Minimizing Cost and Maximizing Profit

Combining the two previous objective functions to create the multi-objective linear program entails using the same data as from the single objective functions and writing one minimization objective function. Since previously the objective was to maximize profit, I can achieve this by subtracting the maximization of the profit from the minimization of cost. Utilizing  $\lambda_1$  and  $\lambda_2$  to weight each of the objective functions, the multi-objective function can therefore be written as:

$$\min \quad (\lambda_1 \cdot \sum_{j=1}^m c_j - \lambda_2 \cdot \sum_{j=1}^m p_j) \cdot x_j$$

$$s.t. \quad \zeta_i \leq \min \sum_{j=1}^m a_{ij} \cdot x_j \leq \eta_i$$

$$\sum_{j=1}^m x_j \leq \gamma$$

$$i = 1, 2, \dots, n$$

$$j = 1, 2, \dots, m$$

### Interpretation of Results

The interpretation of the multi-objective optimal solution is difficult since units of the minimization of cost objective function (dollars) is different than units of the maximization of profits objective function (grams of protein). The value of the objective function is of little significance. It is more important to examine how many of each meal satisfies the multi-objective function. Below are 3 cases where each separate objective function is weighted:

Case 1 ( $\lambda_1 = \lambda_2 = 1$ )

The optimal solution is -868.06. When each part of the mutli-objective function receives equal weight, maximizing grams of protein is greater than minimizing dollars, which results in a negative optimal solution. The following is the optimal number of each meal that should be purchased:

<i>BP</i>	5
<i>BSC</i>	5
<i>CT</i>	1
<i>OAT</i>	5
<i>OEB</i>	5
<i>PAS</i>	5
<i>SCS</i>	1
<i>SDN</i>	5
<i>SP</i>	3
<i>TC</i>	5
<i>TMC</i>	5
<i>TML</i>	5

Case 2 ( $\lambda_1 = 1, \lambda_2 = .01$ )

The optimal solution is 344.68. When significantly more weight is given to the minimization of cost objective function, the resulting optimal solution becomes positive. Again, it is important to examine how many of each meal should be purchased to achieve the optimal solution:

<i>BP</i>	1
<i>BSC</i>	5
<i>CT</i>	5
<i>OAT</i>	5
<i>OEB</i>	1
<i>PAS</i>	5
<i>SCS</i>	3
<i>SDN</i>	5
<i>SP</i>	1
<i>TC</i>	4
<i>TMC</i>	1
<i>TML</i>	5

Case 3 ( $\lambda_1 = .5, \lambda_2 = .01$ )

The optimal solution is 167.72. Even though less weight was given to the minimization of cost objective function compared to Case 2, the optimal solution decreased but



remained positive. It is interesting to note that although the optimal solution was lower in Case 3 than Case 2, the number of each meal that should be purchased to achieve the optimal solution is the same as in Case 2:

<i>BP</i>	1
<i>BSC</i>	5
<i>CT</i>	5
<i>OAT</i>	5
<i>OEB</i>	1
<i>PAS</i>	5
<i>SCS</i>	3
<i>SDN</i>	5
<i>SP</i>	1
<i>TC</i>	4
<i>TMC</i>	1
<i>TML</i>	5

## References

1. ACSM (American College of Sports Medicine). <http://www.acsm.org/public-information/health-physical-activity-reference-database>
2. Trust For America’s Health. <http://healthyamericans.org/reports/stateofobesity2017/>
3. Snap Kitchen. <https://www.snapkitchen.com/>
4. Barre T, Vieux F, Perignon M, Cravedi JP, Darmon N. Reaching Nutritional Adequacy Does Not Necessarily Increase Exposure To Food Contaminants: Evidence From A Whole Diet Modeling-Approach. September, 2016. The Journal Of Nutrition. Volume 96: 439-444.
5. Dibari F, Diop el HI, Collins S, Seals A. Low-cost, ready-to-use therapeutic foods can be designed using locally available commodities with the aid of linear programming. May, 2012. J Nutrition. 142(5):955-61
6. Darmon N, Ferguson E, Briend A. A Cost Constraint Alone Has Adverse Effects on Food Selection and Nutrient Density: An Analysis of Human Diets by Linear Programming. December, 2002. The American Society for Nutritional Sciences. Volume 132 no. 12: 3764-3771.
7. Drewnowski A. The Carbohydrate-Fat Problem: Can We Construct a Healthy Diet Based on Dietary Guidelines. May, 2015. Advances in Nutrition. Volume 6: 318-325.
8. Roze AM, Shizgal HM. Harris Benedict Formula for Women and Men Re-evaluated: Resting Energy Requirements and the Body Cell Mass. July 1984. The American

Clinical Nutrition. 40 (1): 168–182.

**9.** Mifflin M.D. BMR Formula: A new predictive equation for resting energy expenditure in healthy individuals. The American Journal of Clinical Nutrition. 51 (2): 241–247 PMID 2305711.

**10.** Robert Fourer, David M. Gray and Brian W. Kernighan. AMPL A MODELING LANGUAGE FOR MATHEMATICAL PROGRAMMING. Second Edition, 2003. Print. Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Linear\\_Programming\\_for\\_Diet\\_and\\_Nutrition&oldid=1571](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Linear_Programming_for_Diet_and_Nutrition&oldid=1571)"

- 
- This page was last modified on 26 September 2018, at 04:09.
  - This page has been accessed 46,669 times.

# Linear Programming

From CU Denver Optimization Student Wiki

A linear program is a special type of mathematical program where all of the functions are linear, and should be familiar to anyone that has taken linear algebra. They are programs with the set of  $m$  constraints and  $n$  variables defined by an  $m \times n$  matrix. A linear optimization problem takes the form:

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & \sum Ax^T = b \\ & x \leq \geq 0 \end{array}$$

With  $c^T x$  as the objective function, or the thing that being optimized.  $Ax^T$  is the constraint matrix, which consists of all the bounds for the problem.  $b$  is the set of right hand side values for the constraint matrix. The final line are bounds on  $x$ . Linear programs were the first type of mathematical program and were developed for the U.S. Army in the 1960's to resolve scheduling and transportation issues. Linear programs are the simplest type of mathematical program in that they are defined by a coefficient matrix, instead of a set of functions.

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Linear\_Programming&oldid=263"

- This page was last modified on 28 March 2017, at 09:45.
- This page has been accessed 2,133 times.

# Linear Regression as Linear Programming

From CU Denver Optimization Student Wiki

This page was created by Eric Olberding.

Linear regression is one of the most commonly used statistical modelling methods. In linear regression, we model the dependent variable as a linear function of the independent variables. The model's parameters should be chosen to minimize the discrepancy between the dependent variable predicted by the model and the observed dependent variable. In essence, one's parameters should minimize some "lack of fit" function. In this project we show that, when using the Least Absolute Deviations (LAD) lack of fit function, linear regression can be formulated as a linear program. We demonstrate this by constructing two examples in AMPL. The *mtcars* and *iris* default datasets from R are used. The code for this project can be found at: <https://github.com/eric072891/LinProg>

## Contents

- 1 Model Formulation
- 2 Lack of Fit Functions
  - 2.1 Ordinary Least Squares (OLS)
  - 2.2 Least Absolute Deviations(LAD)
- 3 Least Absolute Deviations as a Linear Program
- 4 Uses of Linear Regression

## Model Formulation

In linear regression, we model some response (or dependent) variable  $y$  as a linear combination of predictor (or independent) variables  $x_i$  plus some random error  $\epsilon$ . A statistical unit is a set of measurements for a single object. Suppose we study  $n$  such objects and the  $j$ th object has measurements  $y_j, x_{j,1}, \dots, x_{j,m}$ . Note that we have measured  $m + 1$  different properties per object (including the response/dependent variable). We use this information to write our model as:

$$y_j = \beta_0 + \beta_1 x_{j,1} + \beta_2 x_{j,2} + \dots + \beta_m x_{j,m} + \epsilon_j; \quad j = 1, \dots, n$$

Here the  $\beta_i$  are the constants we must find that minimize our lack of fit function. The figure on the right shows the case when  $m = 1$ .

Often these  $n$  equations are stacked together and written in matrix notation as

$$\mathbf{v} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$
$$X = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,m} \\ 1 & x_{2,1} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,m} \end{pmatrix},$$
$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{pmatrix}, \quad \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}.$$

Note that  $X\boldsymbol{\beta}$  are the response values predicted by our model and that  $\mathbf{y}$  are the observed response values. Also note that the error can be

written  $\boldsymbol{\epsilon} = \mathbf{y} - X\boldsymbol{\beta} = \begin{pmatrix} y_1 - (\beta_0 + \beta_1 x_{1,1} + \dots + \beta_m x_{1,m}) \\ y_2 - (\beta_0 + \beta_1 x_{2,1} + \dots + \beta_m x_{2,m}) \\ \vdots \\ y_n - (\beta_0 + \beta_1 x_{n,1} + \dots + \beta_m x_{n,m}) \end{pmatrix}$

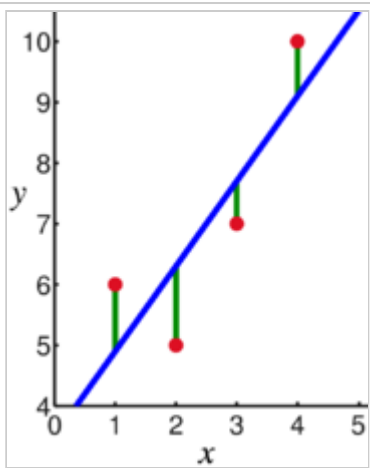
We have more than one choice for how to minimize this error vector. We describe two such methods below.

## Lack of Fit Functions

In order to reduce the error as much as possible. We want to choose  $\boldsymbol{\beta}$  to minimize a lack of fit function. This function somehow captures the magnitude of the overall error.

### Ordinary Least Squares (OLS)

The most commonly used lack of fit function is the mean squared error.



In linear regression, the observations  $y_j$  (**red**) are assumed to be the result of random deviations  $\epsilon_j$  (**green**) from an underlying relationship  $\beta_0 + x_{j,1}\beta_1$  (**blue**) between a dependent variable  $y$  and an independent variable  $x_1$ .

$$L(\boldsymbol{\beta}) = ||X\boldsymbol{\beta} - Y||^2 = \sum_{i=1}^n |y_i - \beta_0 - \sum_{j=1}^m x_{ij}\beta_j|^2$$

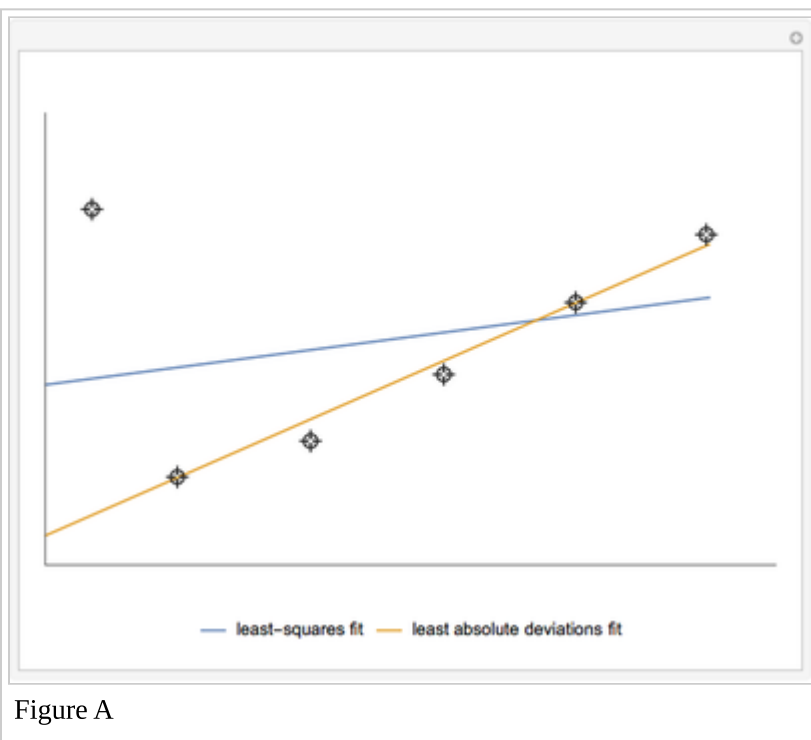
This is just the magnitude of a vector. This is a quadratic positive function, so there exists a unique  $\boldsymbol{\beta}$  that attains a global minimum for the loss function.

## Least Absolute Deviations(LAD)

The loss function for LAD regression is very similar to that of OLS. The only difference is that we no longer square the absolute values.

$$L(\boldsymbol{\beta}) = \sum_{i=1}^n |y_i - \beta_0 - \sum_{j=1}^m x_{ij}\beta_j|$$

LAD regression tends to be more robust to outliers. This means that the estimation  $\boldsymbol{\beta}$  is not as heavily influenced by outliers. For OLS, if the predicted response is far away from the observed response, then that large error is squared. This isn't the case for LAD.



However, LAD may have multiple solutions, unlike for OLS. One known case in which multiple solutions exist is a set of points symmetric about a horizontal line, as shown in Figure B below.

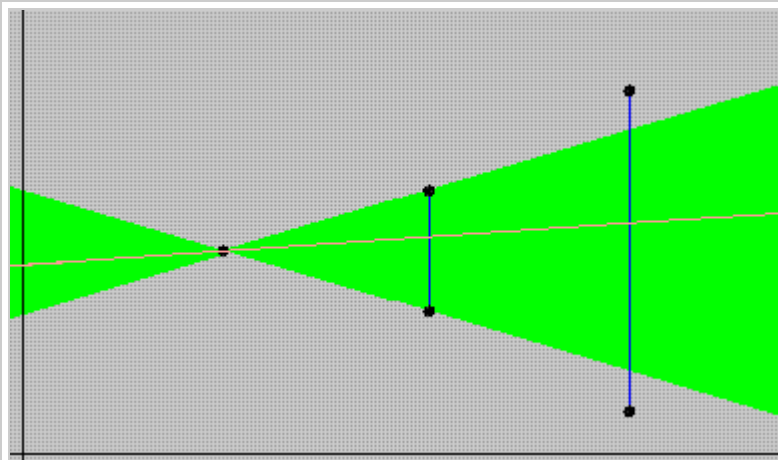


Figure B: A set of data points with reflection symmetry and multiple least absolute deviations solutions. The “solution area” is shown in green. The vertical blue lines represent the absolute errors from the pink line to each data point. The pink line is one of infinitely many solutions within the green area.

To understand why there are multiple solutions in the case shown in Figure B, consider the pink line in the green region. Its sum of absolute errors is some value  $S$ . If one were to tilt the line upward slightly, while still keeping it within the green region, the sum of errors would still be  $S$ . It would not change because the distance from each point to the line grows on one side of the line, while the distance to each point on the opposite side of the line diminishes by exactly the same amount. Thus the sum of absolute errors remains the same. Also, since one can tilt the line in infinitely small increments, this also shows that if there is more than one solution, there are infinitely many solutions.

## Least Absolute Deviations as a Linear Program

We formulate LAD in terms of linear optimization. We need the fact that  $|x| < t$  if and only if  $-t < x < t$ . In the context of our problem we use

$$|y_i - \beta_0 - \sum_{j=1}^m x_{ij}\beta_j| < t_i \Leftrightarrow -t_i < y_i - \beta_0 - \sum_{j=1}^m x_{ij}\beta_j < t_i$$

We can see that minimizing  $\sum_{i=1}^n t_i$  minimizes the loss function since

$$\sum_{i=1}^n |y_i - \beta_0 - \sum_{j=1}^m x_{ij}\beta_j| < \sum_{i=1}^n t_i$$

This means that we can formulate LAD regression in terms of the following linear program in AMPL.

$$\min \sum_{i=1}^n t_i$$

$$-t_i < y_i - \beta_0 - \sum_{j=1}^m x_{ij}\beta_j < t_i; \quad i = 1, \dots, n$$

In this LP we try to find an optimal  $\beta$ .

## Uses of Linear Regression

A fitted linear regression model can be used both predict new values and find which of the independent variables impacts the dependent variable the most.

Suppose we have the model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

and that we find the coefficient vector to be  $\beta = (0, 0, 10)$ . This gives us the fitted model

$$\hat{y} = 0 + 0x_1 + x_2$$

Where  $\hat{y}$  are the response variable values predicted by our fitted model. In this example it's likely that  $x_1$  has no impact on  $y$  since its corresponding coefficient is  $\beta_1 = 0$ .

This model can also be used to predict new response values, given new independent variable values. Suppose we are given  $x_1 = 3, x_2 = 12$ , but we do not know what  $y$  is. We can predict y as

$$\hat{y} = 0 + 0 * 3 + 12 = 12$$

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Linear\_Regression\_as\_Linear\_Programming&oldid=2447"

- This page was last modified on 3 December 2019, at 02:07.
- This page has been accessed 6,959 times.



# Livvi Bechtold

From CU Denver Optimization Student Wiki

I am a first year phd student and Teaching Assistant at University of Colorado Denver. My mathematical interests are Number Theory, Cryptography, Graph Theory, and Optimization. At the University of Nebraska at Omaha, where I received my Masters in Education with an emphasis in mathematics, I spent much of my research time focused on making mathematics more accessible to students. While working on a Masters in Mathematics I received a Kerrigan Grant for my work on the bicycle rebalancing project for b-cycle of Omaha. Currently my education research focus is writing in the undergraduate mathematics classroom to help students develop mathematical language and be able to communicate their thoughts, successes and struggles with mathematics to others. In the world of optimization my current focus is derivative free optimization and its applications. When I am not at school, I spend much of my free time with fiber arts. I am an accomplished quilter, weaver, and spinner. I also love knitting, crochet, and needlepoint just to name a few. Over the past few years while working on my computer programming skills, I have been developing small programs that help determine how much warp and weft are needed for weaving projects and am currently developing a program that will optimize production of various items by combining say a set of hand towels and washcloths to have less waste and more sellable products after each warping of a loom. My goal is to complete the programs as apps that might help smaller cottage industry to take advantage of the same optimization algorithms used by larger factories while maintaining the unique individual feel of their product.

Multi-commodity Flow

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Livvi\\_Bechtold&oldid=1511](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Livvi_Bechtold&oldid=1511)"

Category: Contributors

- 
- This page was last modified on 10 May 2018, at 14:19.
  - This page has been accessed 1,559 times.

# Denver Hate Crime Mapping: Visualizing Fluctuations through Linear Programming

From CU Denver Optimization Student Wiki

(Redirected from Location Fluctuations in Denver Area Bias Motivated Crime Explored Through Linear Programming)

## Contents

- 1 Abstract
- 2 Introduction to Bias Motivated Crime
- 3 Methods
- 4 Analysis and Results
  - 4.1 Case Study: 2011-2014
  - 4.2 Case Study: 2014-2017
- 5 Discussion
- 6 References
- 7 Motivation
- 8 Resources

## Abstract

Bias-motivated crime--criminal acts galvanized by prejudice--showed an uptick in the year 2017 after holding relatively steady in the United States for over a decade. This surge in domestic extremism has awakened concerns about public safety. These crimes are of special interest to the Federal Bureau of Investigation and thus many police departments, including Denver, collect information on the bias-motivated crimes committed within their communities and share it with the national database. This data is available from 2010 to 2018 and is updated frequently. The number of bias-motivated crimes reported in Denver also increased in 2017, similar to the national trend, while the crimes reported to date for 2018 suggest that this year will be equally high. In this project, linear optimization techniques were applied to the data released to the public by the Denver Police Department. This research detected patterns of interest, some matching national trends and others in opposition.

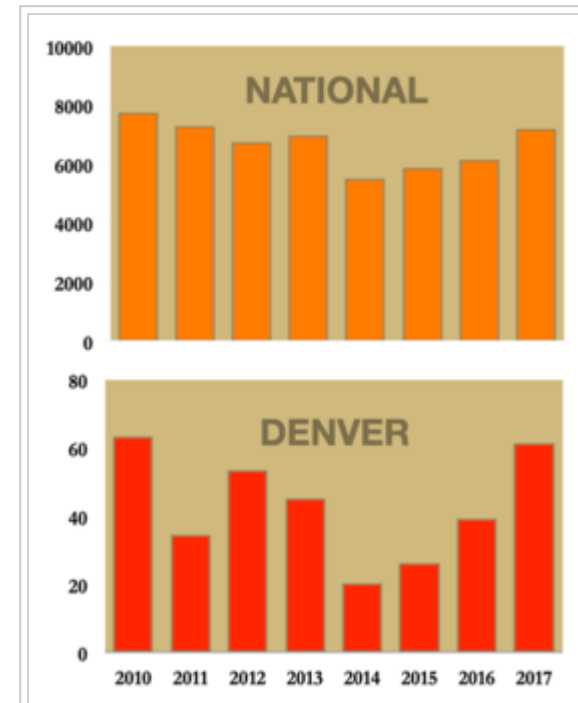
## Introduction to Bias Motivated Crime

Bias motivated crime, commonly called hate crime, are acts committed against a person or persons in an attempt to victimize an entire group of people.

In Colorado, protected categories include: disability, ethnicity, gender identity, race, religion, and sexual orientation. The Federal Bureau of Investigation also considers hate

Typesetting math: 100%

these are tracked in Denver.

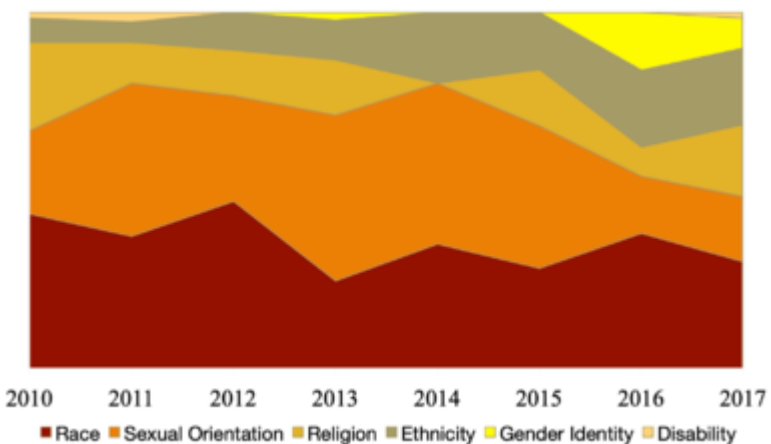


Number of hate crimes reported nationally and in Denver, by year, 2010-2017.

National statistics are gathered monthly by the FBI as part of their Uniform Crime Reporting (UCR) program and summary data is released in a yearly report. Several Colorado agencies contribute, including the Denver Police Department.

Reported hate crimes in Denver tend to follow national trends, but due to the relatively small number of reported crimes, fluctuations can appear magnified. In addition, hate crime is tends to be underreported meaning that increases in reported crime may reflect growing awareness of the issue on the part of the public and not an escalation in bias motivated crime.

Bias motivated crime was high at the beginning of the decade, but declined over the next few years, only to surge again in 2016-2018. Since 2010, the percentage of crimes motivated by race and religion bias in Denver tends to be much lower than the national average. Denver tends to report a higher percentage of crimes related to sexual orientation than the national average. Crimes related to ethnicity (primarily crimes against the Latino community) have increased in Denver, exceeding the national average, especially in recent years. Denver has also seen a dramatic increase in crimes related to gender identity in the past two years, far beyond the national average.



Percentage of various types of hate crime in Denver, by year, 2010-2017.

## Methods

This project used a linear sum assignment problem to connect crimes committed in one year to another. While assignment problems tend to be used to connect people and jobs, or resources and plants, they can also be used to show spatial shifts.

The mathematics is relatively simple. Choose two years. Let  $i$  denote the crimes in the earlier year, and  $j$  be the crimes in the following year. Define the Euclidean distance between locations to be  $c(ij)$  and let  $x(ij)$  be a Boolean variable that is true is there is a connection between  $i$  and  $j$ , and false otherwise . We choose our objective function to be the minimization of the summation of all distances.

$$\min(\sum_{i \in S} \sum_{j \in D} c_{ij} x_{ij})$$

We have two classes of constraints. Without loss of generality, we set the assignments emerging from each origin equal to one,

$$\sum_{i \in S} x_{ij} = 1 \text{ for all } i \in S$$

Since there tend to be a different number of origin  $n_S$  and destination  $n_D$  crimes, the destination constraint is that the assignments to each destination are equal to the ratio between the number of origins and destinations,

$$\sum_{j \in D} x_{ij} = \frac{n_D}{n_S} \text{ for all } j \in D$$

## Analysis and Results

### Case Study: 2011-2014

In this project two assignments were analyzed. Since there were major changes between 2010 and 2018, three years were chosen, 2011, 2014, and 2017, three years apart.

Between 2011 and 2014, reported hate crimes decreased in Denver.

Reported hate crimes were scattered throughout the city in 2011, often occurring near major roadways.

By 2014, we see hate crime condensing into downtown with isolated incidents in less travelled areas. Hate crimes in Five Points and North Capitol Hill moved from major roads onto side streets.

### Case Study: 2014-2017

From 2014 to 2018 there was a dramatic increase in hate crimes in the city.

By 2017 the reported numbers were equal to where they were in 2010. New hot spots developed in 2017, including Hampden and the neighborhoods near Ruby Hill Denver University also was a target with numerous reports of anti-Jewish activities. Numerous hate crimes remain in Downtown, but now center on the Union Station transit center and Arapahoe.

The increase in 2017 seems to be primarily driven by increased reports of criminal mischief, and may not indicate increased aggression and danger.

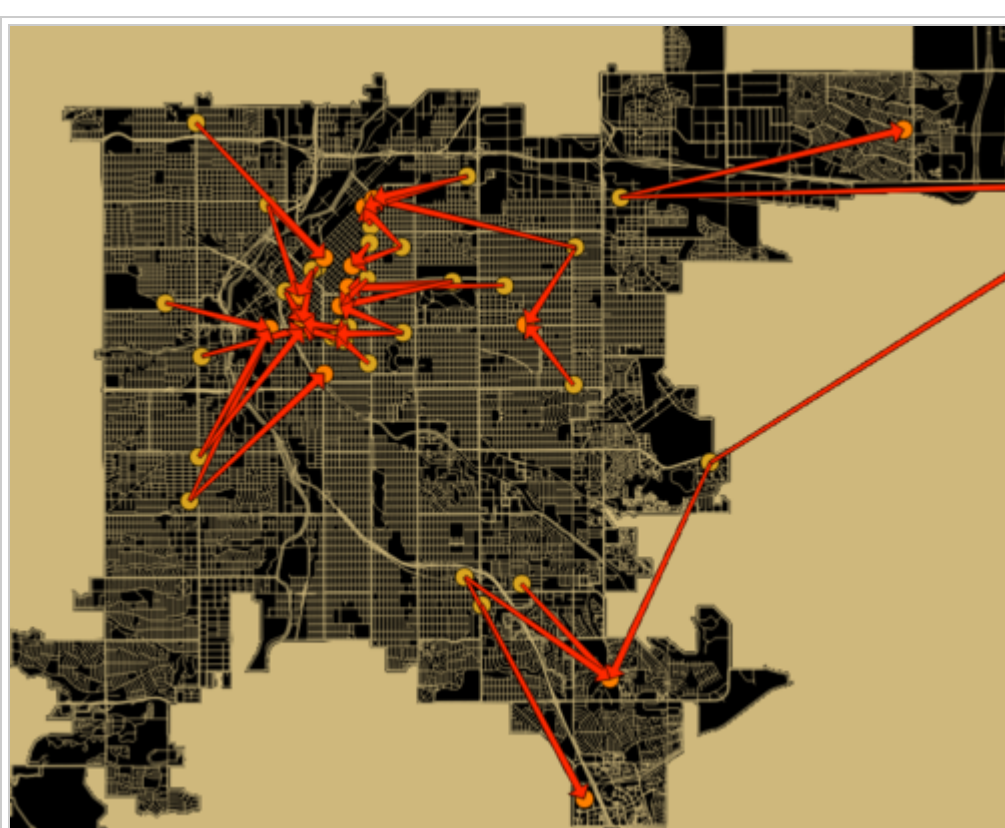
## Discussion

The use of an assignment problem creates a new tool for visualizing changes in criminal activity. Coupled with the associated statistics, the linear sum assignment model can provide new information on crime location.

## References

Burkard R.E., Çela E. (1999) Linear Assignment Problems and Extensions. In: Du DZ., Pardalos P.M. (eds) *Handbook of Combinatorial Optimization*. Springer, Boston, MA

Typesetting math: 100% ning People in Practice ([https://ampl.com/MEETINGS/TALKS/2011\\_01\\_Chiang\\_Mai\\_Plenary.pdf](https://ampl.com/MEETINGS/TALKS/2011_01_Chiang_Mai_Plenary.pdf)). Chiang Mai University International Conference.



Bias motivated crime in Denver for the years 2011 and 2014, linked by assignment. Yellow dots depict reported crimes 2011 and orange dots depict crimes in 2014.

Hate Crime Statistics Act. ([https://en.wikipedia.org/wiki/Hate\\_Crime\\_Statistics\\_Act](https://en.wikipedia.org/wiki/Hate_Crime_Statistics_Act))  
Wikipedia.

## Motivation

This is Kathleen Gatcliffe's Fall 2018 project for MATH 5593: Linear Programming taught by Steffen Borgwardt (<http://math.ucdenver.edu/~sborgwardt/>). This project uses the Hate Crimes (<https://www.denvergov.org/opendata/dataset/hate-crimes>) database from the Denver Open Data Catalog (<https://www.denvergov.org/opendata/>).

This project was performed in AMPL (<https://ampl.com/>), R (<https://www.r-project.org/>), and QGIS (<https://www.qgis.org/en/site/>). This project will be presented at the Auraria Library's Data to Policy (<https://library.auraria.edu/d2pproject>) event on the 30th of November, 2018.

## Resources

The code (<https://github.com/Kgatcliffe/DenverBiasMotivatedCrimes>) and other files produced for this project.

Project data (<https://www.denvergov.org/opendata/dataset/hate-crimes>), last collected 10 October, 2018.

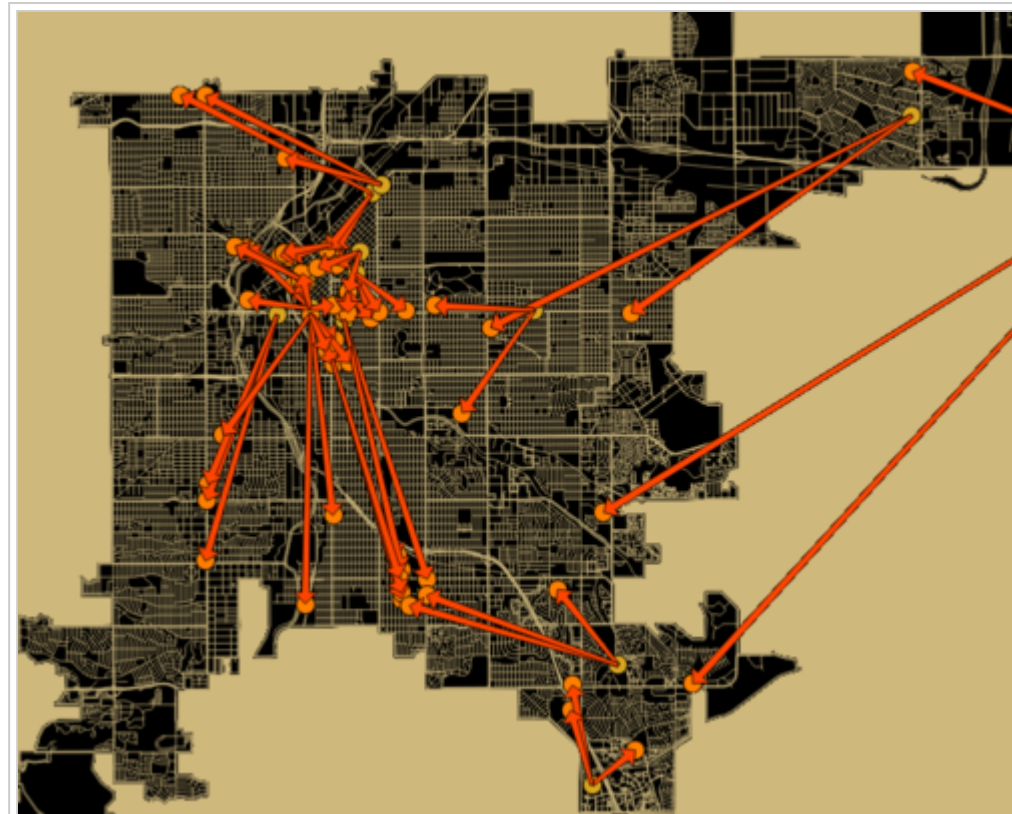
National yearly statistics summaries (<https://www.fbi.gov/investigate/civil-rights/hate-crimes#Hate-Crime%20Statistics>) from the Federal Bureau of Investigation.

Additional information on methods can be found at [Data Visualization Using QGIS and Denver Government Coordinate Systems](#)

Retrieved from "<https://math.ucdenver.edu/~sborgwardt/wiki/index.php?>

title=Denver\_Hate\_Crime\_Mapping:\_Visualizing\_Fluctuations\_through\_Linear\_Programming&oldid=1834"

- This page was last modified on 29 November 2018, at 23:06.
- This page has been accessed 6,415 times.



Bias motivated crime in Denver for the years 2014 and 2017, linked by assignment. Yellow dots depict reported crimes in 2014 and orange dots depict crimes in 2017.

# Main Page

From CU Denver Optimization Student Wiki

## Welcome to the CU Denver Optimization Student Wiki

The CU Denver Optimization Student Wiki is a resource for understanding introductory topics in mathematical programming and is intended to serve as an easy to understand online database for educators and students in the field of Optimization Research. The wiki is completely student-contributed under the direction of Dr. Steffen Borgwardt. A list of contributors can be found on the Contributors page. To begin learning either use the search bar or click here.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Main\\_Page&oldid=590](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Main_Page&oldid=590)"

- 
- This page was last modified on 19 November 2017, at 02:57.
  - This page has been accessed 208,653 times.

# Makayla Cowles

From CU Denver Optimization Student Wiki

Hello! I am currently a PhD student in the Mathematical and Statistical Sciences Department at the University of Colorado, Denver. I received my Bachelor's of Science in Mathematics at the University of Arkansas, Pine Bluff, and completed a Post-Baccalaureate at Iowa State University. Outside of school, I enjoy outdoor activities such as playing soccer, snowboarding, and camping. I also enjoy my fair share of indoor activities, including puzzles, playing board games, and watching a variety of T.V. shows.

During the Fall of 2020, I worked with Drew Horton and Michael Burgher on Making Voting More Accessible.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Makayla\\_Cowles&oldid=2913](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Makayla_Cowles&oldid=2913)"

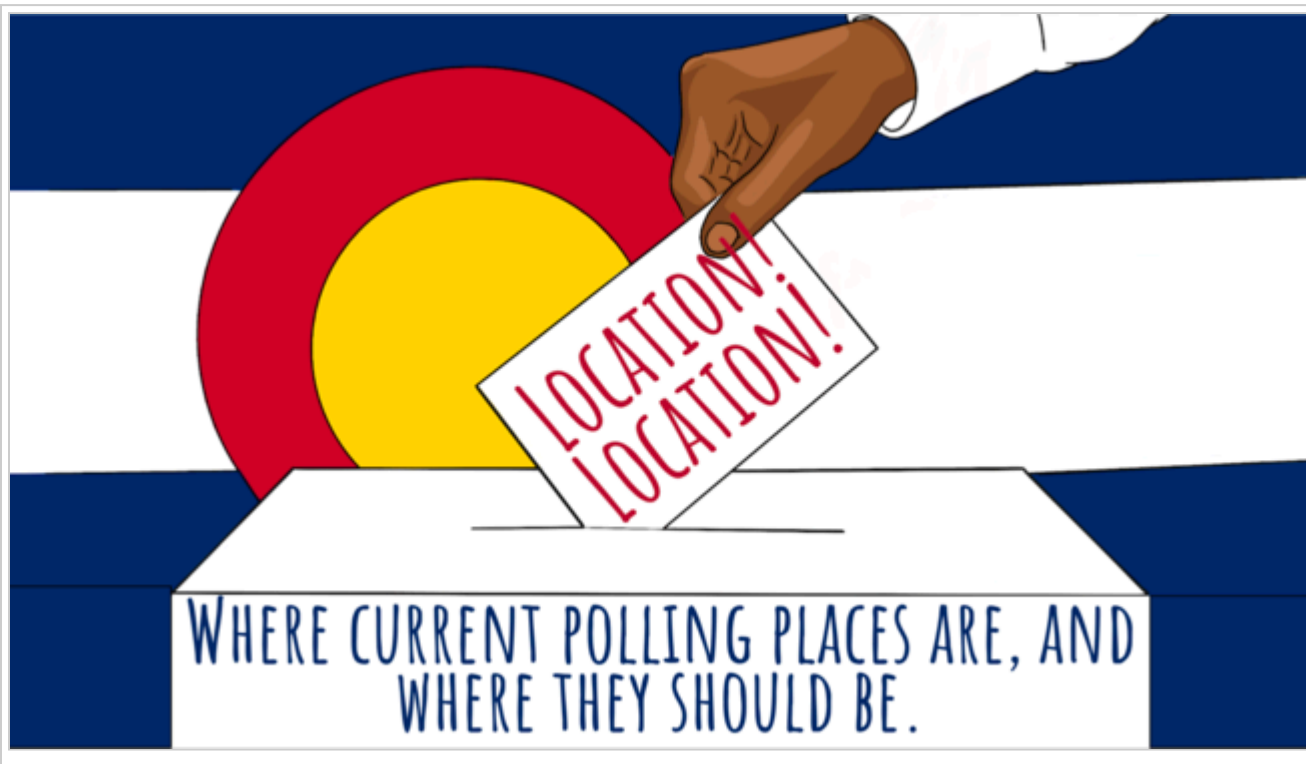
Category: Contributors

- 
- This page was last modified on 9 November 2020, at 18:52.
  - This page has been accessed 6,012 times.



# Making Voting More Accessible

From CU Denver Optimization Student Wiki



## Contents

- 1 Abstract
- 2 Data
- 3 Code
- 4 Results
- 5 Future Work
- 6 Presentation
- 7 References



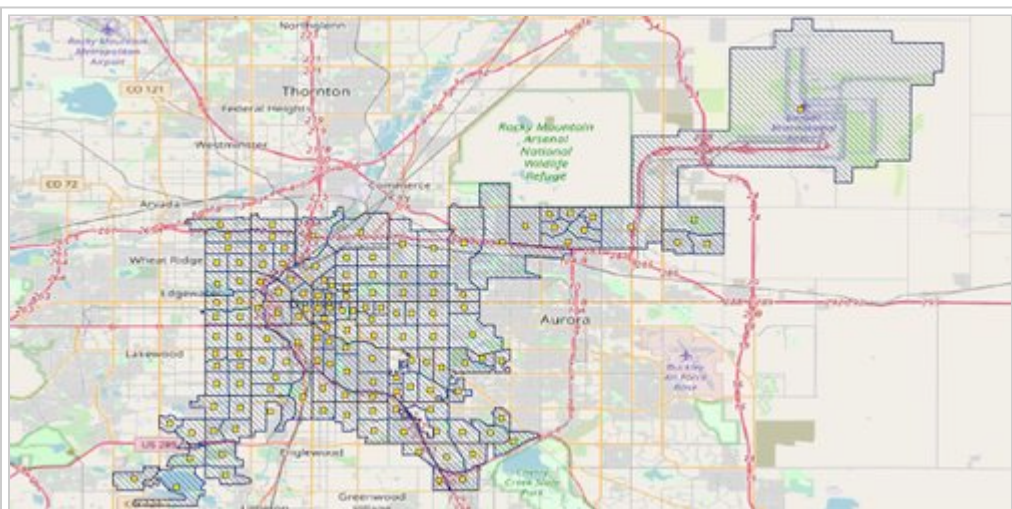
# Abstract

As a result of the civil rights movement of the 1960's, the US congress enacted the Voting Rights Act of 1965 to ensure that Americans who were eligible to vote were not prevented from doing so. In 2013, the US Supreme Court ruled section 4(b) of the Voting Rights Act unconstitutional in the case *Shelby County v. Holden* <sup>[1]</sup> because it was based on outdated data. Section 4(b) was a coverage formula that defined which states would need federal approval to enact laws regarding voting or make changes to elections. Since the US Supreme Court's ruling on *Shelby County v. Holden* more than 1688 polling places across 13 states have closed <sup>[2]</sup>. Our goal is to create a coverage formula that can be used by congress to replace the outdated formula in section 4(b) that was overruled in *Shelby County v. Holden*.

We created a linear program that assigns each census tract in a given county to a reasonable polling location based on the distance between the centroid of the census block and all potential polling locations. This will create an optimal distribution of polling places, which we will then compare to the current locations of polling places. Our program will run using data collected from Denver County, and because of the accessibility of voting locations in Denver, it is expected that our results will only have minor differences from the current polling locations. However, we will be able to use our program to identify other states and counties whose current polling locations greatly differ from the optimal distribution. The overall results of our project could serve as a model for fair placement of voting locations elsewhere and help provide a new coverage formula based on current data.

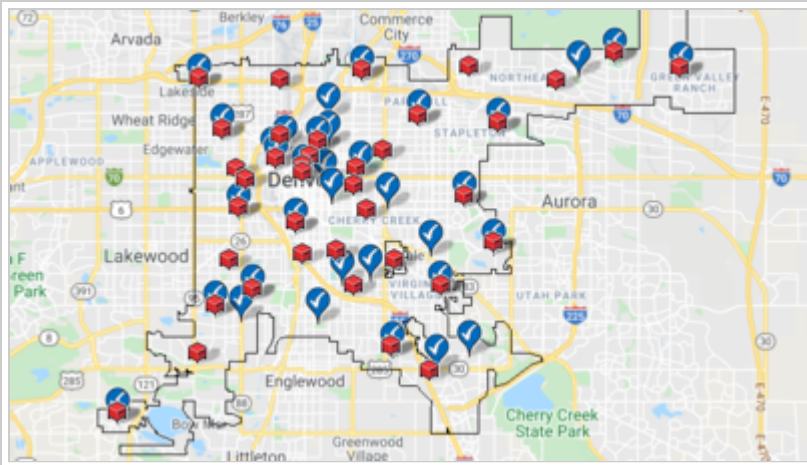
## Data

For our project we needed a list of potential polling locations, the current polling locations, and census data. Much of this data was easily accessible for the County of Denver. Census tracts data <sup>[3]</sup> is provided by the County of Denver. For our program we needed the centroids of these census tracts, which is not directly given from the above link. We were able to compute the centroids of the census tracts using QGIS (a free and open source geographical information system).



**Figure 1:**Census Track Centroids

For the 2020 presidential election, Denver provided maps of the locations of in person polling locations and drop off boxes (Figure 2). To find possible polling locations in Denver County, the national conference of state legislatures website gives restrictions for polling locations for each state <sup>[4]</sup>. The potential polling locations given are public buildings, libraries <sup>[5]</sup>, recreation centers <sup>[6]</sup>, fire stations <sup>[7]</sup>, K-12 schools, universities and colleges, senior centers, and churches <sup>[8]</sup>. Due to growing safety concerns of K-12 students, we opted out of using K-12 schools in our list of potential polling locations. Open Refine was used to clean up our data, while QGIS was used to visualize our data.



**Figure 2:**In-Person Polling locations (blue) and Drop-Off Boxes (red)

## Code

The first step is to input the data. This included single-valued parameters for the maximum amount of people that can be assigned to a polling location and the maximum amount of locations that can be used. This program also inputs data for 515 possible polling locations (PPL) and 144 census tracts (CB). Data for the possible polling locations include the latitude and longitude coordinates and an address. Data for the census tracts include the latitude and longitude coordinates of the centroid as well as the population.

To help define a function to optimize, we created a variable matrix called loc\_used (location used) with the rows corresponding to possible polling locations and the columns corresponding to census tracts. This matrix will show which tracts are assigned to which polling locations. The matrix will contain binary entries such that the  $(i, j)^{th}$  entry of this matrix will contain a 1 if the  $j^{th}$  tract is assigned to the  $i^{th}$  polling location and a 0 otherwise.

In general, we are trying to minimize the total amount of time people will need to spend getting to the polls. For our algorithm,  $l_{ij}$  is the  $(i, j)^{th}$  entry of loc\_used,  $p_i$  is the population in the  $i^{th}$  census tract,  $a$  and  $b$  are the latitude and longitude coordinates for the polling locations and centroids of tracts. Below is our objective function

$$\min : \sum_i^{515} \sum_j^{144} (p_j * l_{ij} * ((a_i - a_j)^2 + (b_i - b_j)^2));$$

We multiply each term by the population of the corresponding district and by the distance between the centroid of that district and the corresponding polling location. We multiply by the entries in loc\_used so that we only count terms where districts are assigned to locations. We multiply by the population to weight this so that outliers (especially populations of 0) do not affect the results too much. Future thought could be given to raising the population to a power (perhaps 0.5). The distance calculated is the square of the Euclidean distance. A different power could be considered in the future to let outliers have a different weight.

We then have the following three constraints:

1. Assignment: For each of the 144 Census tracts,  $\sum_i^{515} l_{ij} = 1$

Each district must be assigned to a location, so for each column, the sum of all column entries must be 1.

2. Max People: For each of the 515 locations,  $\sum_j^{144} l_{ij} * p_j \leq 15000$ .

Each district has a maximum amount of people it can handle so for each row, there is an upper bound on the sum of all row entries times corresponding populations.

3. Max Number of Location: No more than 50 polling locations can have tracts assigned to them,  $\sum_i^{515} \prod_j^{144} (1 - l_{ij}) \geq 515 - 50$ .

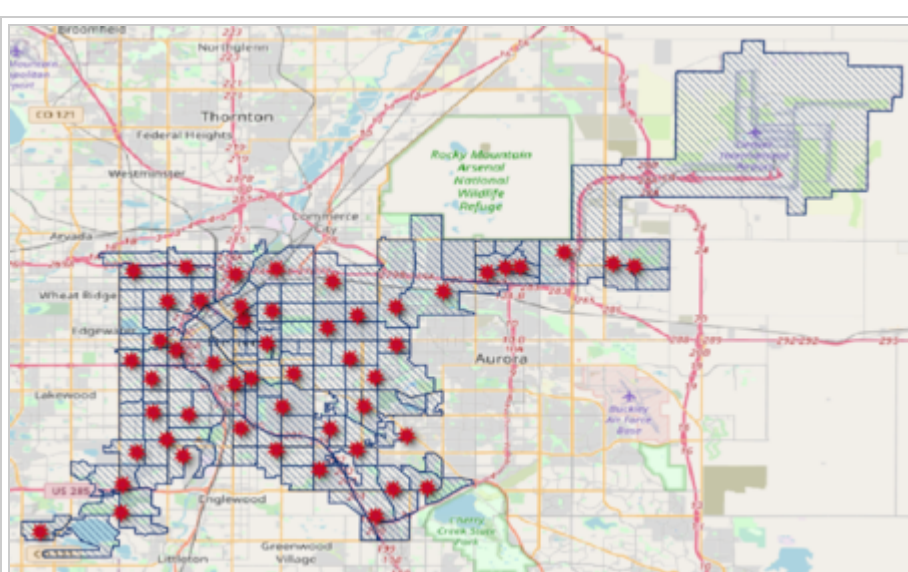
There is an upper limit on the number of locations that can be used. The number of unused locations correspond to the number of rows containing entirely zeros. The can be counted with this following sum over each row. For each row, count the product of 1 minus each row entry. This product will be 0 if there is a 1 in the row and 1 if the row contains entirely zeros. We set the upper bound of this sum as the number of polling locations we can use (50 in this case). It should also be noted that this third restraint is the reason we used tracts instead of blocks. There were over 11,000 blocks making our matrix have over 5 million variables! This was too much for AMPL to handle on our laptops.

The locations selected by the program are outputted onto a map of Denver.

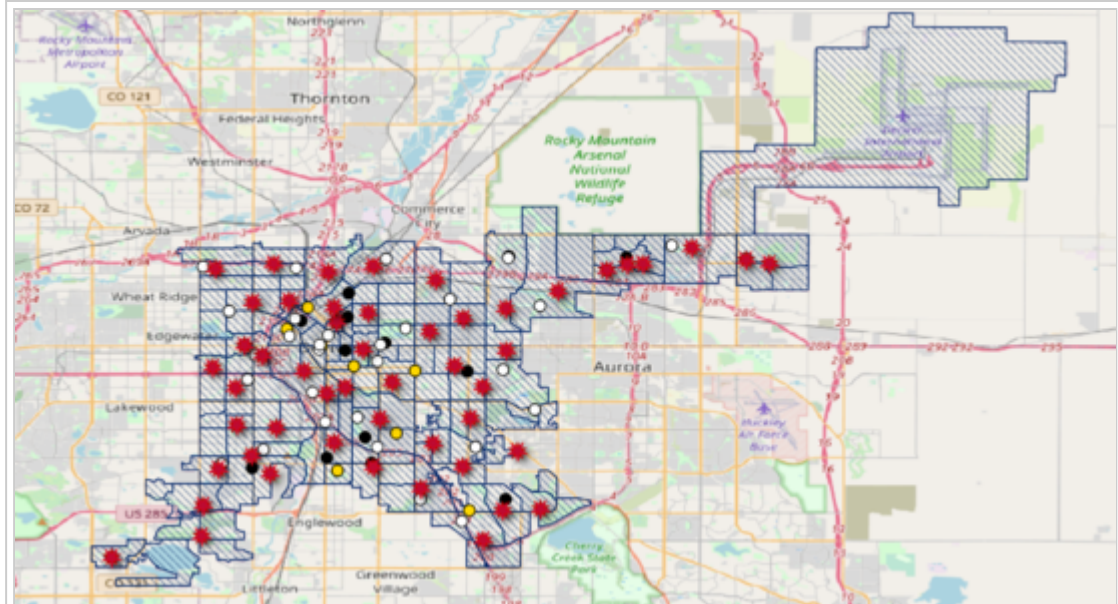
## Results

Figure 3 shows the optimal location of polling places in Denver County. the 2020 presidential election.

Figure 4 shows the optimal polling locations compared with the current polling locations for



**Figure 3:**Optimal Solution



**Figure 4:**Optimal Solution vs Current in-person polling locations (black), drop off box locations (white), hand in to person locations (yellow)

Notice that the solution gives a reasonable distribution of polling locations in Denver County. Also, it is seen that there a few optimal polling locations that are in close proximity to current in person locations.

The following table indicates which locations are included in our optimal solution for Denver County. The Bolded locations are those that were used as in person polling locations for the 2020 presidential election.

Libraries	Churches	Senior Living Facilities	Fire Stations	Recreation Centers
Central	Christ Community	Springbrooke Retirement	FS-29	Southwest Recreation Center
Athmar Park Branch	Church in Denver	Sunrise at Pinehurst	FS-2	
Ross-Barnum Branch	Church in South Denver	Balfour at Stapleton		
Bear Valley Branch	Greater Harvest Church of God	Brookdale University Park		
Blair-Caldwell African American Research	Greater St John Baptist	Broodale Parkplace		
Ross-Broadway Branch	Park Hill Congregational	Rosemark at Mayfair Park		
Byers Branch	Sacred Heart	Modena Cherry Creek		
Ross-Cherry Creek Branch	St Barnabas Episcopal	Jerusalem		
Decker Branch	St Patricks Catholic	Our House II Inc		
Eugene Field Branch	True Light Baptist	Volunteers of America Casa De Rosal		
Ford-Warren Branch	Templo Emmanuel Centro Cristiano de Alabanza Asambleas de Dios	Quincy Place		
Green Valley Ranch Branch	Denver North Park Foursquare			
Hadley Branch	New Beginnings Congregation			
Hampden Branch				
Montbello Branch				
Park Hill Branch				
Rodolfo "Corky" Gonzalez Branch				
Schlessman Family Branch				
Smiley Branch				
Ross-University Hills Branch				
Valdez-Perry Branch				
Virginia Village Branch				
Westwood Branch				
Woodbury Branch				



# Future Work

In the future we hope to expand this program in many different ways. We hope to gather data from every state, especially those that have historically disenfranchised voters of color. We would like to compute an optimal distribution with the given number of current polling places in each of the states and compare it with the current locations of polling places.

Some other immediate ideas are to give further discussion to raising the population and distance in the constraints to a different power. This would affect how big or little of a role the outliers will play.

More long-term goals would include thinking more about resources at polling locations. Including , but not limited to, the number of volunteers for a given jurisdiction, the number of voting machines, and the number of potential polling places. This could take the form of a program that assigns voting machines, workers, and ballots to locations.

## Presentation

Presentation slides and code ran can be found at the following link.

[https://github.com/MichaelBurgher/Linear-Programing\\_Project-Drew-Makayla/find/main](https://github.com/MichaelBurgher/Linear-Programing_Project-Drew-Makayla/find/main)

## References

1. ↑ <https://www.justice.gov/crt/section-4-voting-rights-act#formula>
2. ↑ <http://civilrightsdocs.info/pdf/reports/Democracy-Diverted.pdf>
3. ↑ <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-census-tracts-2010>
4. ↑ [polling-places.aspx](#)
5. ↑ [https://www.denverlibrary.org/location\\_proximity](https://www.denverlibrary.org/location_proximity)
6. ↑ <https://www.denvergov.org/content/denvergov/en/denver-parks-and-recreation/recreation-centers-pools/recreation-centers/find-a-recreation-center.html>
7. ↑ <https://www.denvergov.org/content/denvergov/en/fire-department-home/contact-us/fire-stations.html>
8. ↑ <https://www.churchfinder.com/churches/co/denver>

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Making\_Voting\_More\_Accessible&oldid=3079"

- This page was last modified on 1 December 2020, at 13:30.
- This page has been accessed 2,589 times.

# Malik Odeh

From CU Denver Optimization Student Wiki

I am a graduate student at the University of Colorado Denver studying applied math. I completed my undergraduate education at Boston University with a focus in dynamical systems and their continuous models. My favorite past projects include modeling and altering the mammalian circadian rhythm using Python, classifying neuronal bursting patterns using bifurcation analyses, and exploring oscillatory behaviors in glycolytic pathways.

During my studies at CU Denver, I plan to take courses in PDEs, linear algebra, and statistics to prepare myself for a career in mathematical modeling and analysis.

My passions outside of mathematics include strength training and critical analyses of high level, high octane first-person-shooter video games.

- Contributions

StarCraft II Build Order Optimization

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Malik\\_Odeh&oldid=887](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Malik_Odeh&oldid=887)"

Category: Contributors

- 
- This page was last modified on 7 December 2017, at 03:44.
  - This page has been accessed 1,643 times.

# Mapping Accident Prone Intersections

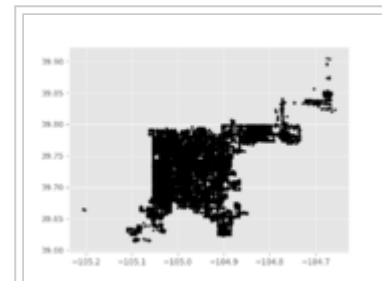
From CU Denver Optimization Student Wiki

*The following is an entry on a project completed by Lauren Hearn in Fall, 2018*

Traffic flow patterns and accidents are a concern that anyone in a city encounters every day. Clustering methods are an important tool when analyzing traffic accidents, as these methods are able to identify groups of road users and segments which could be suitable targets for countermeasures. Ideally, cluster analysis is a statistical technique that can be used to group items together on the basis of similarities or dissimilarities.<sup>[1]</sup>

## Contents

- 1 Inspiration and Method
- 2 Model Overview
  - 2.1 Data
- 3 Results
- 4 Goals & Future Work
- 5 References



Unclustered accident data for  
Denver, CO 2017-2018

## Inspiration and Method

Denver, CO has a reputation for bad drivers, with varying opinions on which intersections are more prone to accidents and why. Using accurate city data, I wanted to explore the different "accident densities" when splitting the city into various sized sub-areas.

Originally, the project involved using K-Means Clustering,<sup>[2][3]</sup> however, it was eventually decided to take a Linear Programming approach, modifying the LP model discussed in the paper, "An LP-based k-means algorithm for balancing weighted point sets".<sup>[4]</sup>

A combination of AMPL (student version),<sup>[5]</sup> Python (version 2.7),<sup>[6]</sup> and the AMPL API, which can be easily accessed through the opensource amplpy<sup>[7]</sup> package was used, initially. This presented challenges in data formatting (a common problem), which lead to a change of tools. The Pyomo<sup>[8][9]</sup> optimization tools package for python was then used to code the model. Like AMPL, Pyomo does not come with native solvers, so the glpk<sup>[10]</sup> solver was chosen, predominately due to its being opensource as well as utilized in many online examples of Pyomo models.



## Model Overview

All relevant code for the model can be easily accessed on github (<https://github.com/mandlebrot/traffic-accidents>). this oncludes the original AMPL .mod file, as well as the altered Python script, Traffic.py.

## Data

Using data available at [denvergov.org/opendata](http://denvergov.org/opendata) and common optimization tools in Python, a linear program was written to assist in looking for various “hot-spots” of traffic accidents. In order to accomplish this, the data was partitioned into clusters using geolocation data and cluster size bounds to compare the density of accidents per partition.

## Results

## Goals & Future Work

The goal of this project is to better understand what areas are especially prone to accidents, and why. Analyzing and comparing this data along with other factors, including one-way roadways, traffic signals, and high-use thoroughfares should supply a pathway for both the city administration and the police force to better plan updates to infrastructure and presence, respectively.

## References

1. ↑ Geurts, Karolien & Wets, Geert & Brijs, Tom & Vanhoof, Koen. (2010). Clustering and profiling traffic roads by means of accident data.
  2. ↑ <https://mubaris.com/posts/kmeans-clustering/>
  3. ↑ <http://benalexkeen.com/k-means-clustering-in-python/>
  4. ↑ S.Borgwardt & A.Brieden & P.Gritzmann. (2017). An LP-based k-means algorithm for balancing weighted point sets. *European Journal of Operational Research*. Volume 263, Issue 2, Pages 349-355
  5. ↑ <https://ampl.com>
  6. ↑ <https://www.python.org/>
  7. ↑ <https://pypi.org/project/amplpy/>
  8. ↑ Hart & William E. & Jean-Paul Watson & David L. Woodruff. (2011). Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation* 3. no. 3: 219-260.
  9. ↑ Hart, William E. & Carl Laird & Jean-Paul Watson & David L. Woodruff & Gabriel A. Hackebeil & Bethany L. Nicholson & John D. Siirola. (2017). Pyomo – Optimization Modeling in Python. *Springer*.
  10. ↑ <https://www.gnu.org/software/glpk/>
- Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Mapping\\_Accident\\_Prone\\_Intersections&oldid=1658](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Mapping_Accident_Prone_Intersections&oldid=1658)"

- This page has been accessed 2,223 times.

# Mapping Accident Prone Regions

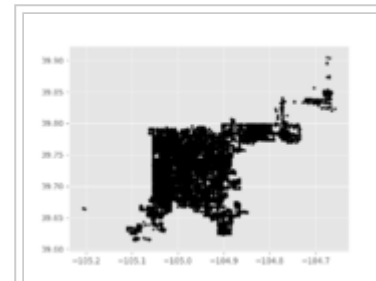
From CU Denver Optimization Student Wiki

*The following is an entry on a project completed by Lauren Hearn in Fall, 2018*

Traffic flow patterns and accidents are a concern that anyone in a city encounters every day. Clustering methods are an important tool when analyzing traffic accidents, as these methods are able to identify groups of road users and segments which could be suitable targets for countermeasures. Ideally, cluster analysis is a statistical technique that can be used to group items together on the basis of similarities or dissimilarities.<sup>[1]</sup>

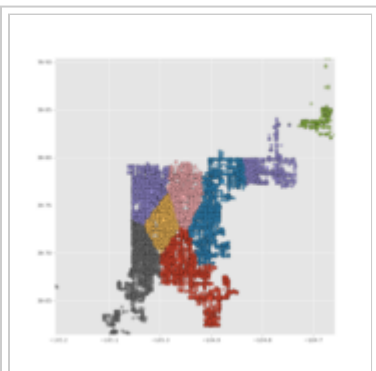
## Contents

- 1 Motivation and Method
- 2 Model Overview
  - 2.1 Data
- 3 Results
- 4 Goals & Future Work
- 5 References



Unclustered accident data for Denver, CO 2017-2018

## Motivation and Method



Accident data clustered with K-Means Algorithm

Denver, CO has a reputation for bad drivers, with varying opinions on which intersections are more prone to accidents and why. Using accurate city data, I wanted to explore the different "accident densities" when splitting the city into various sized sub-areas.

Originally, the project involved using K-Means Clustering,<sup>[2][3]</sup> for which there are common packages in Python available. However, this approach simply clusters data by geolocation, and for this particular project it was decided to choose clusters with an equal number of data points. The K-means approach was then modified to a Linear Program using the LP model discussed in the paper, "An LP-based k-means algorithm for balancing weighted point sets"<sup>[4]</sup> and setting all weights equal, as the problem does not benefit from weighting at this stage.

A combination of AMPL (student version),<sup>[5]</sup> Python (version 2.7),<sup>[6]</sup> and the AMPL API, which can be easily accessed through the opensource amplpy<sup>[7]</sup> package was used, initially. This presented challenges in data formatting (a common problem), which lead to a change of tools. The Pyomo<sup>[8][9]</sup> optimization tools package for python was then used to code the model. Like AMPL, Pyomo does not come with native solvers, so the glpk<sup>[10]</sup> solver was chosen, predominately due to its being opensource as well as utilized in many online examples of

Pyomo models.

# Model Overview

The Linear Program used is as follows:

$$\min \sum_{i=1}^k \sum_{j=1}^n y_{ij} (x_j^T x_j - 2 * x_j^T s_i + s_i^T s_i)$$

$$\frac{n^-}{k} \leq \sum_{j=1}^n y_{ij} \leq \frac{n^+}{k} \quad \text{Limits cluster size}$$

$$\sum_{i=1}^k y_{ij} = 1 \quad \text{Each point is assigned to exactly one cluster}$$
$$y_{ij} \geq 0$$

Where  $y$  is an assignment variable,  $s$  is the chosen site or "centroid" that the data is clustered around, and  $x$  is the location vector.

All relevant code for the model can be easily accessed on github (<https://github.com/mandlebrot/traffic-accidents>). this includes the original AMPL .mod file, as well as the altered Python script, Traffic.py. In addition, a poster presented at the 2018 Data to Policy Symposium can be found on this github page.

## Data

Using the “Traffic Accidents” dataset from the Denver Open Data Catalog (<https://www.denvergov.org/opendata/>) and common optimization tools in Python, a linear program was written to assist in looking for various “hot-spots” of traffic accidents. In order to accomplish this, the data was partitioned into clusters using geolocation data and cluster size bounds to compare the density of accidents per partition.

## Results

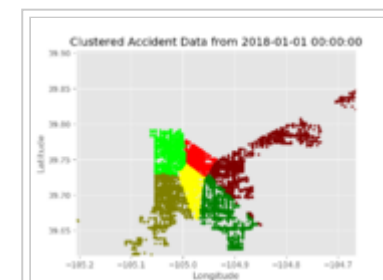
With the limited computing power available, clear results were attainable by limiting data to smaller time periods.

Cursory visual analysis reveals an unsurprising trend. We see higher densities of accidents in centrally-located downtown neighborhoods, while seeing more spread-out clusters (lower accident density) as you move further from the city-center.

## Goals & Future Work

The goal of this project is to better understand what areas are especially prone to accidents, and why. Analyzing and comparing this data along with other factors, including one-way roadways, traffic signals, and high-use thoroughfares should supply a pathway for both the city administration and the police force to better plan updates to infrastructure and presence, respectively.

Future work to do:



Clustered accident data with modified algorithm

- Narrow data to smaller areas to observe specific intersection accident density
- Get code to iterate and improve centroid locations automatically
- Run with CPLEX<sup>[11]</sup> solver (more efficient for Mixed-Integer LPs)

## References

1. ↑ Geurts, Karolien & Wets, Geert & Brijs, Tom & Vanhoof, Koen. (2010). Clustering and profiling traffic roads by means of accident data.
  2. ↑ <https://mubaris.com/posts/kmeans-clustering/>
  3. ↑ <http://benalexkeen.com/k-means-clustering-in-python/>
  4. ↑ S.Borgwardt & A.Brieden & P.Gritzmann. (2017). An LP-based k-means algorithm for balancing weighted point sets. *European Journal of Operational Research*. Volume 263, Issue 2, Pages 349-355
  5. ↑ <https://ampl.com>
  6. ↑ <https://www.python.org/>
  7. ↑ <https://pypi.org/project/amplpy/>
  8. ↑ Hart & William E. & Jean-Paul Watson & David L. Woodruff. (2011). Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation* 3. no. 3: 219-260.
  9. ↑ Hart, William E. & Carl Laird & Jean-Paul Watson & David L. Woodruff & Gabriel A. Hackebeil & Bethany L. Nicholson & John D. Sirola. (2017). Pyomo – Optimization Modeling in Python. *Springer*.
  10. ↑ <https://www.gnu.org/software/glpk/>
  11. ↑ <https://www.ibm.com/products/ilog-cplex-optimization-studio>
- Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Mapping\\_Accident\\_Prone\\_Regions&oldid=1729](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Mapping_Accident_Prone_Regions&oldid=1729)"

- 
- This page was last modified on 27 November 2018, at 11:29.
  - This page has been accessed 4,453 times.

# Marie Terry

From CU Denver Optimization Student Wiki

Marie Terry is a graduate student studying Applied Mathematics at the University of Colorado Denver. She received her undergraduate degree from the University of Wyoming majoring concurrently in Secondary Education Mathematics and Mathematics. She hopes to teach at a junior college when she finishes her degree. Marie is currently an adjunct instructor at Arapahoe Community College. When not studying or working, Marie enjoys knitting, hiking, and spending every waking minute with her husband and amazing twin boys.

Marie's contribution to the wiki:

Column Generation

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Marie\\_Terry&oldid=841](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Marie_Terry&oldid=841)"

Category: Contributors

- 
- This page was last modified on 6 December 2017, at 20:21.
  - This page has been accessed 25,970 times.

# MarkdownExamples

From CU Denver Optimization Student Wiki

This is a page where we can put example markdown (wiki) code. Please create a new section for each category of codes.

## Contents

- 1 Citations
  - 1.1 Inline citation
  - 1.2 References List
- 2 Source Code

## Citations

### Inline citation

Here is a citation. <sup>[1]</sup> It is important to put a references list tag at the bottom of the page.

### References List

1. ↑ Great Citation. www.citation.com. 2009. Pearson

## Source Code

You must use the "source" tag. The code looks like this:

```
a = "hello world"
print(a)
```

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=MarkdownExamples&oldid=220"

- This page was last modified on 16 March 2017, at 14:56.

- This page has been accessed 4,553 times.



# Maryam Khazaei

From CU Denver Optimization Student Wiki

I am a graduate PHD student at the University of California Merced. I got my Master Degree in Applied Math (PDE) from UC Denver and my undergraduate degree in Applied Mathematics from IUST that I was honored with the Outstanding Student Award in my Bachelor's degree.

I have always believed that application of mathematics and Computer Science are one of the important factor in life. During university years, I could devote many years of intensive academic focus to this field which provided me the strong foundation in all subjects and exposure to practical industrial applications through field studies. My research interests are in the Applied Mathematics, Computer Graphic, Max Flow Problem and Robotics.

I worked on “ Spline collocation method for solution of higher order linear boundary value problems” (<http://www.twmsj.az/Files/Contents%20V.6,%20N.1,%202015/pp38-47.pdf>) and "Numerical solution of two dimensional coupled viscous Burgers equation using modified cubic B-spline differential quadrature method" as my master thesis at University of Colorado Denver. Currently I work on Multi Agent Path finding, Robotics and application of Max Flow Problems at UC Merced.

My contribution to this wiki: Minimum Spanning Tree: specifically Kruskal’s algorithm, Prim’s algorithm and Boruvka algorithm.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Maryam\\_Khazaei&oldid=2510](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Maryam_Khazaei&oldid=2510)"

Category: Contributors

- 
- This page was last modified on 24 March 2020, at 01:23.
  - This page has been accessed 2,113 times.

# Matching in General Graphs

From CU Denver Optimization Student Wiki

## Contents

- 1 Project Contributions
- 2 Abstract
- 3 Github
- 4 Bibliography

## Project Contributions

This project was completed by Micah Grip.

## Abstract

In this project we explore the topic of matching in general graphs by first discussing the difficulties with such matching problems. In order to show Hall's Condition does not provide an equivalent condition for the existence of a perfect matching in general graphs, we discuss  $C_3$  as an example of a general graph satisfying Hall's Condition which has no perfect matching. Prior to presenting a proof of Tutte's Theorem, we introduce the necessary terminology for such a discussion which includes  $k$ -factors of a graph, Tutte's Condition, the deficiency of a vertex set, and a Tutte set. During the introduction of new terminology, examples of **1** and **3** factors are presented, and efforts are made to draw similarities between matching in general and bipartite graphs. The proof of Tutte's Theorem establishes Tutte's Condition as a necessary and sufficient condition for the existence of a **1** factor in a general graph. Similarly to how Ore's Defect formula provides the size of a maximum matching in a bipartite graph, we prove the Berge-Tutte formula which provides the size of a maximum matching in a general graph. Two lemmas, which serve as stepping-stones to the proof of the Berge-Tutte formula, are presented with one being proved and the other being only stated.

## Github

In the Github repository linked below, the slide deck for the project presentation can be found. The slide deck contains definitions, examples, and detailed proof outlines for the theorems mentioned above.

Matching in General Graphs (<https://github.com/MicahGrip751/Matching-in-General-Graphs/tree/main>)

# Bibliography

- West, D. B. (2021). Combinatorial mathematics. Cambridge University Press.
- Diestel, Reinhard. Graph Theory. Springer, 2025.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Matching\\_in\\_General\\_Graphs&oldid=4992](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Matching_in_General_Graphs&oldid=4992)"

---

- This page was last modified on 3 May 2025, at 15:42.
- This page has been accessed 24 times.

# Matesi Gregory

From CU Denver Optimization Student Wiki

I am a fourth year undergraduate in the Applied Math Program at CU Denver. I am originally from Sandwich, IL. After spending four years in the USMC I decided to move to Denver.. After I graduate in December or 2019 I hope to continue at CU Denver in the MS in Statistics Program. I am currently working with Siyuan\_Lin on a project examining the gradient descent method for convex quadratic problems in AMPL: Gradient Descent Method in Solving Convex Quadratic Optimization Problems

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Matesi\\_Gregory&oldid=2127](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Matesi_Gregory&oldid=2127)"

- 
- This page was last modified on 10 November 2019, at 19:10.
  - This page has been accessed 409 times.

# Matthew Knodell

From CU Denver Optimization Student Wiki

## About Me

Hello! My name is Matthew Knodell. I am an Applied Math PhD student at UC Denver. I am interested in applications of mathematics to machine learning and optimization.

## Education

I received my Bachelor's in Mathematics and my Bachelor's in Physics from St. Mary's University in May of 2018. From there, I received my Master's in Mathematics from UT Dallas in December 2021.

## Projects

Fall 23 - Optimizing SAR Paths in the Rocky Mountain Region, project with Lillian makhoul and Nicholas Rogers.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Matthew\\_Knodell&oldid=4451](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Matthew_Knodell&oldid=4451)"

Category: Contributors

- 
- This page was last modified on 14 November 2023, at 09:08.
  - This page has been accessed 29 times.

# Megan Duff

From CU Denver Optimization Student Wiki

I started my statistics PhD at CU Denver in the Fall of 2018 after graduating from Willamette University in Salem, OR with a BA in Mathematics. When I am not in the classroom, I enjoy doing aerial dance, yoga, and going to the movies.

For my project, I will be working with Rebecca Robinson on presenting An Integer Linear Programming Approach to Graph Coloring.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Megan\\_Duff&oldid=2308](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Megan_Duff&oldid=2308)"

Category: Contributors

- 
- This page was last modified on 30 November 2019, at 16:43.
  - This page has been accessed 1,037 times.

# Micah Grip

From CU Denver Optimization Student Wiki

## About Me

I am currently a first-year graduate student. I did my undergrad at CU Denver and returned for graduate school after taking a gap year. I like to spend my free time reading.

## Projects

Spring 2025 (Applied Graph Theory Final Project) - Matching in General Graphs

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Micah\\_Grip&oldid=4872](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Micah_Grip&oldid=4872)"

Category: Contributors

- 
- This page was last modified on 16 April 2025, at 14:24.
  - This page has been accessed 28 times.

# Michael Boyce

From CU Denver Optimization Student Wiki

## About Me

I am PhD student a CU Denver. I am interested in graph theory. In my free time I like to play video games, and bool.

## Projects

### Applied Graph Theory, Spring 2025

Alex Semyonov and I created a project on the B.E.S.T theorem (van Aardenne-Ehrenfest, de Bruijn, Tutte-Smith) on how to count Eulerian cycles in digraphs. [1]  
([https://math.ucdenver.edu/~sborgwardt/wiki/index.php/Counting\\_Eulerian\\_Cycles\\_in\\_Graphs](https://math.ucdenver.edu/~sborgwardt/wiki/index.php/Counting_Eulerian_Cycles_in_Graphs))

## Contact information

If you would like to contact me, please email me at [michael.boyce@cudenver.edu](mailto:michael.boyce@cudenver.edu)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Michael\\_Boyce&oldid=5029](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Michael_Boyce&oldid=5029)"

Category: Contributors

- 
- This page was last modified on 12 May 2025, at 16:17.
  - This page has been accessed 18 times.



# Michael Burgher

From CU Denver Optimization Student Wiki

This is the page for Michael Burgher!!!

I am currently working on a project for Topics in Optimization (Fall 2021) that considers a Greedy Algorithm to find the largest stable set in a graph. The goal of the project is to determine whether or not the algorithm utilizes circuit walks. The name of the project is Circuit Walks and Stable Sets.

Sandra Robles, Collin Powell and I all worked on a project to optimize herd immunity through effective Vaccine Distribution (Spring 2021).

Makayla Cowles, Drew Horton and I all worked on fighting voter suppression through our project Making Voting More Accessible. (Fall 2020)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Michael\\_Burgher&oldid=3456](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Michael_Burgher&oldid=3456)"

Category: Contributors

- 
- This page was last modified on 29 November 2021, at 13:28.
  - This page has been accessed 973 times.

# Michael Phillips

From CU Denver Optimization Student Wiki

After spending his childhood in Montana and his adolescent years in Michigan, Michael studied Mathematics at Grand Valley State University in Allendale, Michigan before moving back West for his graduate studies. He accepted a Teaching Assistantship at the University of Montana in Missoula where he focused on Extremal Graph Theory under the advisement of Dr. Cory Palmer. After completing his M.A. in Mathematics, he accepted another Teaching Assistantship at the University of Colorado Denver where he now studies Graph Theory, Probability and Statistics.

Current projects include modeling familial structure in criminal history databases with Dr. Stephanie Santorico, flag algebras and cycle inducibility with Dr. Florian Pfender, and zero-forcing polynomials with collaborators at GRWC 2017.

One peek into his office will tell you most everything you need to know about him: he reads too much, cares too much about Marvel and DC comic book characters, and could stand to straighten up his workspace a bit. What was that line about a cluttered desk and a cluttered mind?

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Michael\\_Phillips&oldid=614](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Michael_Phillips&oldid=614)"

Category: Contributors

- 
- This page was last modified on 27 November 2017, at 21:02.
  - This page has been accessed 2,311 times.

# Michael Schmidt

From CU Denver Optimization Student Wiki

## Contents

- 1 Contact Links
- 2 About Me
- 3 Education
- 4 Projects
  - 4.1 Finding Optimal Shared Streets in Denver
- 5 Converting Disused Lots to Housing: Equitable locations for new housing



## Contact Links

- Email (<mailto:michael.t.schmidt@ucdenver.edu>)

## About Me

Howdy, I am Mike (Michael) Schmidt, and I am a Ph.D. student at the University of Colorado Denver in the Applied Mathematics Department. I have a BA in Math and an MS in Physics from the University of Colorado at Boulder. My research interests are focused on MCMC and uncertainty-quantification. In addition to my graduate studies, I work for the company Redpoll (<https://redpoll.ai/>) for which I am engaged with several DARPA grants; specifically SAIL-ON (<https://www.darpa.mil/news-events/2019-02-14>) and ECoSystemic (<https://sam.gov/opp/4a697296778a4d96aaca850679f67059/view>).

## Education

1. CU Boulder, MS in Physics
2. CU Boulder, BA in Math & Physics

## Projects

### Finding Optimal Shared Streets in Denver

Together with Evan Shapiro and Em Gibbs, we investigated which roads the city could convert within the city to pedestrian walkways and how those closures would change the maximum flow of traffic around the closure. To learn more, see the project page: [Finding Optimal Shared Streets in Denver](#).

# Converting Disused Lots to Housing: Equitable locations for new housing

Denver has thousands of acres of disused and vacant land that could be used for housing. This project seeks to discover which plots of land are the best candidates for conversion with considerations to equity.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Michael\\_Schmidt&oldid=4324](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Michael_Schmidt&oldid=4324)"

Category: Contributors

---

- This page was last modified on 28 April 2023, at 21:53.
- This page has been accessed 390 times.

# Min-Mean Cycle Cancelling Algorithm

From CU Denver Optimization Student Wiki

## Abstract

## Files and Presentation

Presentation slides and code ran can be found at the following link. [1] (<https://github.com/pgmath/Min-Mean-Cycle-Cancelling-Algorithm>)

## Contributors

Paul Guidas

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Min-Mean\\_Cycle\\_Cancelling\\_Algorithm&oldid=4636](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Min-Mean_Cycle_Cancelling_Algorithm&oldid=4636)"

- 
- This page was last modified on 12 April 2024, at 10:35.
  - This page has been accessed 16 times.

# Min-mean Cycle Cancelling Algorithm and It's Applications

From CU Denver Optimization Student Wiki

## Contents

- 1 Abstract
- 2 Introduction
- 3 Applications
- 4 Overview of Algorithm
- 5 Finding Minimum-Mean Cycles
- 6 Number of Iterations
  - 6.1 Structural Properties
  - 6.2 Weakly Polynomial Time Algorithm
  - 6.3 Strongly Polynomial Time Algorithm
- 7 References And Powerpoint

## Abstract

The min-mean cycle canceling algorithm is an algorithm that improves upon the general cycle canceling algorithm. The idea of the min-mean cycle is that the algorithm places preference on cycle that are shorter in length. There are several application of the min-mean cycle canceling algorithm, including min-cost flow and maximum flow problems.

Ideally, we would be able to pick a cycle that improves our solution the most. Solving this type of problem is NP-complete, which means it is computationally difficult to find this cycle. The running time of the min-mean cycle canceling algorithm is much more efficient. We concluded this project by showing that the min-mean cycle canceling algorithm runs in weakly polynomial time. Specifically, it runs in  $\mathcal{O}(n^2 m^2 \log(nC))$  time.

## Introduction

Cycle Cancelling Algorithms are used to improve feasible solutions. It does this by sending the flow on a negative cycle in the residual network, and thus, improves the feasible solution. If we iterate the algorithm enough times, we are guaranteed to get a feasible solution. Now, there are several ways for us to pick which cycles to cancel. A generic cycle cancelling algorithm finds any negative cycle and send flow through it. Ideally, we would pick the negative cycle that improves our feasible solution the most. Unfortunately, finding this cycle is NP-Complete, a call of problems that tends to be quite difficult to solve. The min-mean cycle cancelling algorithm provides a method to pick a cycle that is a way to improve open the generic cycle cancelling algorithm, but isn't quite as computationally difficult to find.

The min-mean cycle canceling algorithm places emphasis on cycles that have fewer amounts of arcs. For example, If there are two negative cycles A and B that have a total cost of -12. Let cycle A have 4 arcs, and cycle B with 3 arcs. The min-mean cycle cancelling algorithm will choose to send flow through cycle B over cycle A.

Figure 1-3 show a more specific example. Figure 1 displays a feasible of length 40. Figure 2 shows what a generic cycle cancelling algorithm produce after sending flow through the cycle nodes 2, 3, 7, and 8. This new feasible solution now has a total cost of 23. Figure 3 shows what would happen if we applied the min-mean cycle cancelling algorithm.

## Applications

The main type of problem that the cycle canceling algorithm can be applied to is the min-cost flow problem. Problems that fall into the min-cost flow include the matching problem, the transportation problem, and the traveling salesmen problem. For some of this problems we might have to alter the network on which these problems exist in order for it to be possible for us to find negative cycles. Matching problems tend to be formed on bipartite networks, which, by definition, contain no cycles. In order to allow the network to have cycles, we can include a source node that connects to one side of the bipartite graph and a sink nodes that connects to the other side of the bipartite graph, and then connecting the sink node to the source node. This creates a network that can now contain cycles, for which we can apply our cycle cancelling algorithm to.

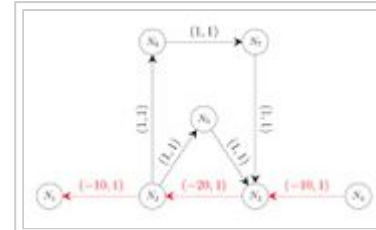
The traveling salesman problem is a problem that usually involves a complete network. This type of problem is a perfect candidate for the cycle cancelling algorithm, as a complete graph contains the most cycles of any type of simple graph. To see how a cycle could improve a feasible solution in a TSP, see Figures 4-6. Figure 4 shows a feasible solution, Figure 5 shows a potential negative cycle that could be cancelled. Figure 6 shows the new feasible after applying flow to the feasible cycle.

Additionally, cycle canceling can be applied to max-flow problems. We use the exact same process that we would on the min-cost flow problems, except instead of trying to find negative costs, we try to find positive costs. If we find a positive cost cycle on a network, we send flow through this cycle and we are able to increase the flow on our feasible solution. Thus, we improve our solution.

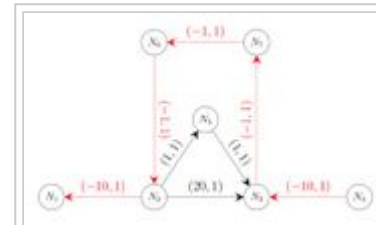
There are several applications that involve finding a maximum flow on a network. These examples include finding a system of distinct representative and the matrix rounding problem. It is worth mentioning that in order to find a cycle in either of these applications, there needs to be alterations made to the original network. These alterations are fairly simple and although the optimal solution to the new network will be different from the optimal solution in the original network, the arcs in the original network will be the same arcs that are found in the optimal solution of the new network.

## Overview of Algorithm

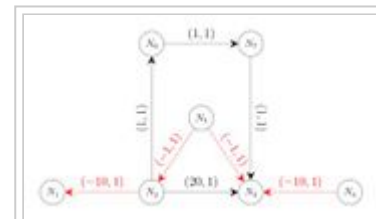
The algorithm is based on the optimality condition that given some feasible flow, the flow is optimal if and only if there are no negative cycles in the residual network. This is because otherwise we could send flow through this cycle (without changing the feasibility of the flow) while decreasing the total cost. A generic cycle cancelling algorithm then finds a negative cycle, updates the network, and repeats until there are no negative cycles. This runs in  $\mathcal{O}(mnCU)$  time, where  $C$  is the maximum cost of any edge, and  $U$  is the maximum capacity of any edge, assuming integer costs and capacities. The run time comes from a time of  $\mathcal{O}(mn)$  to find a cycle and  $\mathcal{O}(CU)$  iterations. This is a good algorithm if we have constant integer costs and capacities, however, we can improve this generic algorithm to actually give a strongly polynomial time algorithm.



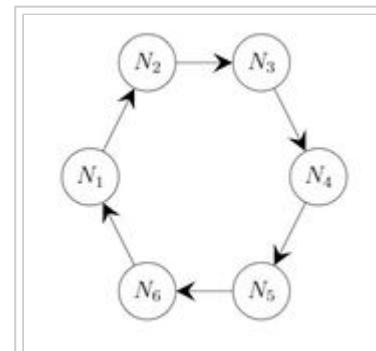
**Figure 1:** Feasible Solution



**Figure 2:** Feasible Solution



**Figure 3:** Feasible Solution



**Figure 4:** Feasible Solution

The idea of the MMCC algorithm is that rather than finding any negative cycle, we find a minimum mean cycle. This can be done in  $\mathcal{O}(mn)$  time, that is (asymptotically) no different than finding an arbitrary negative cycle. This the number of iterations then needed for minimum mean cycles is then  $\mathcal{O}(nm^2 \log n)$ . We give full proofs of both of these in the two preceding sections. It has actually been shown recently that we can improve the number of iteration of the algorithm to  $\mathcal{O}(m \log n)$  in <sup>[1]</sup>. We do not give a detailed version of the proof, but the idea is to reuse as much information as possible each time you find a minimum mean cycle.

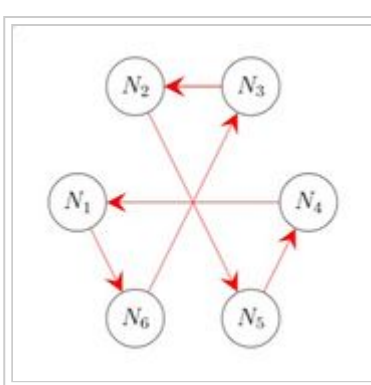
## Finding Minimum-Mean Cycles

In this section we give a  $\mathcal{O}(mn)$  time algorithm which finds a minimum mean cycle. First, arbitrarily fix some node  $s$ . Begin by iteratively creating  $d^k(u)$ , which represents the distance to  $u$  using exactly  $k$  edges. Let  $d(u)$  denote that shortest path from  $s$  to  $u$  (not using any cycles). Now, any path of length  $k$  to  $u$  must be the shortest of path of length  $k - 1$  which goes to some vertex  $w$ , plus the edge between  $w$  and  $u$ . Therefore if we calculate  $d^k(u)$  in increasing order of  $k$  we can take the minimum such  $w$  which gives us the recurrence relation:  $d^k(u) = \min_{w \in N, wu \in A} \{d^{k-1}(w) + c_{w,u}\}$ . This then gives a  $\mathcal{O}(mn)$  dynamic program to find all of these  $d^k(u)$ 's for  $k \leq n$ , as for each  $k$  we consider each arc exactly once.

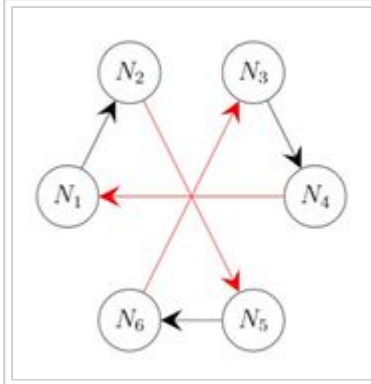
Let  $a = \min_{v \in N} \max_{k \in [n-1]} \frac{d^n(v) - d^k(v)}{n-k}$ . We claim that if  $a \geq 0$  then there is no negative cycle and the algorithm can terminate, and if  $a < 0$  then there is a negative cycle with minimum mean cost  $a$  and that we can easily ( $\mathcal{O}(mn)$  time) find a cycle with said minimum mean cost.

First consider when  $a \geq 0$ . Note that by the pigeonhole principle, the path associated with  $d^n(v)$  must contain at least one cycle for any  $v \in N$ . If this cycle has length  $k$  then we know that  $d^n(v) - d^{n-k}(v) \geq 0$  because a path to  $v$  could always use the  $n - k$  edges not used in the cycle (note that it could possibly be even smaller, but this does not change the fact the quantity is positive). Since we are maximizing over  $k \in [n - 1]$  this means that  $a \geq 0$ .

Now consider when  $a < 0$ . We first show that  $a$  gives the correct value. Note that we may assume that the minimum mean cycle length is 0 and show that  $a = 0$ . To see this take some negative mean cycle with average edge cost  $r$  and update the network by adding  $r$  to every edge. Then we increased every edge the same amount, and so the mean of any cycle increases by  $r$ . Therefore the new network only has non-zero mean cycles, and the cycle with minimum mean cost now has cost 0. Now, let  $C$  be our cycle of zero length. Assume that  $u \in C$  is some vertex at the end of a shortest path between  $s$  to  $C$ . We claim that if  $v \in C$  and  $x$  is the cost to reach  $v$  from  $u$  along that cycle, that  $d(v) = d(u) + x$ . To see this note that  $d(v) \leq d(u) + x$  as a path from  $s$  to  $v$  could possibly pass through  $u$  and then around the cycle. Furthermore  $d(u) \leq d(v) - x$  as a shortest path from  $s$  to  $u$  could possibly go to  $v$  and then through the cycle to  $u$ , which has  $-x$  cost, as the mean cycle length, and therefore the cycle length, is zero. Now let  $v$  be the vertex for which if we went from  $s$  to  $u$  and then around the cycle enough times to have  $n$  edge in our walk, we would end at  $v$ . Note that we must go around the cycle at least once, as the shortest path from  $s$  to  $u$  has less than  $n - |C|$  edges as it doesn't go around any cycles, and if it hit  $C$  in a sooner place we would just take that vertex. Therefore, combining this with with the result above we know that  $d(v) = d^n(v) = d^{n-|C|}(v)$  meaning that  $a \leq 0$ . However, we also know that  $a \geq 0$  and so we get the desired result.



**Figure 5:** Cycle



**Figure 6:** New Feasible Solution



To find the actual minimum mean cycle, we can slightly modify the algorithm to include the predecessor for  $d^k(v)$  that is the vertex  $u$  such that  $d^k(v) = d^{k-1}(v) + c_{u,v}$ . Then we can see from our description above, that for the correct vertex  $v$ ,  $d^n(v)$  contains a minimum mean cycle, so we can simply walk through the predecessors to get the cycle.

## Number of Iterations

One of the main benefits to the Minimum Mean Cycle Cancelling Algorithm is that it gives a strongly polynomial time algorithm for the min cost flow problem, meaning that it can be run with irrational data, and the edge capacities and costs could grow very quickly with respect to the number of edges and vertices. We follow the order and structure of the proof found in the book, <sup>[2]</sup>, although ideas were also used from <sup>[1]</sup> as they also had a nice explanation of this algorithms running time. To complete the proof, we have three different sections. First, we prove structural results about reduced costs, and how that compares to the minimum mean cycle. Then we prove that after some number of steps ( $n \leq$ ) we lower the upper bound on the cost of the minimum mean cycle by some constant factor ( $1/n$ ), which then gives a weakly-polynomial time algorithm incidentally as a by product. Then we utilize the idea of "fixing an edge" after some number of steps, that is we show that after some number of steps ( $\mathcal{O}(nm \log n)$ ) that at least one edge will no longer be in any minimum mean cycles, which gives us our strongly polynomial time algorithm.

## Structural Properties

For some set of node potentials  $\pi$  and some flow  $x$  let  $-\epsilon^\pi(x)$  be the minimum reduced cost in the network, that is,  $\epsilon^\pi(x) = -\min_{(i,j) \in A} c_{i,j}^\pi$ . Furthermore, let  $\epsilon(x) = -\min_\pi (\epsilon^{\pi(x)})$ . Big picture, we wish to show how quickly our minimum mean cycle cost goes to zero, which will initially give us our weakly polynomial time. However, analyzing all cycles (and their mean cost) in the graph is difficult, so instead we will relate  $\epsilon(x)$  to the minimum mean cost cycle for flow  $x$ , which we will denote as  $\mu(x)$ . This then allows us to analyze the behavior of  $\mu$  by looking at the reduced costs, rather than any cycles. One additional thing to note is that for any fixed node potential  $\pi$  the network with reduced costs has the same minimum mean cycle with the same mean cost. This is because  $\sum_{i,j \in C} c_{i,j}^\pi = \sum_{i,j \in C} c_{i,j} + \pi_i - \pi_j = \sum_{i,j \in C} c_{i,j}$  as the second to last sum is telescoping.

Lemma: For any nonoptimal flow  $x$ ,  $\epsilon(x) = -\mu(x)$ .

Proof: We first show that  $\epsilon(x) \geq -\mu(x)$ . Take some minimum mean cycle, then some edge must have at least the value  $\mu(x)$  as otherwise the average could not possibly be  $\mu(x)$ . Since we know that the minimum mean cycle cost does not change if we use reduced costs, this means that for any set of node potentials  $\pi$  there must be some edge with value at least  $\mu(x)$ , and so therefore  $\epsilon(x) \geq -\mu(x)$ .

Now we show that  $\epsilon(x) \leq -\mu(x)$ . This will use a similar technique to showing the running time of finding the minimum mean cycle, where we will convert the network to one with no negative cycles. In particular, what that will allow us to do is use shortest path labels as our node potentials. Specifically for any edge  $(i, j)$  let  $c'_{i,j} = c_{i,j} - \mu(x)$ . Then select some arbitrary node  $s$  and set  $\pi(i) = d(s, i)$ . It then suffices to show that  $c'_{i,j} \geq 0$  for any edge  $i, j$ . The shortest path optimality condition gives us this instantly however as  $c'_{i,j} = c'_{i,j} - d(i) + d(j) \geq 0$ .

Lemma: For any nonoptimal flow  $x$  there exists some set of node potentials  $\pi$  such that the reduced costs along some minimum mean cycle are the same, that is  $c_{i,j}^\pi = \mu(x)$  for all  $(i, j) \in C$ .

Proof: The almost follows directly from the previous lemma, take the same graph transformations and same potentials. Then our minimum mean cycle has zero mean cost, and every edge has a non-negative cost, which means that every edge has zero cost. Then converting back to our original network in the same way proves the lemma.

## Weakly Polynomial Time Algorithm

From now on, using the results of the structural properties, we will deal with the convergence of  $\epsilon(x)$  to 0 rather than the size of the minimum mean cycle. We show that the value of  $\epsilon(x)$  must be strictly decreasing in blocks of steps (where the block sizes are constant sizes). This then gives our weakly polynomial time algorithm, and will be used in the strongly polynomial time algorithm to bound time until edges are fixed (see the next section).

Lemma:  $\epsilon(x)$  is (not strictly) decreasing.

Proof: Since at each step we are augmenting only around the minimum mean cycle, these are the only edges that change from one step to the next. From the previous section, take  $\pi$  to be such that all edges around our minimum mean cycle take value  $-\epsilon(x)$ . Then after we augment flow around the cycle to give  $x'$  we either do not introduce any additional edges, or only introduce edges with cost  $\epsilon(x)$ , so  $\epsilon^\pi(x') \leq \epsilon(x)$ . Since  $\epsilon(x')$  is taken as the minimum over all node potentials, we have the desired result.

Lemma: After  $m$  steps  $\epsilon(x)$  decreases by a factor of at least  $1 - 1/n$ .

Proof: Let  $\pi$  be node potentials such that  $c_{i,j}^\pi \geq -\epsilon(x)$  for any edge  $(i, j)$ . We will consider all costs as reduced costs. Note that each time we augment flow through some minimum mean cycle we flip at least one edge, as we are sending the maximum amount of flow through. So either we augment flow around  $m$  cycles all with negative reduced cost, and we are done as all reduced costs are non-negative, which certainly gives an improvement of at least  $1 - 1/n$ , or every  $m$  steps we augment flow around a cycle with some positive reduced cost. We claim doing this decreases  $\epsilon(x)$  by a factor of at least  $1 - 1/n$ .

We know from the previous lemma that the minimum mean cycle cost is increasing. Let  $x$  be the flow one step before augmenting around the cycle with the positive reduced cost edge, and let  $x'$  be the flow on the step after that. The minimum mean cycle  $C$  has mean weight at most  $(|C| - 1)(-\epsilon(x))/|C|$ . This is because  $\epsilon(x)$  is decreasing so each edge except one around  $C$  has cost at least cost  $-\epsilon(x)$ , and one has cost 0. So  $\epsilon(x') \leq (1 - 1/|C|)(\epsilon(x)) \leq (1 - 1/n)(\epsilon(x))$ .

Theorem: The minimum mean cycle cancelling algorithm runs in  $\mathcal{O}(n^2 m^2 \log(nC))$  time with integer costs and .

Proof: The bottleneck for each step is finding the minimum mean cycle, which as we have previously proved runs in  $\mathcal{O}(nm)$  time. Now if at some step  $\epsilon(x) < 1/n$  then we have an optimal flow as our minimum mean cycle would have cost more than  $-1$  but as we know we preserve integrality of (not reduced) costs, this means that the cycle has flow at least 0 and we are done. Therefore combining this with the previous lemma, it suffices to show that  $(1 - 1/n)^k C < 1/n$  for  $k = \mathcal{O}(n \log(nC))$  as we may start with cost (and therefore  $\epsilon(x)$ ) of at most  $-C$ . Solving for  $k$  gives  $k < \frac{\log(nC)}{-\log(1-1/n)}$ , so we just have to show that  $\frac{1}{-\log(1-1/n)} = \mathcal{O}(n)$ . We know that

$$\left(\log\left(\frac{x}{x-1}\right)\right)^{-1} = \left(\int_1^{x/(x-1)} dx/x\right)^{-1} \leq \left(\frac{x}{x-1} - 1\right)^{-1} = \mathcal{O}(x) \text{ as desired.}$$

# Strongly Polynomial Time Algorithm

To create a strongly polynomial time algorithm, we have to consider a slightly different strategy from the previous two subsections. We can't just consider how quickly individual arc edges go to positive reduced cost as this depends on the value of  $C$ . Instead we show that after so many steps edges will become 'fixed', that is they will never appear in another minimum mean cycle. Then we just have to go through  $m$  groups of these steps as then there aren't any edges that could possibly be used by the algorithm.

Lemma: For some given set of flow  $x$  let  $\pi$  be such that  $\epsilon^\pi(x) = \epsilon(x)$ . Then if  $|c_{i,j}^\pi| \geq 2n\epsilon(x)$  then edge  $i, j$  is fixed in all future iterations of the algorithm.

Proof: We first consider when  $c_{i,j}^\pi \geq 2n\epsilon(x)$ . Assume towards a contradiction at some future step we send flow along that arc. Then as  $\epsilon(x)$  is decreasing, this cycle has cost at least  $(|C| - 1)(-\epsilon(x)) + 2n\epsilon(x) > 0$ , a contradiction as the algorithm would have already terminated. So now assume that  $c_{i,j}^\pi \leq -2n\epsilon(x)$ , this however cannot happen by our definition of  $\epsilon(x)$ .

Theorem: The minimum mean cycle cancelling algorithm runs in  $\mathcal{O}(n^2m^3 \log n)$  time.

Proof: It suffices to show that after  $\mathcal{O}(nm^2 \log(n))$  steps at least one additional edge is fixed, as we know that we can find minimum mean cycles in  $\mathcal{O}(nm)$  time, and we can fix at most  $m$  edges. Let  $x$  be the flow before our set of steps, and  $x'$  be the flow afterwards, and let  $\pi, \pi'$  be the sets of node potentials which give the best  $\epsilon$  for  $x$  and  $x'$  respectively. We can modify the algebra in the proof of the weakly polynomial theorem to show that  $(1 - 1/n)^{cnm \log(n)} \leq \frac{1}{2n}$  for some fixed constant  $c$ . This means that  $\epsilon(x') \leq \frac{\epsilon(x)}{2n}$ . However, with flow  $x$  there is at least one edge  $i, j$  in the minimum mean cycle with reduced cost  $c_{i,j}^{\pi'} = -\epsilon(x)$ , which we send the maximum amount of flow through (we can deal with  $\pi'$  instead  $\pi$  as the cost of the minimum mean cycle does not change with the node potentials). Then we get  $|c_{i,j}^{\pi'}| \geq 2n\epsilon(x')$  and so by our previous lemma, we know that it is fixed.

## References And Powerpoint

You can access the powerpoint of our presentation here: <https://github.com/toadhkjl/Min-Mean>

Jean Bertrand Gauthier, Jacques Desrosiers, Marco Lübbecke. About the minimum mean cycle-canceling algorithm.

- <sup>1.0 1.1</sup> J. Bertrand Gauthier, J. Desrosiers, M. E. Lübbecke; About the Minimum Mean Cycle-Canceling Algorithm, Discrete Applied Mathematics, (2013).
- <sup>↑</sup> Ravinda K. Ahuja, Thomas L. Magnanti, James B. Orlin. Network Flows. 1993.

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Min-mean\_Cycle\_Cancelling\_Algorithm\_and\_Its\_Applications&oldid=2806"

- This page was last modified on 5 May 2020, at 02:23.
- This page has been accessed 4,287 times.

# Minimum Cost-to-Time Ratio Cycle Problem

From CU Denver Optimization Student Wiki

## Contents

- 1 Abstract
- 2 Introduction
- 3 Finding the Optimal Value
- 4 Algorithms
- 5 Special Case: The Minimum Mean Cycle Problem
- 6 References
- 7 Contributor
- 8 GitHub

## Abstract

The Minimum Cost-to-Time Ratio Cycle Problem is a min-cost flow problem concerned with finding the smallest ratio between cost and traversal time of the cycle. We first discuss the problem setup, then bounds and algorithms for finding the optimal value. We end by discussing the Minimum Mean Cycle, a special case of this problem.

## Introduction

The Minimum Cost-to-Time Ratio Cycle Problem is a type of min-cost flow problem derived from an application referred to as the 'tramp steamer problem'. Information about this problem can be found in <sup>[1]</sup>. In the tramp steamer problem, a boat is moving between ports, generating profit as it moves. Each trip takes  $\tau_{ij}$  units of time and generates  $p_{ij}$  units of profit. The goal is to find the maximum profit that can be generated by traveling to ports in a directed cycle. We do this by finding the maximum ratio of profit divided by travel time along the cycle. In other words, we want to maximize the ratio

$$\mu(W) = \frac{\sum_{(i,j) \in W} c_{ij}}{\sum_{(i,j) \in W} \tau_{ij}},$$

where  $\mu$  is the value of the ratio.

This problem is framed as a maximization problem, but by defining  $c_{ij} = -p_{ij}$ , we can treat it as a minimization problem instead. We also assume all data is integral,  $\tau_{ij} \geq 0$  for all arcs (i,j), and that  $\sum_{(i,j) \in W} \tau_{ij} > 0$  for all directed cycles W in G.

# Finding the Optimal Value

There are several ways to find the optimal value  $\mu^*$ . The first two methods revolve around detecting negative cycles. We start with an initial guess for our optimal solution, call it  $\mu$ . We then define the length of each arc as  $l_{ij} = c_{ij} - \mu\tau_{ij}$ . Then we look for a negative cycle  $W$  in our graph  $G$ . This subtraction produces three cases.

Our first case is that  $G$  contains a negative cycle  $W$ . In this case,  $\sum_{(i,j) \in W} (c_{ij} - \mu\tau_{ij}) = \sum_{(i,j) \in W} l_{ij} < 0$ . From this, we get that  $\mu > \frac{\sum_{(i,j) \in W} c_{ij}}{\sum_{(i,j) \in W} \tau_{ij}}$ . Since this is the case, our chosen value of  $\mu$  is too large, and thus  $\mu$  is a strict upper bound on  $\mu^*$ .

Our second case is that  $G$  contains no negative cycle and contains a cycle of length zero  $W^*$ . In this case, since there is no negative cycle, we get that  $\sum_{(i,j) \in W} (c_{ij} - \mu\tau_{ij}) = \sum_{(i,j) \in W} l_{ij} \geq 0$  for every directed cycle  $W$ . This means that the value of  $\mu$  is less than or equal to the value of the ratio of every directed cycle in  $W$ , or  $\mu \leq \frac{\sum_{(i,j) \in W} c_{ij}}{\sum_{(i,j) \in W} \tau_{ij}}$ . Since a zero cycle exists, we know  $\mu = \frac{\sum_{(i,j) \in W} c_{ij}}{\sum_{(i,j) \in W} \tau_{ij}}$ . These two conditions together imply that  $\mu = \mu^*$ , and that  $W^*$  is a minimum cost-to-time ratio cycle.

The last case is that  $G$  contains only positive-length directed cycles. In this case,  $\sum_{(i,j) \in W} (c_{ij} - \mu\tau_{ij}) = \sum_{(i,j) \in W} l_{ij} > 0$ . From this, we get that  $\mu < \frac{\sum_{(i,j) \in W} c_{ij}}{\sum_{(i,j) \in W} \tau_{ij}}$ . Since this is the case, our chosen value of  $\mu$  is too small, and thus  $\mu$  is a strict lower bound on  $\mu^*$ .

Using these guidelines, there are two algorithms that can be used to find the optimal value  $\mu^*$ : the sequential search algorithm, and the binary search algorithm.

## Algorithms

The sequential search algorithm starts with  $\mu^0$ , a known upper bound on  $\mu^*$ . We then compute  $(c_{ij} - \mu^0\tau_{ij})$  and look for cycles. Since  $\mu^0$  is a strict upper bound on  $\mu^*$ , this will either result in a zero-length cycle or a negative cycle. If we have a zero-length cycle, we are done. If we have a negative cycle, we let  $\mu^1 = \frac{\sum_{(i,j) \in W} c_{ij}}{\sum_{(i,j) \in W} \tau_{ij}}$  and repeat the process. This algorithm can be shown to run in pseudo-polynomial time.

The binary search algorithm starts with an interval  $[\underline{\mu}, \bar{\mu}]$  that contains the optimal solution  $\mu^*$ . An easy interval to start with is  $[-C, C]$ , where  $C$  is the largest arc cost in  $G$ .

At each iteration, we let  $\mu^0 = (\frac{\bar{\mu} + \underline{\mu}}{2})$  and look for a negative cycle using the arc lengths  $(c_{ij} - \mu^0\tau_{ij})$ . If a negative cycle exists, then we know  $\mu^0$  is a strict upper bound on  $\mu^*$ , so we let  $\mu^0 = \bar{\mu}$  and repeat this process. During each iteration, we halve the length of the interval. It is clear that the algorithm will eventually converge to a small enough interval such that there is a unique solution. This algorithm runs in  $O(\log(\tau_0 C))$  iterations, where  $\tau_0$  is the largest traversal time of any arc in  $G$ .

# Special Case: The Minimum Mean Cycle Problem

While the previous two algorithms will work for any minimum cost-to-time ratio problem, we can use an  $O(nm)$  implementation for the special case where  $\tau_{ij} = 1$  for all arcs in  $G$ . This setup is called the minimum mean cycle problem.

For the minimum mean cycle problem, we are now trying to find the minimum ratio  $\frac{\sum_{(i,j) \in W} c_{ij}}{|W|}$  among directed cycles in  $G$ . In addition to the previous assumption (integrality of data, etc), we also assume that the graph is strongly connected. If it is not strongly connected, we can add arcs where they are missing with sufficiently large cost. These arcs will not be contained in the minimum mean cycle unless the graph is acyclic.

To find the minimum mean cycle, we start by finding  $d^k(j)$  for  $1 \leq j \leq n$ , where  $d^k(j)$  denotes the shortest path with respect to arc costs  $c_{ij}$  from a designated starting node to another node  $j$  that contains exactly  $k$  arcs. We set  $d^0(j) = \infty$  and then compute  $d^k(j) = \min\{d^{k-1}(i) + c_{ij}\}$  for all  $j$ . This takes  $O(m)$  time, and since we do this for all nodes, the entire computation takes  $O(nm)$  time.

We will use the following theorem to find the optimal value:

$$\text{Theorem: } \mu^* = \min_{j \in N} \max_{0 \leq k \leq n-1} \left[ \frac{d^n(j) - d^k(j)}{n - k} \right]$$

This can be proven in two cases. The first case is that  $\mu^* = 0$ . In this case, the network contains a zero cycle and no negative cycle. For each node  $j$ , we compute the shortest path distance from a defined node  $s$  to node  $j$ . We denote this  $d(j)$ . We then replace each arc cost  $c_{ij}$  with the reduced arc costs  $c_{ij}^d = c_{ij} + d(i) - d(j)$ . After this transformation, all arc costs are nonnegative, and the arcs on the cycle  $W$  have cost zero, each arc on the shortest path from  $s$  to  $t$  has cost zero, and lastly, the shortest path distances  $d^k(j)$  differ by a constant amount from their values pre-transformation.

Now, we let  $\bar{d}^k(j)$  denote the length of the shortest walk from node  $s$  to  $j$  using the reduced cost arcs  $c_{ij}^d$ . Since all arc costs are nonnegative and differ by a constant amount, we get that  $\max_{1 \leq k \leq n-1} [\bar{d}^n(j) - \bar{d}^k(j)] \geq 0$ . We know this is true because the shortest path length will be 0, and  $\bar{d}^n(j)$  will be at least as large.

From here, we find the shortest path from node  $s$  to a node  $j$  that is in our zero cost cycle  $W$ . That path will have length of at most  $n-1$ , so we continue to walk from node  $j$  along  $W$ . This creates a walk from node  $s$  to another node  $p$  that is on  $W$ . All arcs on  $W$  and all arcs in the shortest path from  $s$  to  $j$  have zero cost, so we now have a walk from  $s$  to  $p$  with zero cost. We also know that the walk must contain one or more directed cycles because it contains  $n$  arcs. So, when we remove the directed cycle from  $W$ , we end up with a walk of length  $k \leq n - 1$  from  $s$  to  $p$ . We have shown that  $d^n(p) - d^k(p) = 0$ , satisfying the first case of our theorem.

If  $\mu^* \neq 0$ , then we simply adjust the cost of each arc by a number chosen to change  $\mu^*$  to zero. We then apply the results of case 1, which completes case 2.

The last step is to take the minimum of these differences. Since all path lengths will be greater than or equal to 0, we know that the minimum difference will be 0. Thus, the theorem is proven.

If we suppose that  $d^k(j)$  for all  $j \in N$  and  $1 \leq k \leq n$  are available, then by simply calculating  $\max_{1 \leq k \leq n-1} \left[ d^n(j) - d^k(j) \right]$  and then taking the minimum, we can get the minimum mean cycle. This takes  $O(nm)$  time to get the shortest path pairs, and then takes  $O(n^2)$  time to get the minimum of these maxima. The proof of this algorithm can be read about here <sup>[2]</sup>.

## References

1.

↑

Ravinda K. Ahuja, Thomas L. Magnanti, James B. Orlin. Network Flows. 1993.

2.

↑

Karp, R. M. (1978). A characterization of the minimum cycle mean in a digraph. In Discrete Mathematics (Vol. 23, Issue 3, pp. 309–311). Elsevier BV. [https://doi.org/10.1016/0012-365x\(78\)90011-0](https://doi.org/10.1016/0012-365x(78)90011-0)

## Contributor

Nicholas Rogers

## GitHub

<https://github.com/rogersnj13/Minimum-Cost-to-Time-Ratio-Cycle-Problem> This GitHub link takes you to the repository containing a powerpoint presentation about this problem and any code that I used for this project.  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Minimum\\_Cost-to-Time\\_Ratio\\_Cycle\\_Problem&oldid=4811](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Minimum_Cost-to-Time_Ratio_Cycle_Problem&oldid=4811)"

- This page was last modified on 2 May 2024, at 08:59.
- This page has been accessed 32 times.

# Minimum Planar Crossing Number

From CU Denver Optimization Student Wiki

Finding the minimum crossing number of a graph is a problem that is easy to state but notoriously difficult to solve. It can be stated simply as given a Graph  $\mathbf{G}(\mathbf{V}, \mathbf{E})$ . What is the minimum number of edge crossing over all possible drawings in the Euclidean plane. In graph theory the minimum crossing number is often stated simply as the crossing number of a graph and is denoted as  $\text{cr}(\mathbf{G})$ . Determining the crossing number of a graph is still a very active problem in research, due to it's importance in many fields. Studies have shown people find graphs with few edge crossing to be easier to understand. The python package graphviz (<https://pypi.python.org/pypi/graphviz>) uses different approximation methods to the problem to draw the graphs you see below. The crossing number of a graph also plays a role in VLSI design (computer chips). To avoid trying to build optimal integrated circuits I will present a Integer Programming formulation to a simplified version of the problem. The minimum single crossing number denoted as  $\text{crs}(\mathbf{G})$ . The single crossing number of a graph is a restriction of the general problem in that it requires that any edge may only cross a single other edge. This will allow use to build an integer programming formulation for the problem.

## Contents

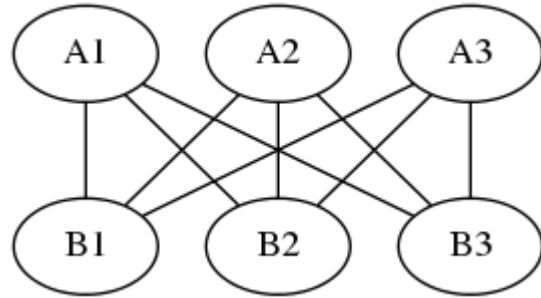
- 1 Planarity of a Graph
- 2 Building a Integer Programming Formulation
  - 2.1 Realizability Problem
  - 2.2 Kuratowski constraints
  - 2.3 IP for the  $\text{crs}(\mathbf{G})$
- 3 Finding the Kuratowski subdivisions
  - 3.1 Sources

## Planarity of a Graph

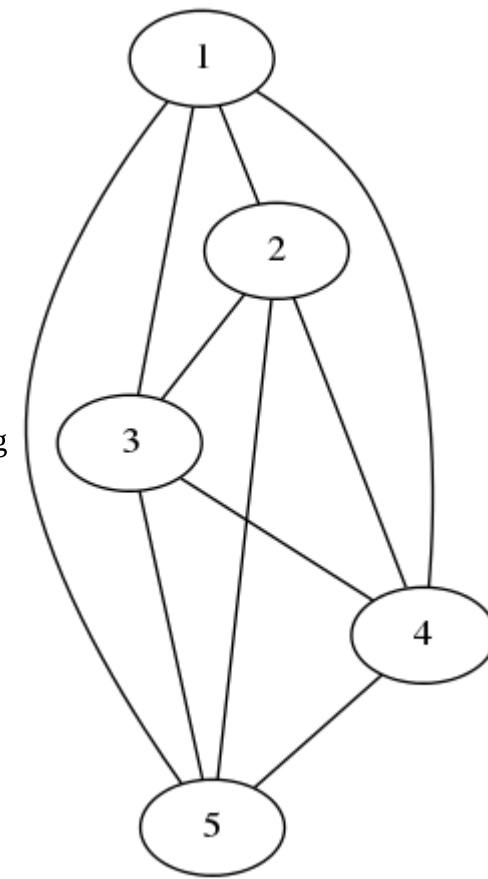
Graph planarity is a well studied and well understood problem, there are many known linear time algorithms for testing a graph for planarity ([https://en.wikipedia.org/wiki/Planarity\\_testing](https://en.wikipedia.org/wiki/Planarity_testing)). Many of which extract a forbidden sub graph when they return a non-planar determination. The forbidden sub graphs that can't exist in any planar graph were classified by Kazimierz Kuratowski which is stated in a theorem named after him. Kuratowski's Theorem states that a finite graph is planar if and only if it does not contain a subgraph that is a subdivision of  $\mathbf{K}_5$  (the complete graph on five vertices) or of  $\mathbf{K}_{3,3}$  (complete bipartite graph on six vertices, three of which connect to each of the other three, also known as the utility graph).



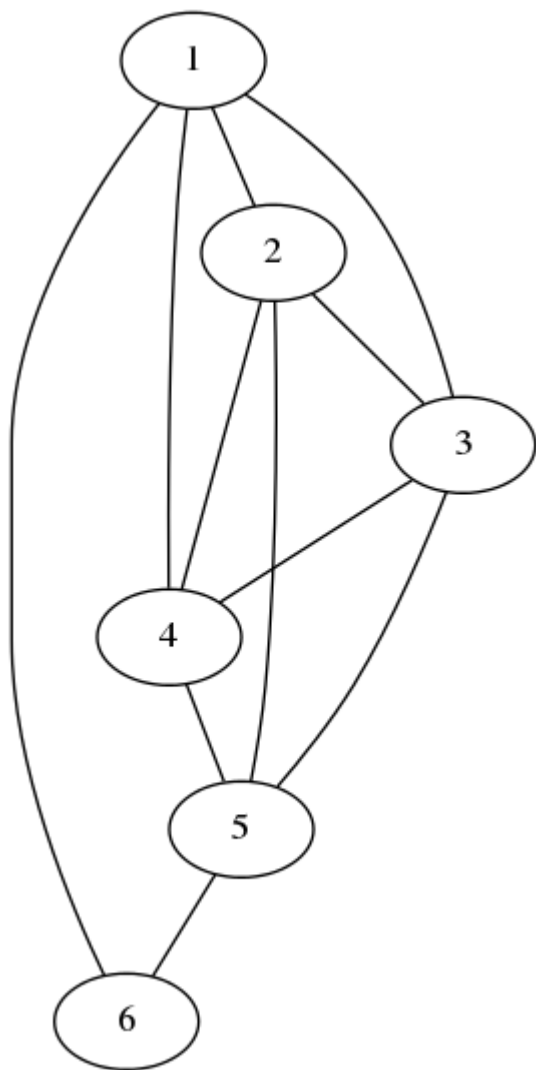
Utility Graph



$K_5$  drawn with a single crossing

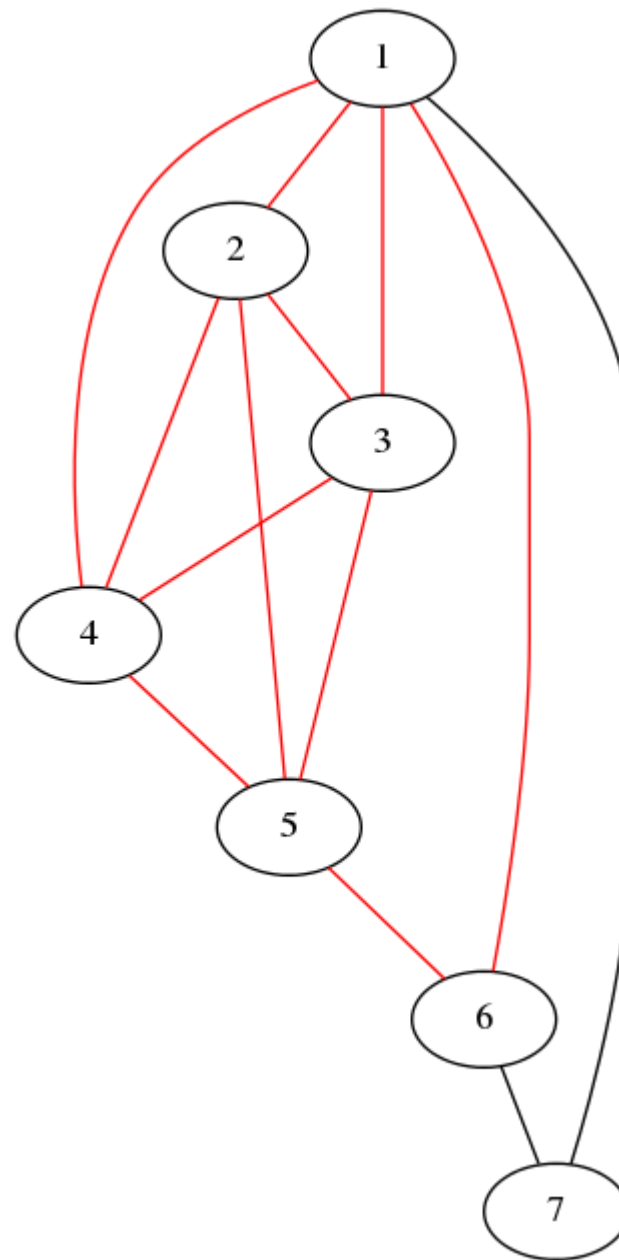


A subdivision of a graph is when you can add vertices along any edge between any two vertices(Example Below). From here on I will refer to a Kuratowski subgraph  $\mathbf{H}$  of a graph  $\mathbf{G}$  as collection of edges that form a subdivision of  $\mathbf{K}_5$  or  $\mathbf{K}_{3,3}$ . Example below.



Simple sub division of K5

Kuratowski subgraph.

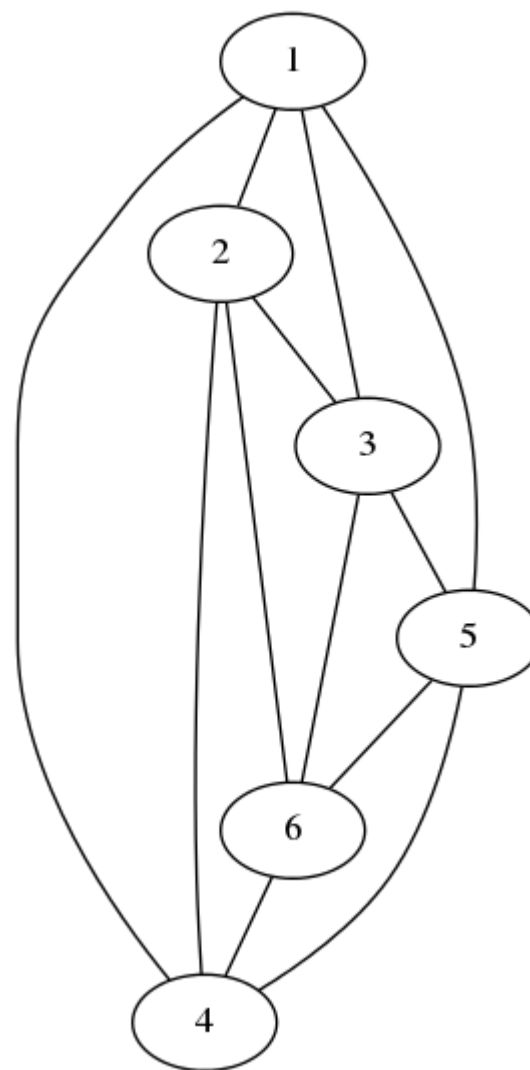


We will rely heavily on this theorem for are formulation.

# Building a Integer Programming Formulation

We will begin are formulation by introducing a set of variables. We will define a set of  $\{0,1\}$  variables which will represent the crossing of edge  $i$  and  $j$   $x_{ij} = 1$  if edge  $i$  crosses edge  $j$  else  $x_{ij} = 0$ . I will refer to this set of integer variables as  $\mathbf{E}^2$  from here on out and the set of edges as  $\mathbf{E}$ . To simplify the problem we will restrict the problem again to simple graphs (<http://mathworld.wolfram.com/SimpleGraph.html>). Which we will use to define are variables.  $x_{ii} = 0$  no loops. Also if  $x_{ij} = 1$  then  $x_{ji} = 1$ , undirected graph. This leads to a linear time why to check if a solution does in fact result in a planar graph. We can place a new vertex at the each edge crossing and we could then use any of the known planarity testing algorithms to see if the resulting graph is planar.

Example of placing a new vertex in  $K_5$  at the intersection draw in the graph above



The Integer Program can now be informally stated as:

$$\begin{array}{ll} \min & \sum_{x \in \mathbf{E}^2} x_{ij} \\ \text{s.t.} & \sum_{j \in \mathbf{E}} x_{ij} \leq 1 \forall i \in \mathbf{E} \end{array}$$

A set of constraints that forbids the resulting graph from having a Kuratowski subgraph "Kuratowski constraints"

The first set of constraints forces the condition for the minimum single crossing number. The second set must now be constructed to assure that the graph resulting from placing degree 4 vertices at the edge intersections will be planar. If we consider dropping the first set of constraints we would not end up with a Integer Program for the general problem. If we were to solve this problem without the first set of constraints we would be left with another problem that is also difficult. Known as the Realizability problem.

## Realizability Problem

By dropping the first set of constraints we would allow for edges to cross more than one other edge. The result may not even be planar and if it is it may be very hard to find. The problem comes from not having any information on the proper ordering of the crossings. We are not left with a simple way to add vertices to the graph to check for planarity. The Realizability Problem has been proven to be NP-complete (Kratohvil).

## Kuratowski constraints

So we seek a set of linear inequalities that will assure us that the graph resulting from inserting a vertex for each  $x_{ij} = 1$  will result in a planar graph. First definitions.

Let  $\mathbf{G}_{\mathbf{D}}$  represent a graph that is generated from adding vertices to  $\mathbf{G} \forall x_{ij} \in \mathbf{D}$ .

Let  $\mathbf{H}$  represent a Kuratowski subgraph in  $\mathbf{G}_{\mathbf{D}}$  (A subdivision of  $\mathbf{K}_5$  or  $\mathbf{K}_{3,3}$  in the graph generated by adding a vertex for each element of  $\mathbf{D}$ ) Let  $\mathbf{H}'$  denote the set of edges that exist in the original set of edges  $\mathbf{E}$ . And  $\mathbf{H}'^2$  represent the set of all possible pairings of these edges then

$$\sum_{\mathbf{H}'^2 \setminus \mathbf{D}} x_{ij} - \sum_{\mathbf{H}'^2 \cap \mathbf{D}} x_{ij} \geq 1 - |\mathbf{H}'^2 \cap \mathbf{D}| \text{ for every } \mathbf{D} \subseteq \mathbf{E}^2 \text{ and every Kuratowski subgraph } \mathbf{H} \in \mathbf{G}_{\mathbf{D}}$$

So we end up with not only an exponential number of subsets to consider. We have an undetermined number of Kuratowski subgraphs in each subset. If we wanted to try and write down every constraint for a given instance of the problem. To understand these constraints I will put together some examples of why it must be true for any feasible solution. (I am still working on making it presentable). Even though we have an exponential number of constraints it turns out to be a good formulation for a branch and bound solution strategy. See Dr. Petra Mutzel, Dr. Gunnar W. Klau, and Rene Weiskircher. Optimal Crossing Minimization using integer linear programming (<http://www.complang.tuwien.ac.at/cd/ebner/ebner05da.pdf>).

# IP for the crs(G)

Then are final formulation becomes

$$\begin{aligned} \min \quad & \sum_{x \in \mathbf{E}^2} x_{ij} \\ \text{s.t.} \quad & \sum_{j \in \mathbf{E}} x_{ij} \leq 1 \forall i \in \mathbf{E} \\ & \sum_{\mathbf{H}'^2 \setminus \mathbf{D}} x_{ij} - \sum_{\mathbf{H}'^2 \cap \mathbf{D}} x_{ij} \geq 1 - |\mathbf{H}'^2 \cap \mathbf{D}| \text{ for every } \mathbf{D} \subseteq \mathbf{E}^2 \text{ and every Kuratowski subgraph } \mathbf{H} \in \mathbf{G}_{\mathbf{D}} \end{aligned}$$

We still have a another problem that needs to be solved for are formulation to be on any practical use. How to find the Kuratowski subdivisions.

## Finding the Kuratowski subdivisions

See here (<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.206.557&rep=rep1&type=pdf>). If you are interested.

## Sources

J. Kratochvil. String graphs 2. Recognizing string graphs is NP-hard. J. comb Theory Ser. B, 52(1):67-78, 1991 Dr. Petra Mutzel, Dr, Gunnar W. Klau, and Rene Weiskircher. Optimal Crossing Minimization using integer linear programming. February 2005 (need to finish this)  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Minimum\\_Planar\\_Crossing\\_Number&oldid=534](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Minimum_Planar_Crossing_Number&oldid=534)"

- This page was last modified on 3 May 2017, at 11:57.
- This page has been accessed 13,189 times.

# Minimum Spanning Tree

From CU Denver Optimization Student Wiki

## Project Description

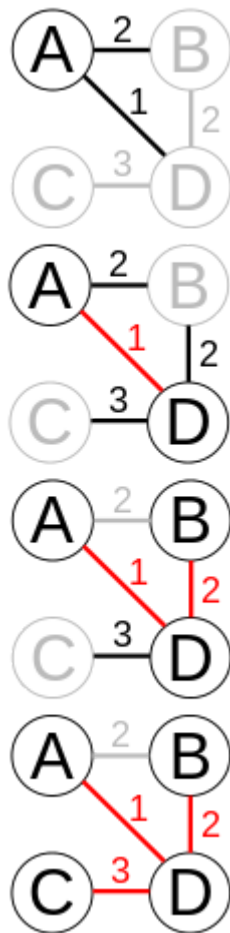
**A Minimum Spanning Tree** [Robert Sedgewick and Kevin Wayne / <https://algs4.cs.princeton.edu/home/> ] is a subgraph of an undirected graph such that the subgraph spans (includes) all nodes, is connected, is acyclic, and has minimum total edge weight. Three main algorithms related to Min Spanning Tree are Kruskal's algorithm, Prim's algorithm and Boruvka's algorithm.

Application:

Reducing data storage in sequencing amino acids in a protein.  
Network design (communication, electrical, hydraulic, computer, road).

**Prim's algorithm:** [Algorithm Design: Parallel and Sequential" book by Umut A. Acar and Guy E, [1] (<http://www.parallel-algorithms-book.com/>)] Prim's algorithm is an algorithm for determining the minimal spanning tree in a connected graph. The algorithm is as the following

- Choose any starting vertex. Look at all edges connecting to the vertex and choose the one with the lowest weight and add this to the tree.
- Look at all edges connected to the tree that do not have both vertices in the tree.
- Choose the one with the lowest weight and add it to the tree.
- Repeat step 2 until all vertices are in the tree.



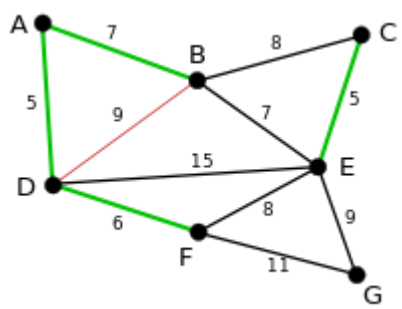
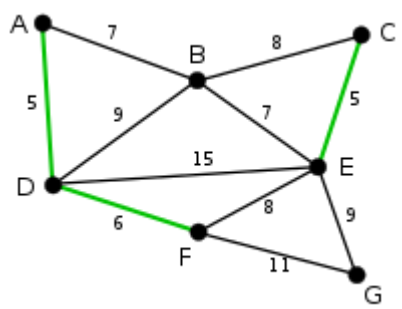
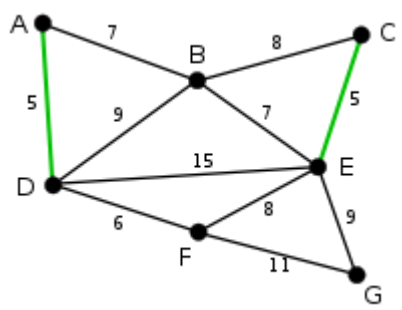
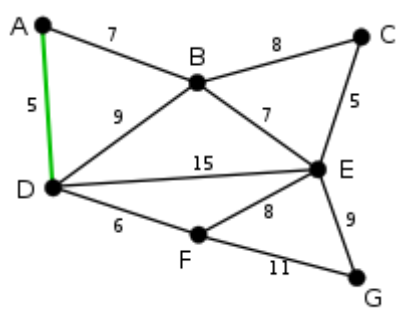
**Example of Prim's Algorithm:**

**Kruskal's MST Algorithm:** Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest (a minimum spanning tree for each connected component).

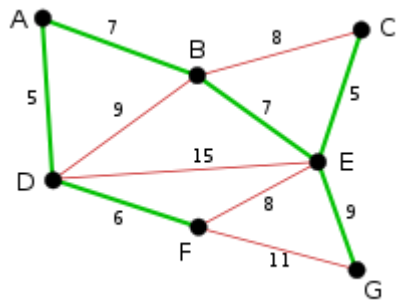
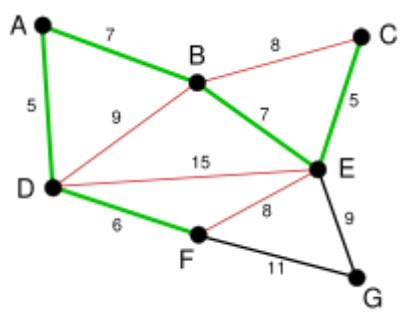
Description

- create a forest T (a set of trees), where each vertex in the graph is a separate tree
- create a set S containing all the edges in the graph
- while S is nonempty and T is not yet spanning
- remove an edge with minimum weight from S
- if that edge connects two different trees, then add it to the forest, combining two trees into a single tree, otherwise discard that edge.

**Example of Kruskal's Algorithm in 5 steps:**







**The Boruvka’s Algorithm:** [2] ([https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s\\_algorithm](https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm)) This algorithm was proposed by Otakar Borůvka in 1926. This is an algorithm to find the minimum spanning tree for a graph with distinct edge weights (none of the edges have the same value). The goal of the algorithm is to connect “components” using the shortest edge between components. It begins with all of the vertices considered as separate components.

Pseudocode:

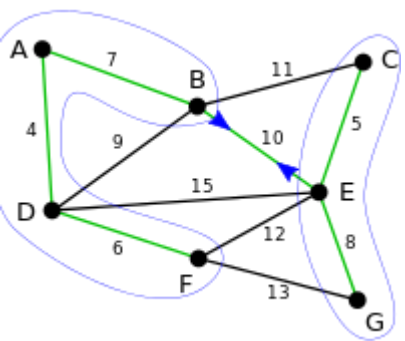
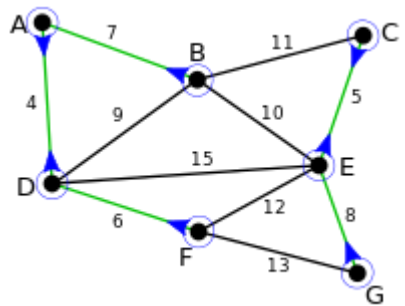
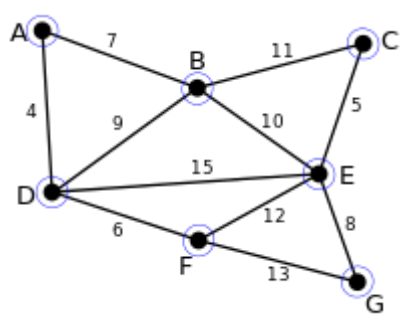
```

Input: A graph G whose edges have distinct weights
Initialize a forest F to be a set of one-vertex trees, one for each vertex of the graph.
While F has more than one component:
Find the connected components of F and label each vertex of G by its component
Initialize the cheapest edge for each component to "None"

For each edge uv of G:
If u and v have different component labels:
If uv is cheaper than the cheapest edge for the component of u:
Set uv as the cheapest edge for the component of u
If uv is cheaper than the cheapest edge for the component of v:
Set uv as the cheapest edge for the component of v
For each component whose cheapest edge is not "None":
Add its cheapest edge to F
Output: F is the minimum spanning forest of G.

```

**Example of Burukav Algorithm in three steps**



Resources:

1. Robert Sedgewick and Kevin Wayne [ [/https://algs4.cs.princeton.edu/home/](https://algs4.cs.princeton.edu/home/)]
2. Algorithm Design: Parallel and Sequential" book by Umut A. Acar and Guy E [3] (<http://www.parallel-algorithms-book.com/>)
3. [https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s\\_algorithm](https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm)  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Minimum\\_Spanning\\_Tree&oldid=1490](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Minimum_Spanning_Tree&oldid=1490)"

- This page was last modified on 1 May 2018, at 23:32.
- This page has been accessed 6,661 times.

# Monitoring Population to Track Low Student Retention

From CU Denver Optimization Student Wiki

## Abstract

We are investigating the enrollment trends in Denver neighborhoods to see if there are any schools losing students. The focus is on finding neighborhoods in Denver whose schools are either gaining school population slower than other regions in Denver during times when Denver's population is growing, or the population of enrolled students is decreasing without the population of Denver decreasing in the k-12 grades.

This project aims to find these trends by looking at the school data the city of Denver collected in the neighborhood surveys. This data collects data over 5 years and releases the aggregate broken down by neighborhood and serves as a good high-level view of the neighborhoods of Denver. We will be grouping the neighborhoods to include at least one school of every grade level and track the survey outputs to look at the stability of the population of each grade level over time. For example, if we start with 350 students in grades 1-4, then 4 years later if there is no population movement then there should be 350 students enrolled in grades 5-8 in a neighborhood.

There are many ways policy can affect schools that can affect school enrollment so the goal of this project is to find regions that are exhibiting these troubling trends so that policy makers can target them specifically to stop the loss of students.

## Helpful Project Links

GitHub (<https://github.com/szirkelbach/ORTopicsFinalProject>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Monitoring\\_Population\\_to\\_Track\\_Low\\_Student\\_Retention&oldid=3606](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Monitoring_Population_to_Track_Low_Student_Retention&oldid=3606)"

- 
- This page was last modified on 12 December 2021, at 01:39.
  - This page has been accessed 391 times.

# Moses Khan

From CU Denver Optimization Student Wiki

Moses Khan received his Master's Degree in Applied Mathematics at the University of Colorado Denver in 2018. Moses has been in the Health, Wellness and Fitness Industry for the past 27 years. He also received a Master's Degree in Exercise Physiology (With an emphasis in Sport Science and Nutrition) at the University of Texas, Austin. He also has studied Massage, Sports and Rehabilitative Therapy and completed his Bachelor's degree in Mathematics at the University of St. Thomas in Houston, Texas. Moses also has many certifications/licenses in Personal Training, Massage Therapy and as a Strength and Conditioning Expert.

Given Moses last Master's Degree in Applied Mathematics, he specialized in Operations Research with an emphasis in Optimization. Moses is hoping to use the mathematics that he is learning in order to optimize human performance, recovery and weight loss/gain/health maintenance to it's maximal capacity. Moses would also like to apply mathematical modelling towards learning/discovering more about neuroplasticity. Moses is also hoping to use Mathematical Modeling/Programming to help companies function at their maximal optimal potential.

Moses comes from the island of Trinidad and Tobago, and he hopes to use the mathematics that he is learning combined with everything else he has learned in his life in order to try and make a very significant and positive difference in the world.

Here are Moses's contributions to the Optimization Wiki:

Linear Programming for Diet and Nutrition

Derivation of Blood Flow as a Network Flow

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Moses\\_Khan&oldid=2508](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Moses_Khan&oldid=2508)"

Category: Contributors

- 
- This page was last modified on 10 February 2020, at 23:10.
  - This page has been accessed 5,709 times.

# Multi-commodity Flow

From CU Denver Optimization Student Wiki

The multi-commodity flow problem is a generalization of the maximum flow problem, where we need to find a maximum  $(s_i, t_i)$ -flow through the network for all commodities  $i = 1, \dots, k$ . while keeping the sum of flow over all commodities on each arc below its capacity. The formulation for this problem is a linear program that can be solved in strongly polynomial time.

$$\begin{aligned} \max \quad & \sum_{i=1}^k \left( \sum_{a \in \delta^+(s_i)} x_a^i - \sum_{a \in \delta^-(s_i)} x_a^i \right) \\ s.t. \quad & \sum_{a \in \delta^+(v)} x_a^i - \sum_{a \in \delta^-(v)} x_a^i = 0 \quad \text{for all } v \in V \setminus \{s_i, t_i\}, i = 1, \dots, k \\ & \sum_{i=1}^k x_a^i \leq u_a \quad \text{for all } a \in A \\ & x_a^i \geq 0 \quad \text{for all } a \in A, i = 1, \dots, k \end{aligned}$$

However due to the fact that more than one commodity travel on the same arc the problem becomes  $NP$ -hard when you look for an integer solution. Unlike standard max flow problem there is also no guarantee that an integer solution exists just because you have a feasible flow. Even if you are not looking for an integer solution it turns out in practice that while polynomial time algorithms exist to solve the multi-commodity flow problem, there is a larger issue to address. Say we have a graph with 1000 nodes and 10,000 edges, and we have 1 million commodities to send through this network (sending flow from each node to every other node). Then the number of variables in this seemingly reasonable problem becomes about 10 trillion, and at a measly 8 bytes per variable which is less than the space usually allocated to each variable in this type of problem, you have already used 80 gigabytes of storage space. Ultimately this leads us to decide that we need to use a method such as column generation to greatly reduce the size of the matrix needed to perform the necessary calculations to find our multi-commodity flow.

## Column Generation

Column generation is the dual to the cutting plane method. It allows us to consider only a subset of the variables while keeping all of the constraints of the problem. Our original problem will now be called the master LP. We then consider a restricted LP which contains all the constraints of the original LP and a subset of the variables. Each iteration we solve a pricing problem (see if there are variables we should add to improve our objective function value) until we reach the optimal solution which means there are no more variables to add that will improve the objective function value.

For this to work with the multi-commodity flow problem we need to start with our master LP being the path formulation of the multi-commodity flow problem.

$$\begin{aligned}
& \max \sum_{p \in P} f_p \\
& s.t. \sum_{p \in P_a} f_p \leq u_a \quad \text{for all } a \in A \\
& \quad 0 \leq f_p \quad \text{for all } p \in P
\end{aligned}$$

So essentially with this formulation each column you add represents a path through the network that a commodity is sent along. So our restricted master LP is the constraints over a small starting number of paths  $P' \subseteq P$  which makes our restricted master LP:

$$\begin{aligned}
& \max \sum_{p \in P'} f_p \\
& s.t. \sum_{p \in P'_a} f_p \leq u_a \quad \text{for all } a \in A \\
& \quad 0 \leq f_p \quad \text{for all } p \in P'
\end{aligned}$$

And the dual of the restricted master problem is:

$$\begin{aligned}
& \min \sum_{a \in A} u_a \mu_a \\
& s.t. \sum_{a \in p} \mu_a \geq 1 \quad \text{for all } p \in P' \\
& \quad \mu_a \geq 0 \quad \text{for all } a \in A
\end{aligned}$$

By the duality theorem we know that a solution  $\mu$  to the restricted dual is a lower bound on a solution to the dual which also provides a lower bound to the primal problem. If both the primal and dual problem are feasible finding the optimal solution for the dual master problem will result in the optimal solution for the primal master LP. Below is a basic algorithm for solving the multi-commodity flow problem using column generation. Once the algorithm terminates with no other columns to add you have the optimal solution to the masterLP. Since the largest calculation done every iteration is coming up with the new pricing vector, if this pricing problem is solved in polynomial time then the master problem is solvable in polynomial time. Plus you no longer have the storage problems discussed above because you are only using the variables necessary to reach the optimal solution.

---

**Algorithm 1** Column Generation for MCF

---

```
1: repeat  
2:   Solve (MCF') for  $\mathcal{P}'$   
3:   Let  $\boldsymbol{\mu}$  be the dual variables for (D-MCF')  
4:   for  $i = 1, \dots, k$  do  
5:     Find a shortest  $(s_i - t_i)$ -path w.r.t. weights  $(\mu_a)$   
6:     If weight of shortest path  $p$  is less than 1, add  $p$  to (MCF')  
7:   end for  
8: until no path has been added
```

---

## Dantzig-Wolfe Decomposition

$$\max \mathbf{c}^T \mathbf{x}$$

$$s.t. \ N\mathbf{x} = \mathbf{0}$$

$$U\mathbf{x} \leq \mathbf{u}$$

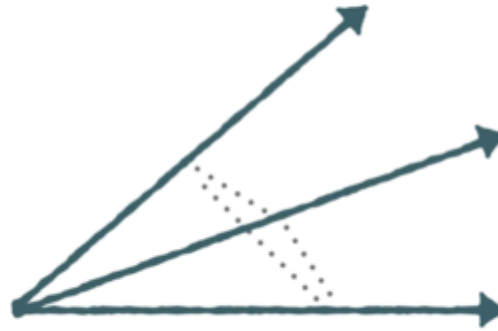
$$\mathbf{x} \geq \mathbf{0}.$$

*Flow conservation*

*Capacity Constraints*

*Non-negativity Constraints*

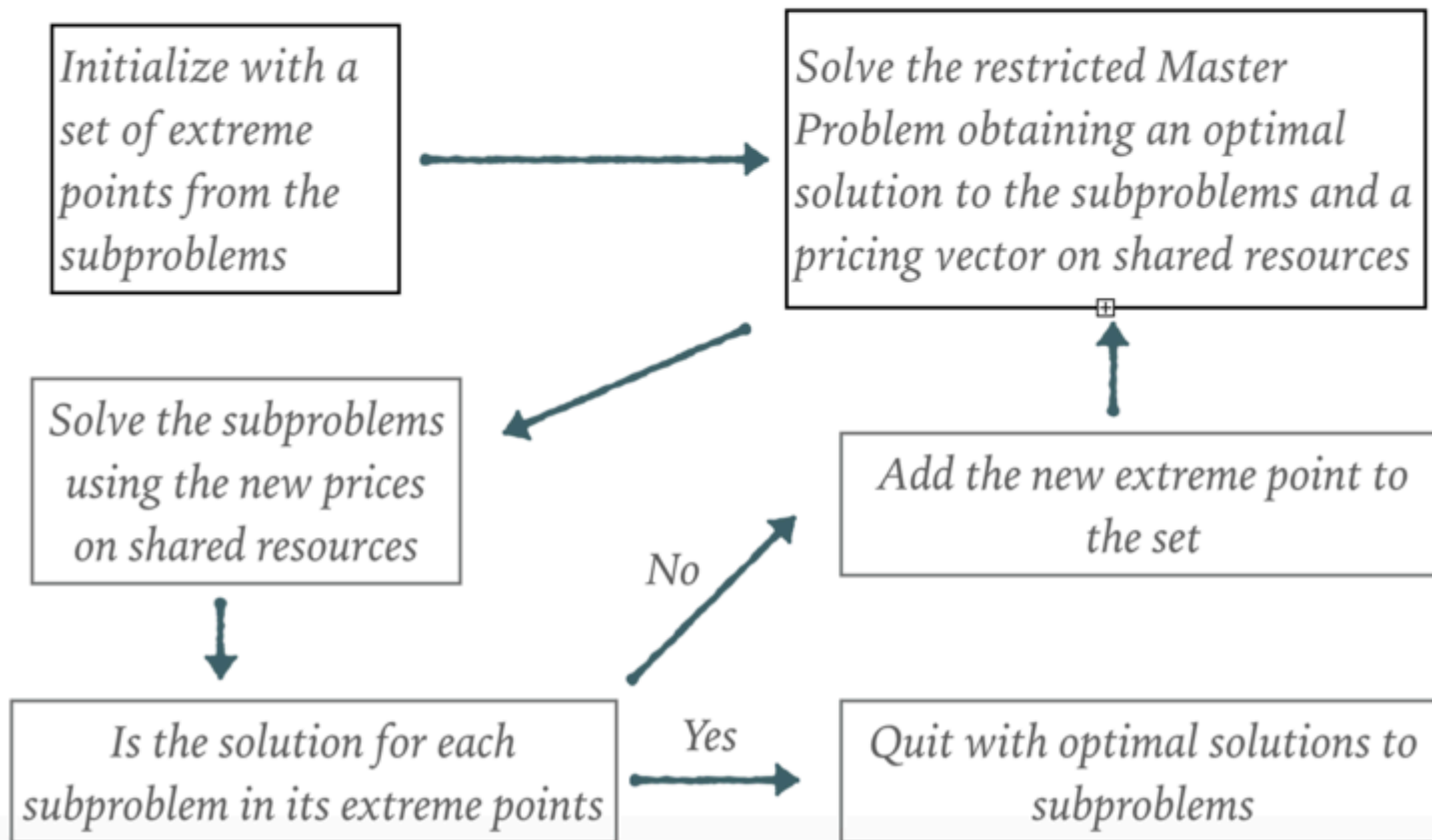
$$P := \{\boldsymbol{x} \geq \mathbf{0} : N\boldsymbol{x} = \mathbf{0}\}.$$





# Dantzig-Wolfe Decomposition

## Example



Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Multi-commodity\\_Flow&oldid=1568](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Multi-commodity_Flow&oldid=1568)"

- This page was last modified on 10 May 2018, at 18:16.
- This page has been accessed 9,781 times.

# Natalia VillegasFranco

From CU Denver Optimization Student Wiki

I am a highly motivated person who, through academic and consultant work, has developed a variety of programming skills and problem solving abilities.

I am very creative in the use of mathematical tools and their practical application. I am originally from Colombia and am working on my Master's Degree in Applied Mathematics at the University of Colorado Denver.

Here is my contribution to the Optimization Wiki:

Crime response planning by linear programing  
Spectral clustering

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Natalia\\_VillegasFranco&oldid=1282](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Natalia_VillegasFranco&oldid=1282)"

Category: Contributors

- 
- This page was last modified on 28 April 2018, at 22:50.
  - This page has been accessed 2,168 times.

# Nicholas Crawford

From CU Denver Optimization Student Wiki

Hello I am Nick (said it Groot voice). I am a first year PhD student at University of Colorado Denver. I got my B.S in Mathematics from Cal Poly, San Luis Obispo and my M.S is Applied Mathematics from San Jose State University in Applied Mathematics. My main research projects are finding algorithms for determining if a graph with certain bounds on minimum degree are 3-colorable. Other research projects I have worked on include Spectral Graph Theory and the student of Betti number diagrams. In my free time I enjoy many outdoor activities such as hiking, rock climbing, disc golf, and playing soccer.

## Projects

I worked with Zachary Sorenson on categorizing circuits for the Matching Problem. Our project was titled Circuits and Bloom's Algorithm.

I worked with Drew Horton on finding efficient Disaster Evacuations for Colorado.

I did a small project on Using AMPL and Python to solve the cutting stock integer program

I am currently working on a project called Primal and Strong Duality of Linear Programs.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Nicholas\\_Crawford&oldid=4299](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Nicholas_Crawford&oldid=4299)"

Category: Contributors

- 
- This page was last modified on 13 April 2023, at 10:55.
  - This page has been accessed 642 times.

# Nicholas Rogers

From CU Denver Optimization Student Wiki

## About me

Hello, my name is Nick Rogers. I am a PhD student at CU Denver. I am interested in applications of mathematics to biology and medicine.

## Education

I received my Bachelor's in Math with a minor in Computer Science from Elmhurst University in December of 2021.

## Projects

Fall 2023 - Optimizing SAR Paths in the Rocky Mountain Region, project with Matthew Knodell and Lillian makhoul.

Spring 2024 - Minimum Cost-to-Time Ratio Cycle Problem

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Nicholas\\_Rogers&oldid=4626](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Nicholas_Rogers&oldid=4626)"

Category: Contributors

- 
- This page was last modified on 9 April 2024, at 10:40.
  - This page has been accessed 50 times.

# One-way: To Improve Denver Traffic

From CU Denver Optimization Student Wiki

## Introduction

This project was an effort of Jacob Johns across the span of three months. The traffic data were provided by the Colorado Information Marketplace (<https://data.colorado.gov/Transportation/Road-Traffic-Counts-in-Colorado-2019/4j38-u7sj>) and some limited satellite street data from Google Maps (<https://www.google.com/maps>).

## Abstract

One-way roads have been hailed as an effective measure to increase traffic flow, decrease pedestrian accidents and reduce congestion. This is because there are fewer meeting points between cross traffic as well as the fact that pedestrians only have to check one direction to ensure safety. There have been recent reports that in practice, despite this logic, the latter claim that one-way streets reduce traffic does not hold. In this project, we aim to model the flow of traffic in a Denver neighborhood under current circumstances as well as under the stipulation of adding one-way streets. We use the linear programming techniques of network flows to make this model— initially with real world data, then again with predicted data. These predictions come from a linear regression model, which allows us to estimate the amount of traffic given certain conditions, such a road being one-way or not. Our results indicate that the City of Denver should not implement a policy to turn these streets into one-ways.

## Links

GitHub Repository (<https://www.github.com/jakeat555/TrafficProject>)

There are three primary folders in the GitHub: Administrative, Linear Regression and Linear Programming. In the Administrative folder, there are various presentations, graphics and an abstract. In the Linear Regression folder, there are both R code and html output for each of our linear modeling steps (Cleaning, Exploration, Visualization, Modeling). There are also the completed linear models, as well as the data. In the Linear Programming folder, there is R code and html output for the predictions of streets that were missing data. There is also the AMPL model and the AMPL data files for each of the three conditions we modeled.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=One-way:\\_To\\_Improve\\_Denver\\_Traffic&oldid=4484](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=One-way:_To_Improve_Denver_Traffic&oldid=4484)"

- 
- This page was last modified on 4 December 2023, at 15:34.
  - This page has been accessed 20 times.

# Optimization Solvers in Five Different Programming Languages

From CU Denver Optimization Student Wiki

## Abstract

Learning a new programming language to solve a mathematical program can be challenging. For many practitioners, it would be easiest to be able to access optimization solvers in one of the programming languages with which they are familiar. This project provides detailed instructions on the installation and use of optimization solvers such as CPLEX, GUROBI, and XPRESS, as well as information on accessing these solvers in five different programming languages. The core programming languages used are AMPL, MATLAB, R, Python, and C++. We show sample codes in these programming languages for linear, integer, quadratic, nonlinear, and semi-definite programs.

## Link to Github with Project file & sample codes

Here is the link which contains all the sample codes and the detailed procedure on how to installation solvers and calling in different programming languages. Link: <https://github.com/kushmakarb/Optimization-Solvers-Sample-Codes>

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Optimization\_Solvers\_in\_Five\_Different\_Programming\_Languages&oldid=2596"

- 
- This page was last modified on 30 April 2020, at 10:16.
  - This page has been accessed 1,023 times.

# Optimization Solvers

From CU Denver Optimization Student Wiki

## Abstract

Learning a new programming language to solve a mathematical program can be challenging. For many practitioners, it would be easiest to be able to access optimization solvers in one of the programming languages with which they are familiar. This project provides detailed instructions on the installation and use of optimization solvers such as CPLEX, GUROBI, and XPRESS, as well as information on accessing these solvers in five different programming languages. The core programming languages used are AMPL, MATLAB, R, Python, and C++. We show sample codes in these programming languages for linear, integer, quadratic, nonlinear, and semi-definite programs.

## Link to Github with Project file & sample codes

Here is the link which contains all the sample codes and the detailed procedure on how to installation solvers and calling in different programming languages. Link: <https://github.com/kushmakarb/Optimization-Solvers-Sample-Codes>

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Optimization\_Solvers&oldid=2595"

- 
- This page was last modified on 30 April 2020, at 10:16.
  - This page has been accessed 446 times.

# Optimization

From CU Denver Optimization Student Wiki

Optimization is the mathematical process of selecting the best solution or element from a set of solutions or elements. This is a simple process when given a small and finite set of elements, such as picking the largest of 5 integers, but becomes much more complex when dealing with multi-dimensional systems and infinite spaces. The basics of optimization are rather intuitive, since most people are subconsciously optimizing all the time. Everybody wants to get the best value for the products they buy, the most money for the hours they work, and the most joy out of activities they partake in. While most people aren't building mathematical models and solving them for these day-to-day activities, they partake in basic optimization every day. Optimization is commonly used in business analytics for finding the best method of doing business. Some uses include making the most profit, spending the least, creating the smallest environmental impact, generating the best public response, and using the least time to transport goods. Optimization has many different sub-fields of research.

## Contents

- 1 Uses for Optimization
  - 1.1 Business Analytics and Industrial Engineering
  - 1.2 Military Operations
  - 1.3 Engineering
- 2 History
- 3 Modern Optimization
  - 3.1 The Optimization Problem
    - 3.1.1 Linear Program
    - 3.1.2 Nonlinear Program
    - 3.1.3 Combinatorial Optimization
    - 3.1.4 Multi-Objective Optimization
- 4 Solving Methods
  - 4.1 Algorithms
  - 4.2 Using Computers
  - 4.3 Lagrange Multipliers
- 5 Duality Theory



# Uses for Optimization

## Business Analytics and Industrial Engineering

There are an endless number of ways to use mathematical optimization. Businesses hire mathematicians to increase their profits and reduce their costs. For example, consider a small business that is thinking about expanding. They want to know how much they should invest and how many new people they should hire to see the best returns for their money. This is an optimization problem for small businesses. Investing too much up front may bankrupt the business, while investing too little can hold them back and make it harder to expand in the future. Because there are so many variables associated with small businesses and expansion, this type of optimization problem takes mathematicians and computers to solve. Large businesses use optimization to find ways to run their business most efficiently, to minimize costs and maximize cash flow.

## Military Operations

Optimization is commonly used in military planning. Military operations around the world use optimization to find the optimal way to engage a target. In fact, Alan Turing and his group of mathematicians used statistics and optimization to decide the best intelligence to act upon once they retrieved it from the Enigma decoder they built. Optimization is used in military operations to find the quickest and least risky route to transport goods to ground troops. It is also used to find the best air route for air raid runs.

## Engineering

Electrical engineers use optimization to find the path of least resistance for a circuit. This type of optimization falls under the category of network flows and is similar to optimizing transportation of goods. That's right, semi-trucks and electrons have more in common than you thought.

## History

Newton and Gauss introduced the idea of iterative searches to find an optimal point for a function. Lagrange and Fermat expanded on Newton's work in optimization and calculus to create gradient and calculus based formulas for optimization. Mathematical programming was developed by Dantzig for the U.S. military, and Neumann used that research to develop Duality theory shortly after.

## Modern Optimization

Optimization is a branch of applied mathematics that uses computers to solve large problems that are programmed into them. modern computers are incredibly fast, but not fast enough to pick every point and test it until the best point is found. There are an infinite number of solutions to any continuous problem, and computers can still only work with discrete data. Therefore, computers use algorithms to jump between points or sets until an optimal solution is found. The most popular algorithm for linear programs is the simplex algorithm, developed by George Dantzig for the U.S. military in the 1940s. This is still the most widely used algorithm for linear systems because it has the shortest run time for any algorithm known. In essence, it is the optimal optimizer. There are other algorithms for non-linear optimization but they take longer to run.

# The Optimization Problem

Optimization problems take various forms, depending on the type of and relationship between the variables. They all have objective functions, constraints, and feasible regions. The objective function is the thing that is being optimized. This is the part of the problem that states whether you want to spend the least of a set of variables or receive the most of a set of variables. There can be multiple objective functions to a program. The constraints are bounds on what the optimal objective function value can be. They are combinations of variables that must be less than, greater than, or equal to a constant. The feasible region is the set of all points that satisfy all of the constraints. The optimal solution (or solutions) lie within the feasible region.

## Linear Program

Linear programs are a special type of program where all of the constraints are linear, and should be familiar to anyone that has taken linear algebra. They are programs with the set of  $m$  constraints and  $n$  variables defined by an  $m \times n$  matrix. A linear optimization problem takes the form:

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & \sum Ax^T = b \\ & x \leq \geq 0 \end{array}$$

With  $c^T x$  as the objective function, or the thing that being optimized.  $Ax^T$  is the constraint matrix, which consists of all the bounds for the problem.  $b$  is the set of right hand side values for the constraint matrix. The final line are bounds on  $x$ .

## Nonlinear Program

Nonlinear programs are mathematical programs that do not have linear constraints or a linear objective function. Nonlinear optimization problems take the form:

$$\begin{array}{ll} \min & f(\mathbf{x}) \\ \text{s.t.} & g_i(\mathbf{x}) \leq \geq \mathbf{b}_i \\ & \mathbf{x} \in X \end{array} \quad i = 1, 2, \dots, m$$

where  $X$  is the domain of definition for  $\mathbf{x}$ .

So a program is nonlinear if any of the functions  $f(\mathbf{x})$  or  $g_i(\mathbf{x})$  have nonlinear features. Common types of non-linear programs are quadratic, exponential, and logarithmic.

## Combinatorial Optimization

Combinatorial optimization deals with finite sets of data and finite possibilities for optimal values. This branch of optimization includes integer programming, discrete optimization, optimization of graphs, and other interesting applications such as the knapsack problem.

## Multi-Objective Optimization

Multi-objective optimization occurs when there is more than one objective function that is being optimized to the same set of constraints. Often, these objective functions are in contention with each other. For example, a business trying to maximize profit while minimizing cost, or trying to maximize output while minimizing environmental impact. In this case, there is usually not one optimal objective function value, but a whole set of points that are feasible. Each of these points will make one of the objective functions better and one of them worse. The set of these points is called the Pareto set.

## Solving Methods

Solving mathematical programs gets increasingly difficult as the programs gain more constraints and increase in dimension. Therefore, mathematicians rely on methods of solving these programs that make them easier to keep track of.

## Algorithms

There are a wide variety of algorithms used to solve mathematical programs. Each algorithm has advantages and disadvantages and works for different types of problems. The first algorithm created was the simplex algorithm. It was developed for the U.S. government in the 1960s to optimize scheduling for the military. This particular algorithm only works for linear programs. There are other algorithms for non-linear programs and integer programs. The Bellman-Ford algorithm is another common algorithm and is used to find the shortest path for a graph. There are even algorithms that jump between the primal problem and the dual problem until the optimal solution for both is found.

## Using Computers

Most mathematical programs are far too large to solve by hand. Therefore, computers are used to solve these programs exponentially quicker than humans can. Computers use pre-determined algorithms to solve programs based on the file type and program input parameters of the solver. For professionals in optimization, understanding programming is almost as important as understanding the math.

## Lagrange Multipliers

## Duality Theory

In mathematical programming, duality is the theory that any program can be both viewed and solved from two perspectives. The first is the primal program and the second is the dual program. Using both of these to look at a program provides deeper insight into the program functions. The dual problem is often used to give insight into the sensitivity of the primal problem. It can also be used to place bounds on the primal problem to make it easier to solve.

Topics:

Duality: Bounding the Primal

Duality: Economic Example

Shadow price

Complementary slackness

Lagrangian Duality

KKT Multipliers

- This page was last modified on 19 April 2017, at 12:40.
- This page has been accessed 402,275 times.

# Optimize the patrolling route

From CU Denver Optimization Student Wiki

Abstract: The Denver Metro area is administered by a multitude of policing zones and within these policing zones, crime incidences distribute unevenly. Optimizing the patrolling route for a police car is always a strategy which we can apply to improve policing. By analyzing historical crime data, we can shed light on which locations are more crime-prone than others. Using graph theory knowledge, we can simplify those locations as vertices in a graph, and then determining an optimized patrolling route can be reduced and solved as a classical TSP problem.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Optimize\\_the\\_patrolling\\_route&oldid=1955](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Optimize_the_patrolling_route&oldid=1955)"

---

- This page was last modified on 21 March 2019, at 12:27.
- This page has been accessed 411 times.

# Optimizing Highschool Graduation Rates

From CU Denver Optimization Student Wiki

This project is by Alyssa Newman Collin Powell and Zane Showalter-Castorena, and is for the Linear Programming course in Fall of 2020. The project will be presented at the Data to Policy Project on Friday, December 4th.

## Abstract

In this project, we consider existing data on variables which affects high school graduation rates in Colorado. Specifically, we are considering student teach ratio, average teacher salary, demographic disparity between teachers and students, and several others. This data is used to build a model to maximize the graduation rates of students in its purview. This model is solved using linear programming techniques. This would provide these institutions with a powerful tool for adjusting current budget usage and other resource allocation, relying on existing data. In order to demonstrate this model and the results it provides for decisions, we have created a mock school based on average constraints for budget, number of students, and other considerations from the state of Colorado. Ultimately this will provide an optimized distribution of resources and other policy questions.

## GitHub links

All information, including code, PDF, and data can be found at <https://github.com/Z-SC/OptimizingGraduationRates>

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Optimizing\_Highschool\_Graduation\_Rates&oldid=3088"

- 
- This page was last modified on 7 December 2020, at 14:03.
  - This page has been accessed 676 times.

# Optimizing SAR Paths in the Rocky Mountain Region

From CU Denver Optimization Student Wiki

## Contents

- 1 Contributors
- 2 Abstract
- 3 Code and Presentation
  - 3.1 AMPL Files
  - 3.2 Miscellaneous Files

## Contributors

This Project is by Matthew Knodell, Lillian makhoul, and Nicholas Rogers.

## Abstract

The Colorado Search and Rescue Association reports there are approximately 3000 annual search and rescue (SAR) cases spanning 8000 hours every year. It is important that this time is used in the most effective way to find the subject. We analyzed SAR patterns in the Rocky Mountain Region to find optimal patterns for teams to take. We use an integer-based optimization model to find optimal SAR patterns based on a given starting location for the SAR team and the subject's (last known) location. We then extend the model in the following ways: multiple SAR teams with different starting locations were dispatched, the grid was changed from a square to a rectangular region, and the type of incident was factored into the value of searching each grid space. Our model can serve as a guide for SAR associations to increase the likelihood of subjects being found as SAR teams search.

## Code and Presentation

Available on GitHub: <https://github.com/makhoullillian153/D2p-fall-2023>

Materials available:

### AMPL Files

These files contain the main code we used in this problem.

Assignment\_Model.mod contains the code for the assignment style problem.

Shortest\_Path\_Model.mod contains the code for the shortest path style problem.

Data\_File.dat contains the data file used for both the shortest path and assignment models

## Miscellaneous Files

Bidirectional Path and Value Calculations.ipynb contains the Python code to print out the valid paths between nodes in the region and to assigns the value of the nodes.

Data\_cleaning.Rmd contains the R code we used to filter and clean the data.

mission\_data.csv contains the cleaned data we used for this project.

Optimizing\_SAR\_Paths\_In\_Class\_Pres.pdf (and .pptx) contains our in-class presentation for our Fall 2023 Linear Programming class.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Optimizing\\_SAR\\_Paths\\_in\\_the\\_Rocky\\_Mountain\\_Region&oldid=4569](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Optimizing_SAR_Paths_in_the_Rocky_Mountain_Region&oldid=4569)"

- 
- This page was last modified on 5 December 2023, at 09:50.
  - This page has been accessed 38 times.



# Optimizing the patrolling route

From CU Denver Optimization Student Wiki

The Denver Metro area is administered by a multitude of policing zones and within these policing zones, crime incidents have different distributions. Optimizing the patrolling route for a police car is a strategy which we can apply to improve policing. By analyzing historical crime data, we can identify which locations would benefit the most using optimized patrolling routes. Using graph theory, we model those locations as vertices in a graph. Determining an optimized patrolling route can then be solved as a classical Travelling Salesman Problem.

## Contents

- 1 Motivation and Method
- 2 Model
  - 2.1 K-means clustering<sup>[3]</sup>
  - 2.2 Miller-Tucker-Zemlin formulation<sup>[4]</sup>
- 3 Implementation
  - 3.1 Results
- 4 Poster and Files
- 5 Summary and Further Works
- 6 References

## Motivation and Method

A number of police trials have suggested that patrolling, in particular foot patrolling, can significantly reduce crime incidents and disorders when they are targeted on crime “hotspots” at “hot times”. An experiment done by Minneapolis PD put twice as many policemen patrolling 55 crime hotspots have shown the “treatment group” sees a significant decline in crimes and public disorders compared with the “control group”<sup>[1]</sup>. In addition, British research has shown that targeted foot patrol can also improve public confidence in the police, perceptions of crime, and feelings of safety<sup>[2]</sup>.

So the question naturally arises, how to design the optimized patrolling route to maximize the positive effects of this practice?

In order to create a model for police patrolling, we have first to come up with a series of destinations that a policeman would need to pass through. I find the K-means clustering algorithm is particularly helpful for creating geometric centroids which can be processed as vertices on the graph. Luckily there is no need to rewrite the step by step code for K-means clustering in Matlab. A few command lines would do the work.

Once the centroids are found, we can plug it into AMPL and use Miller-Tucker-Zemlin formulation to find the shortest path traveling through those points. That is solving a Travelling salesman problem. I also explored the possibility to use K-means clustering to improve the design of precincts. This is done by group crime points according to its nearest police station.

## Model

### K-means clustering<sup>[3]</sup>

For  $n$  data points  $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$ , where each  $\mathbf{x}_i$  is a equally dimensional real vector,  $k$ -means clustering partitions the  $n$  data points into  $k$  ( $\leq n$ ) sets  $\mathbf{S} = \{S_1, S_2, \dots, S_k\}$ , so that the overall sum of the squares of the difference within cluster could be minimized. In formula, it finds:

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

where  $\boldsymbol{\mu}_i$  is the mean of points in  $S_i$ .

### Miller-Tucker-Zemlin formulation<sup>[4]</sup>

Given  $n$  nodes indexed from  $1$  to  $n$  and define:

$$x_{ij} = \begin{cases} 1 & \text{the path goes from node } i \text{ to node } j \\ 0 & \text{otherwise} \end{cases}$$

Denote  $c_{ij}$  to be the distance from node  $i$  to node  $j$  and introduce  $u_i$  as a dummy variable. Then we can formulate the TSP as the following integer linear programming problem:

$$\begin{aligned}
& \min \sum_{i=1}^n \sum_{j \neq i, j=1}^n c_{ij} x_{ij} : \\
& x_{ij} \in \{0, 1\} & i, j = 1, \dots, n; \\
& u_i \in \mathbf{Z} & i = 1, \dots, n; \\
& \sum_{i=1, i \neq j}^n x_{ij} = 1 & j = 1, \dots, n; \\
& \sum_{j=1, j \neq i}^n x_{ij} = 1 & i = 1, \dots, n; \\
& u_i - u_j + nx_{ij} \leq n - 1 & 2 \leq i \neq j \leq n; \\
& 0 \leq u_i \leq n - 1 & 2 \leq i \leq n.
\end{aligned}$$

The first and second set of constraints are degree constraints, which requires that each node can only be arrived at and departed from exactly once. The third constraint is the subtour elimination. It can be shown that (1) every feasible solution contains one and only one subtour and (2) every single tour which goes through all nodes, there is a set of feasible values for the  $u_i$  satisfying the constraints.

## Implementation

The data for this project is from the Data2policy website<sup>[5]</sup>. We used Matlab to draw graphs and use K-means clustering to group the data points. The first attempt to group all the data points at once cost too much computer running time, so we filter the data just by precinct 521 which significantly shrink the running time into seconds.

Once we find all the geometric centroids for the clusters and record their longitudes and latitudes, I can write an AMPL program to automatically calculate their distances with each other. Due to the limit of human resource in our team, we use geometric distance to approximate the traveling time between those centroids. If time permitted, a better model would incorporate a more accurate driving/walking time pulled out from Google Map. Plug into the AMPL program based on Miller-Tucker-Zemlin formulation. We can then calculate and shortest path through all of the centroids. Drawing lines between those centroids according to the 0,1 outputs from AMPL program would give us the shortest path.

# Results

Node	Order	Longitude	Latitude
1	1	-104.8408	39.7877
2	8	-104.8134	39.7942
3	5	-104.8285	39.7868
4	11	-104.8193	39.7870
5	14	-104.8415	39.7806
6	13	-104.8108	39.7752
7	12	-104.8118	39.7844
8	6	-104.8459	39.7805
9	7	-104.8430	39.7954
10	4	-104.8341	39.7940
11	2	-104.8304	39.7763
12	15	-104.8178	39.7777
13	3	-104.8239	39.7779
14	10	-104.8359	39.7790
15	9	-104.8238	39.7944

## Poster and Files

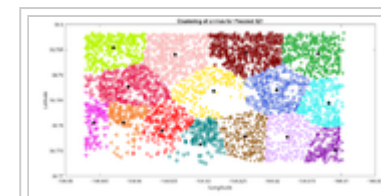
Please see my Github account for the poster and program files.

<https://github.com/LDDCZCN/IP-Final-Project.git>

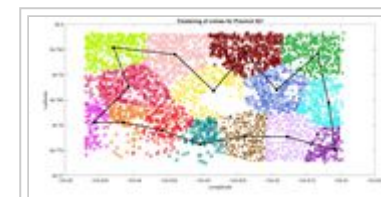
## Summary and Further Works

For a more efficient policing, we can use the K-means clustering technique to construct to precincts. Further improvements can be archived by incorporating more accurate transportation information from Google Map and the capacity of each police station.

Among those precincts, the above procedures would help design better patrolling routes. Since the crime data varies from period to period, for the practical purpose, a program which can automatically generate patrolling routes for every precinct by retrieving the data from the database and performing the above analysis would be applicable and more attractive.



**Figure 1:** Centroids of Precinct 521



**Figure 2:** Optimized Patrolling route of Precinct 521



**Figure 3:** New Precincts design for Denver

Divide the crime data into day time crimes and nighttime crimes and types, the above procedures may produce different patrolling routes and reveal new information on the distribution patterns of certain types of crimes.

## References

1. ↑ Sherman, L. and Weisburd, D. (1995). General Deterrent Effects of Police Patrol in Crime Hotspots: A Randomized Controlled Trial. *Justice Quarterly*, 12: 625–648.
2. ↑ Bradley, R. (1998). *Public Expectations and Perceptions of Policing*. London: Home Office.
3. ↑ [https://en.wikipedia.org/wiki/K-means\\_clustering](https://en.wikipedia.org/wiki/K-means_clustering)
4. ↑ [https://en.wikipedia.org/wiki/Travelling\\_salesman\\_problem](https://en.wikipedia.org/wiki/Travelling_salesman_problem)
5. ↑ City of Denver Police Department/Data Analysis Unit Country of Denver Crime. 2018.  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Optimizing\\_the\\_patrolling\\_route&oldid=2059](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Optimizing_the_patrolling_route&oldid=2059)"

- This page was last modified on 6 May 2019, at 18:09.
- This page has been accessed 3,673 times.

# Orlando Gonzalez

From CU Denver Optimization Student Wiki

## Contact Links

- [orlando.2.gonzalez@ucdenver.edu Email]

## Early Life and Background

Orlando Gonzalez was born in Jalisco, Mexico.

## Projects

Clustering Neighborhoods in Order to Analyze Policy Needs

([http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Clustering\\_Neighborhoods\\_in\\_Order\\_to\\_Analyze\\_Policy\\_Needs](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Clustering_Neighborhoods_in_Order_to_Analyze_Policy_Needs))

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Orlando\\_Gonzalez&oldid=4472](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Orlando_Gonzalez&oldid=4472)"

Category: Contributors

- 
- This page was last modified on 17 November 2023, at 02:50.
  - This page has been accessed 20 times.

# Paul Guidas

From CU Denver Optimization Student Wiki

## Contents

- 1 Contact Links
- 2 About Me
  - 2.1 Education
  - 2.2 Programming Languages/Experience
  - 2.3 Projects

## Contact Links

- Email (<mailto:paul.guidas@ucdenver.edu>)
- GitHub (<https://github.com/pgmath/>)

## About Me

My name is Paul Guidas and I am a BS/MS student studying applied mathematics with a focus in Operations Research/Optimization at CU Denver.

## Education

1. Colorado State University, General coursework towards Mechanical Engineering and Computer Science
2. Front Range Community College, General coursework towards Network Administration
3. University of Colorado Denver, B.S./M.S. in Applied Mathematics - Expected Graduation in May 2025

## Programming Languages/Experience

- AMPL
- Python

- R
- LaTeX

## Projects

- Fall 2023 - Emissions and Equality: Colorado Car Share Optimization with Colin Furey and Alana Saragosa.
- Spring 2024 - Cycle Cancelling Algorithm

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Paul\\_Guidas&oldid=4760](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Paul_Guidas&oldid=4760)"

Category: Contributors

- 
- This page was last modified on 1 May 2024, at 23:19.
  - This page has been accessed 50 times.



# Police and Fire Coordination: Response to Fatalities

From CU Denver Optimization Student Wiki

Police dispatch and Fire dispatch in Denver do not coordinate. However, they both respond to certain types of crashes. In order to facilitate cooperation between the different services, we will match up each Fire Station with a unique Police Station which it can coordinate well with. Specifically, we will solve a Many-to-One matching problem for Police Stations and Fire Stations, where the weights we are putting on the edges corresponds to how far away they are from car crashes. More specifically, we will solve the following integer program (where  $F$  is the set of Fire Stations, and  $P$  is the set of Police Stations):

$$\begin{aligned} &\text{Minimize} && \sum_{i \in P, j \in F} c_{ij} x_{ij} \\ &\text{Subject to} && \sum_{i \in P} x_{ij} = 1 \quad \forall j \in F \\ & && \sum_{j \in F} x_{ij} \geq 1 \quad \forall i \in P \\ & && x_{ij} \in \{0, 1\} \quad \forall i \in P, j \in F \end{aligned}$$

Where  $c_{ij} = \sum_{c \in C} (\text{dist}(c, i) + \text{dist}(c, j))$  and  $C$  is the set of all crashes we are considering.

Project by Eric Culver and Christina Ebben

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Police\\_and\\_Fire\\_Coordination:\\_Response\\_to\\_Fatalities&oldid=1967](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Police_and_Fire_Coordination:_Response_to_Fatalities&oldid=1967)"

- This page was last modified on 18 April 2019, at 18:19.
- This page has been accessed 731 times.

# Police vs Firefighters

From CU Denver Optimization Student Wiki

Police dispatch and Fire dispatch in Denver do not coordinate. However, they both respond to certain types of crashes. In order to facilitate cooperation between the different services, we will match up each Fire Station with a unique Police Station which it can coordinate well with. Specifically, we will solve a Many-to-One matching problem for Police Stations and Fire Stations, where the weights we are putting on the edges corresponds to how far away they are from car crashes. More specifically, we will solve the following integer program (where  $F$  is the set of Fire Stations, and  $P$  is the set of Police Stations):

$$\begin{aligned} &\text{Maximize} && \sum_{i \in P, j \in F} c_{ij} x_{ij} \\ &\text{Subject to} && \sum_{i \in P} x_{ij} = 1 \quad \forall j \in F \\ & && \sum_{j \in F} x_{ij} \geq 1 \quad \forall i \in P \\ & && x_{ij} \in \{0, 1\} \quad \forall i \in P, j \in F \end{aligned}$$

Where  $c_{ij} = \sum_{c \in C} (\text{dist}(c, i) + \text{dist}(c, j))$  and  $C$  is the set of all crashes we are considering.

Project by Eric Culver and Christina Ebben

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Police\\_vs\\_Firefighters&oldid=1949](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Police_vs_Firefighters&oldid=1949)"

- This page was last modified on 20 March 2019, at 16:46.
- This page has been accessed 1,267 times.

# Preflow-Push Algorithm

From CU Denver Optimization Student Wiki

This project is on the Preflow-Push Algorithm, used to find a maximum flow in a network.

## Contents

- 1 Contributor
- 2 GitHub repository
- 3 Abstract
- 4 Project Summary
  - 4.1 The Generic Preflow-push Algorithm
  - 4.2 Analysis of the generic Preflow-push Algorithm
    - 4.2.1 Theorem 1:
    - 4.2.2 Lemma 2:
    - 4.2.3 Lemma3:
    - 4.2.4 Lemma 4:
  - 4.3 Code Implementation of the generic Preflow-push Algorithm
  - 4.4 The FIFO Preflow-push Algorithm
  - 4.5 Complexity of the Preflow-push Algorithm
    - 4.5.1 Lemma 5:
      - 4.5.1.1 Proof:
    - 4.5.2 Theorem 6:
      - 4.5.2.1 Proof:
  - 4.6 A Comparison of the Two Implementations
  - 4.7 Examples of Networks
- 5 References

## Contributor

This project was completed by Jacob Dunham ([https://math.ucdenver.edu/~sborgwardt/wiki/index.php/Jacob\\_Dunham](https://math.ucdenver.edu/~sborgwardt/wiki/index.php/Jacob_Dunham)).

## GitHub repository

Preflow-Push Project Repository (<https://github.com/JacobDun/Preflow-Push-alg>)

Typesetting math: 100%

# Abstract

In this project we analyze two different implementations of the preflow-push algorithm. We first examine the generic preflow-push algorithm, and then compare this to the first-in, first-out(FIFO) preflow-push algorithm. We take a look at the running times of each of these implementations, and discuss examples of when the FIFO implementation performs either better than or very similar to the generic implementation.

## Project Summary

### The Generic Preflow-push Algorithm

Consider a capacitated network  $G = (N, A)$  with a nonnegative capacity  $u_{ij}$  associated with each arc  $(i, j) \in A$ . Define a source node  $s$  and a sink node  $t$  of  $G$ . We define a *preflow* as a function  $x : A \rightarrow \mathbb{R}$  such that:  $\sum_{\{j:(j,i) \in A\}} x_{ji} - \sum_{\{j:(i,j) \in A\}} x_{ij} \geq 0$  for all  $i \in N - \{s, t\}$  and  $0 \leq x_{ij} \leq u_{ij}$  for each  $(i, j) \in A$ . The

preflow-push algorithm will maintain a preflow at each intermediate stage of the algorithm. For some given preflow  $x$ , the *excess* of each node  $i \in N$  is given by:  $e(i) = \sum_{\{j:(j,i) \in A\}} x_{ji} - \sum_{\{j:(i,j) \in A\}} x_{ij}$ . Node that in any preflow,  $e(i) \geq 0$  for all  $i \in N - \{s\}$ . We refer to any node with a strictly positive excess as an active node.

The preflow-push algorithm works by selecting an active node and trying to remove its excess by pushing flow to its neighbors. The neighbors in which flow is pushed are those which are closer to the sink node  $t$ , where closeness is determined by distance labels which are updated throughout the algorithm. We refer to the distance of a node  $i$  as  $d(i)$ . We also note that flow is sent only through admissible arcs, where an arc  $(i, j)$  is admissible if it satisfies the condition that  $d(i) = d(j) + 1$ . Any other arcs are referred to as inadmissible.

### Analysis of the generic Preflow-push Algorithm

As we discussed the proof of many lemmas and the theorem of the runtime of the generic preflow-push algorithm in class, we will give just a brief overview of the argument of the runtime here as to not simply reiterate class materials.

#### Theorem 1:

The generic preflow-push algorithm runs in  $O(n^2m)$  time.

The key observations to support this argument are the following lemmas:

#### Lemma 2:

If the algorithm relabels any node at most  $k$  times, the algorithm saturates arcs at most  $km/2$  times.

**Lemma3:**

Each distance label increases at most  $2n$  times. Consequently, the total number of relabel operations is at most  $2n^2$ .

**Lemma 4:**

The generic preflow-push algorithm performs  $O(n^2m)$  nonsaturating pushes.

## Code Implementation of the generic Preflow-push Algorithm

An implementation of the generic preflow-push algorithm in Sage is given on GitHub, with guidance taken from <sup>[1]</sup> and <sup>[2]</sup>.

## The FIFO Preflow-push Algorithm

We now introduce the first-in,first-out(FIFO) implementation of the preflow-push algorithm as presented in <sup>[1]</sup>. In order to describe how the FIFO implementation differs from the generic implementation, we first describe the concept of *node examination*. For a selected node  $i$  in the generic preflow-push algorithm, the algorithm may perform a saturating/nonsaturating push or relabel the given node. Note that if the algorithm performs a saturating push, then the node  $i$  may still be active. If this is the case, the algorithm may then select a new node for a push/relabel operation. In the FIFO implementation, we implement a rule in which when the algorithm selects a node, it continues to push flow from that node until the node's excess becomes zero or the algorithm relabels the node. Note that the algorithm may then perform many saturating pushes followed by either a nonsaturating push or a relabeling. This sequence of operations is referred to as *node examination*. The FIFO implementation maintains the set **LIST** as a queue. It selects a node  $i$  from the front of **LIST**, performs pushes from this node, and adds newly active nodes to the rear of **LIST**. The algorithm examines node  $i$  until it becomes inactive or it is relabeled. If node  $i$  is relabeled, it is added to the rear of the queue. The FIFO implementation ends once the queue of active nodes is empty.

## Complexity of the Preflow-push Algorithm

To help us examine worst-case complexity of the FIFO algorithm, we first partition the total number of node examinations into phases. Phase one consists the node examinations for nodes that become active during the pre-process operation. Phase two is the the node examinations of all nodes in the queue after the algorithm has examined all nodes in phase one. We continue in this manner, noting that the algorithm examines any node at most once during a phase. We present proofs similar to those in <sup>[3]</sup>.

**Lemma 5:**

The FIFO algorithm performs at most  $4n^2$  phases.

**Proof:**

Consider the potential function given by  $\Phi = \max\{d(i) : i \text{ is active}\}$  , and let  $\Phi = 0$  when there are no active nodes. We are interested in the initial value and final value of  $\Phi$  during a given phase. We consider two cases:

Case 1: The algorithm performs at least one relabel operation during a phase. In this case  $\Phi$  may increase by the maximum amount a distance label can increase by. Thus, by lemma 3, the total increase in  $\Phi$  over all phases is at most  $2n^2$ .

Case 2: The algorithm does not perform an relabel operations during a phase. Then the excess of each node that was active at the beginning of the phase moves to nodes with smaller distance labels. Therefore,  $\Phi$  must decrease by at least one unit.

As  $\Phi$  has initial value zero, final value zero, and is nonnegative, the total number of phases in which  $\Phi$  decreases can be at most the total increase of  $\Phi$  over all phases. Thus, the number of phases of type 2 is at most  $2n^2$ . As there are at most  $2n^2$  distance label updates, there are also at most  $2n^2$  phases of type 1. Therefore, there are at most  $4n^2$  phases.

**Theorem 6:**

The FIFO preflow-push algorithm runs in  $O(n^3)$  time.

**Proof:**

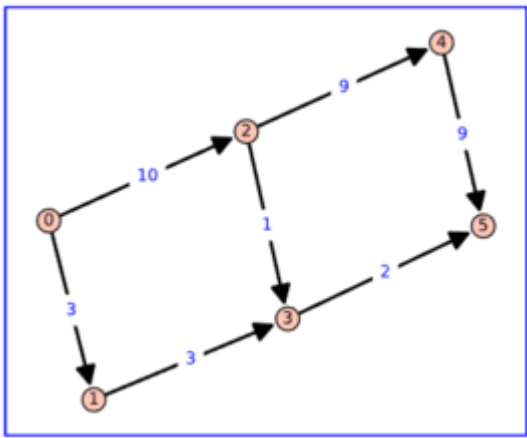
As each phase examines any node at most once and each node examination performs at most one nonsaturating push, the FIFO preflow-push algorithm performs at most  $n(4n^2) = 4n^3$  nonsaturating pushes by lemma 5. Therefore, the FIFO preflow-push algorithm performs  $O(n^3)$  nonsaturating pushes. As the bottleneck operation for the generic preflow-push algorithm is the number of nonsaturating pushes, we have that the FIFO preflow-push algorithm runs in  $O(n^3)$  time.

## A Comparison of the Two Implementations

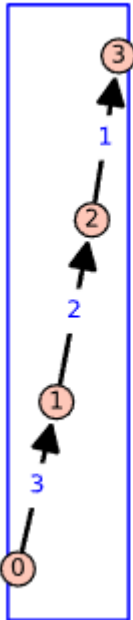
When looking at the running times of these two implementations, it's clear the FIFO implementation has the potential to far outperform the generic implementation when running on networks that have many edges. For instance, a network with multiedges(nonparallel) can have up to  $n^2 - n$  edges, in which case the generic preflow-push algorithm then runs in  $O(n^4)$  time. It's also worth noting that in a connected network in which each arc has a corresponding arc in the opposite direction(which we assume), the minimum number of edges is  $2n - 2$ . Thus, for any connected network under our assumptions,  $n^3 \leq n^2 m$ . This implies we should in practice just always implement this node examination rule for any implementation of the preflow-push algorithm. This is in fact what is done in practice.

## Examples of Networks

Below we show two examples of networks. One in which FIFO outperforms the generic algorithm and another in which they perform similarly. A deeper analysis of the performances on these networks is on GitHub.



FIFO outperforms generic



FIFO similar to generic

## References

1. <sup>1.0</sup> <sup>1.1</sup> <sup>1.2</sup> Ahuja, Ravindra K., Thomas L. Magnanti, and James B. Orlin. "Network flows." (1988).
2. <sup>↑</sup> [https://en.wikipedia.org/wiki/Push%E2%80%93relabel\\_maximum\\_flow\\_algorithm](https://en.wikipedia.org/wiki/Push%E2%80%93relabel_maximum_flow_algorithm)
3. <sup>↑</sup> David P. Williamson and June Andrews "https://people.orie.cornell.edu/dpw/orie633/LectureNotes/lecture5.pdf"

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Preflow-Push\\_Algorithm&oldid=4826](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Preflow-Push_Algorithm&oldid=4826)"

- This page was last modified on 2 May 2024, at 11:41.
- This page has been accessed 83 times.



# Primal and Strong Duality of Linear Programs

From CU Denver Optimization Student Wiki

## Contents

- 1 Abstract
- 2 Linear Programming basics
- 3 Duality
  - 3.1 Farkas' Lemma
  - 3.2 Strong Duality Theorem
  - 3.3 Weak Duality Theorem
- 4 Examples
- 5 References

## Abstract

A linear program is a problem of maximizing or minimizing a linear function of several variables to a system of linear constraints. The theory of linear programming concerns itself with the study of duality. Duality theory is established through a series of statements about linear constraints which lead to Farkas' Lemma and thus the existence of the duality theorems. These results give bounds on the optimal solutions of the primal (original) linear program. In this project we will look at both duality theorems and their implications on solving linear programs. We will also look at several applications of duality extending to the realm of network flows and graph theory.

## Linear Programming basics

Linear programs are program where all of the constraints are linear. They have a set of  $m$  constraints and  $n$  variables defined by an  $m \times n$  matrix. A linear optimization problem takes the form:

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & \sum Ax^T = b \\ & x \leq \text{or} \geq 0 \end{array}$$

With  $c^T x$  as the objective function,  $Ax^T$  is the constraint matrix.  $b$  is the set of right hand side values for the constraint matrix. There are also conditions on  $x$ .

# Duality

In this section we will look at important results from duality theory. The first result is Farkas' Lemma. This lemma is key to proving the strong duality theorem.

## Farkas' Lemma

The statement of Farkas' Lemma is:

**Farkas' Lemma:** Let  $b \in \mathbf{R}^m$ . Either there exists  $x \in \mathbf{R}^n$  such that  $x \geq 0$  and  $Ax = b$ , or there exists  $y \in \mathbf{R}^m$  such that  $y^t A \geq 0$  and  $y^t b = -1$ .

**Proof of Farkas' Lemma:** Let  $C = \{Ax : x \in \mathbf{R}^n, x \geq 0\}$  and suppose that the 'or' case does not hold, so  $b \notin C$ . Let  $B$  be the closed ball of radius  $R$  about  $b$ , where  $R \geq \|b\|$ , so  $B \cap C \neq \emptyset$ . Since  $C$  is closed and  $B$  is compact, there is a closest vector  $w \in B \cap C$  to  $b$ . If  $u \notin B$  then  $\|b - u\| \geq R$ , whereas  $\|b - w\| \leq R$ , so  $w$  is a closest vector in  $C$  to  $b$ .

Take any  $v \in C$  and consider the line segment joining  $v$  to  $w$ . Since  $C$  is convex, this segment lies in  $C$ . Therefore for any  $\alpha$  such that  $0 \leq \alpha \leq 1$ , we have

$$\begin{aligned}\|w - b\|^2 &\leq \|\alpha v + (1 - \alpha)w - b\|^2 \\ &= \|\alpha(v - w) + (w - b)\|^2 \\ &= \alpha^2\|v - w\|^2 + 2\alpha\langle v - w, w - b \rangle + \|w - b\|^2.\end{aligned}$$

Hence, taking  $\alpha$  to be a small positive number we have  $\langle v - w, w - b \rangle \geq 0$ .

Set  $z = w - b$ . Setting  $v = 0$  we get  $\langle w, z \rangle \leq 0$ , and so

$$\langle b, z \rangle = \langle b - w, z \rangle + \langle w, z \rangle = -\|w - b\|^2 + \langle w, z \rangle < 0$$

Hence there exists  $\gamma < 0$  such that

$$\langle v, z \rangle \geq \langle w, z \rangle > \gamma > \langle b, z \rangle$$

for all  $v \in C$ . Fix  $v \in C$ . Since  $\langle \lambda v, z \rangle > \gamma$  for all  $\lambda \geq 0$  we have  $\langle v, z \rangle > \gamma/\lambda$  for all  $\lambda > 0$ . Taking  $\lambda$  large enough,  $\langle v, z \rangle \geq 0$ . Therefore  $y = z/|\langle z, b \rangle|$  satisfies the required conditions for the first case.

**Farkas' Lemma Variant** Let  $b \in \mathbf{R}^m$ . Either there exists  $x \in \mathbf{R}^n$  such that  $Ax \leq b$ , or there exists  $y \in \mathbf{R}^m$  such that  $y \geq 0, y^t A = 0$  and  $y^t b = -1$ .

We will now prove that Farkas' Lemma and the variant are equivalent. Suppose that Farkas' Lemma holds. If the 'or' case of the variant fails to hold then there is no  $y \in \mathbf{R}^m$  such that

and  $y^t b = -1$ . Hence, by Farkas' Lemma, there exists  $x \in \mathbf{R}^n$  and  $z \in \mathbf{R}^m$  such that  $x \geq 0, z \geq 0$  and

$$\begin{pmatrix} A & I_m \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} = b$$

Therefore  $Ax \leq b$  and the first case of the variant holds.

Now suppose that the variant holds. If the first case of Farkas' Lemma fails to hold then there is no  $x \in \mathbf{R}^n$  such that  $x \geq 0$  and

$$\begin{pmatrix} A & -b \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix} = 0.$$

Let  $c = (0, \dots, 0, 1) \in \mathbf{R}^{n+1}$ . Thus there is no  $w \in \mathbf{R}^{n+1}$  such that  $w \geq 0$ ,

$$w^t \begin{pmatrix} A^t \\ -b^t \end{pmatrix} = 0$$

and  $-w^t c = -1$ . Hence, by Farkas' variant, there exists  $y \in \mathbf{R}^m$  such that  $y \geq 0$  and

$$\begin{pmatrix} A^t \\ -b^t \end{pmatrix} y \leq c.$$

Thus  $A^t y \leq 0$  and  $b^t y \geq 1$ . By scaling we may assume that  $b^t y = 1$ , as required for the second case of Farkas' Lemma.

## Strong Duality Theorem

With Farkas' Lemma above we will prove the following theorem:

**Strong Duality Theorem** If both the primal and dual problems are feasible then they have the same optimal solution.

**Proof** Let  $\tau_P \in \mathbf{R}$  be the optimal value of the primal problem and let  $\tau = \tau_P + \varepsilon$ . Since there exists no  $x \in \mathbf{R}^n$  such that  $Ax \leq b, c^t x \geq \tau$ , there exists no  $x \in \mathbf{R}^n$  such that

$$\begin{pmatrix} A \\ -I_n \\ -c^t \end{pmatrix} x \leq \begin{pmatrix} b \\ 0 \\ -\tau \end{pmatrix}.$$

By the variant of Farkas' theorem there exist  $y \in \mathbf{R}^m, z \in \mathbf{R}^n$  and  $\alpha \in \mathbf{R}$  such that  $(y^t z^t \alpha) \geq 0$ ,

$$\begin{pmatrix} y^t & z^t & \alpha \end{pmatrix} \begin{pmatrix} A \\ -I_n \\ -c^t \end{pmatrix} = 0$$

and  $y^t b - \alpha \tau < 0$ . Thus  $y^t A = z^t + \alpha c^t$  and  $y^t b < \alpha \tau$ .

Suppose that  $\alpha = 0$ . Then, since  $y^t A = z^t \geq 0$  and  $y^t b = -1$ , thus the dual problem is either infeasible or unbounded. This contradicts the Weak Duality Theorem since, by assumption, both problems are feasible. Therefore  $\alpha \neq 0$  and  $\alpha = 1$ . So  $y^t A \geq c^t$  and  $y^t b < \tau$ . Hence if  $\tau_D \in \mathbf{R}$  is the optimal value of the dual problem then  $\tau_D < \tau = \tau_P + \varepsilon$ . By the Weak Duality Theorem it follows that  $\tau_P \leq \tau_D < \tau_P + \varepsilon$ . Since  $\varepsilon$  was arbitrary we have  $\tau_P = \tau_D$ .

## Weak Duality Theorem

**Weak Duality Theorem:** If  $x$  is feasible for the primal problem and  $y$  is feasible for the dual problem then  $c^t x \leq y^t b$ .

**Proof:** From  $Ax \leq b$  and  $c^t \leq y^t A$  we get  $c^t x \leq y^t Ax \leq y^t b$  (Fix this)

A nice implication of this is that if the primal problem is feasible then the dual problem is bounded, and if the dual problem is feasible then the primal problem is bounded. The converses to these statements do not hold, because it is possible that both primal and dual problems are infeasible (and so, immediately from the definition, they are both bounded): for example, take

$$A = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad b = \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \quad c = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Then for any  $x \in \mathbf{R}^2$  such that  $x \geq 0$  we have  $Ax \not\leq b$ , and for any  $y \in \mathbf{R}^2$  such that  $y \geq 0$  we have  $y^t A \not\geq c^t$ .

## Examples

In this example we will construct the dual program from the primal program. The way we do this is by taking the negative constraint matrix and switching the values on the right hand side for the negative objective function values. Consider the example below:

$$\begin{array}{llllll} \text{max} & x_1 & - & x_2 & + & 3x_3 & \\ \text{s.t.} & x_1 & + & x_2 & + & x_3 & \leq 10 \\ & 2x_1 & - & x_2 & - & x_3 & \leq 2 \\ & 2x_1 & + & 2x_2 & + & 3x_3 & \leq 8 \\ & x_1 & , & x_2 & , & x_3 & \geq 0 \end{array}$$

point in the primal problem given. For example,  $(1, 1, 1)$  is a feasible point. The objective function value for this point is

$$1(1) - 1(1) + 3(1) = 3$$

Since  $(1, 1, 1)$  is a feasible point, we know that the optimal objective function value is either at this point or at a point of greater value. Thus, any feasible point provides a lower bound on the objective function value. This function value can be improved with another feasible solution  $(0, 0, 2)$ . This gives the objective function value of 6. Now that we have a lower bound for our objective function, our next step is to get an upper bound. We do this by adding constraints together:

$$x_1(1 + 2 + 2) + x_2(1 - 1 + 2) + x_3(1 - 1 + 3) \leq (10 + 2 + 8)$$

$$\text{simplifying we get: } 5x_1 + 2x_2 + 3x_3 \leq 20$$

Since each variable is greater than or equal to 0, we know

$$x_1 - x_2 + 3x_3 \leq 5x_1 + 2x_2 + 3x_3 \leq 20.$$

This implies that 20 is an upper bound on the objective function value. The difference between these 6 and 20 is called the **duality gap**. The goal of many optimization problems is to make this gap as small as possible (zero is ideal) Since linear programs are always convex, the duality gap is 0. The only time there will be non-zero gap is if the program is concave. (Reference)

The upper bound can be improved by adding the constraints together in different ways to get a lower upper bound. For example, adding the 2 of the third constraint to the second constraint yields:

$$2(2x_1 + 2x_2 + 3x_3 \leq 8) + (2x_1 - x_2 - x_3 \leq 2) = 6x_1 + 3x_2 + 5x_3 \leq 18$$

which is a lower upper bound than the previous one. Our goal is to find the the greatest lower bound (infimum) of the upper bounds. We accomplish this task by assigning variables to each of the constraints and then solving for those variables to find an optimal solution. Using the variable  $y$ , the constraints become:

$$y_1(x_1 + x_2 + x_3 \leq 10)$$

$$y_2(2x_1 - x_2 - x_3 \leq 2)$$

$$y_3(2x_1 + 2x_2 + 3x_3 \leq 8)$$

Simplifying

$$x_1y_1 + x_2y_1 + x_3y_1 \leq 10y_1$$

$$2x_1y_2 - x_2y_2 - x_3y_2 \leq 2y_2$$

$$2x_1y_3 + 2x_2y_3 + 3x_3y_3 \leq 8y_3$$

Dropping the inequalities and rewriting

$$x_1(y_1 + y_2 + y_3)$$

$$x_2(2y_1 - y_2 - y_3)$$

$$x_3(2y_1 + 2y_2 + 3y_3)$$

We assume that each  $x$  variable's coefficient is at least as large as its objective function coefficient, so

$$\begin{array}{ccccccc} y_1 & + & y_2 & + & y_3 & \geq & 1 \\ 2y_1 & - & y_2 & - & y_3 & \geq & -1 \\ 2y_1 & + & 2y_2 & + & 3y_3 & \geq & 3 \end{array}$$

Bringing back in the inequalities and remembering that each of the variables must be less than their corresponding coefficients, implies that is the upper bound. In order to find the least upper bound, we minimize and get the following program

$$\begin{array}{llllllll} \min & 10y_1 & + & 2y_2 & + & 8y_3 & & \\ s.t & y_1 & + & y_2 & + & y_3 & \geq & 1 \\ & 2y_1 & - & y_2 & - & y_3 & \geq & -1 \\ & 2y_1 & + & 2y_2 & + & 3y_3 & \geq & 3 \\ & y_1 & , & y_2 & , & y_3 & \geq & 0 \end{array}$$

This program is the dual of the primal optimization problem. It is also possible to follow this same logic and get back to the primal problem.

# References

1. Robert J. Vanderbei. Linear Programming: Foundations and Extensions, 4th edition, Springer, 2014
2. Stephen Boyd and Lieven Vandenberghe. Convex Optimization, Cambridge University Press
3. Royal Hollyway. Duality of Linear programs, University of London  
Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Primal\_and\_Strong\_Duality\_of\_Linear\_Programs&oldid=4380"

- This page was last modified on 4 May 2023, at 12:24.
- This page has been accessed 154 times.

# Programming Files

From CU Denver Optimization Student Wiki

The most common programming file types are .lp and .mps files. These are both file types that can be read by a wide variety of solvers. Many languages have solvers for these built in. For example, Python has lpsolve55 as its default solver for .lp and .mps files. Python will be the standard language discussed, as most mathematicians have worked with it. The solver for python, lpsolve55, was written for C++ and modified to work for python, which makes some of the implementation difficult in Python. However, because Python is a higher level language and therefore further away from machine code, it is easier for humans to read and consequently understand. Therefore, this page will be written using Python.

## Contents

- 1 .lp Files
- 2 .mps Files
  - 2.1 Dualizing a File
- 3 References

## .lp Files

A .lp file is the most intuitive linear program file, as it is written much like a linear program in standard form. It has an objective function, constraints, inequalities and equalities, and non-negaivity constraints where necessary. The following linear program is the same program found in the shadow price section of the wiki, but written as if it was a .lp file:

```
max: 40 x 1 50 x 2 ;  
x 1 x 2 <= 1000;  
x 1 <= 700;  
2 x 2 <= 1100;  
x 1 >= 300;  
x 2 >= 300
```

Is the exact input into Python for a linear program once lpsolve55 has been called. Notice that there are no addition signs. This is because an .lp format can only handle linear programs. However, if the variables were multiplied together the program would become non-linear. The solver that Python uses has a variety of algorithms that can be called to use for solving any .lp file. These range from the simplex algorithm, which is the standard algorithm for lpsolve, to primal-dual algorithms and dual solve methods. The .lp file can also be generated in Python, by writing a program in this form and then commanding "return = lpsolve('write\_lp', lp, filename)" instead of "return = lpsolve('solve\_lp', lp)".<sup>[1]</sup>

While Python is a very easy language to do this in, there are some IDLEs that have interfaces that are tuned more for linear program writing, such as Lp\_Solve. This program can be used to write .lp files and solve them, and can be imported as a package into Python, Matlab, and many other languages to be solved in whatever language the user prefers to code in.

## .mps Files

.mps (mathematical programming system) files contain the same information that .lp files contain, but are written differently. While a .lp file is written vary similar to the conventional way a linear program is written, while a .mps file is written in column format. For comparison, the previous program written in .mps form looks like:

name		friction				
n	cost					
l	lim1					
l	lim2					
l	lim3					
g	dem1					
g	dem2					
columns						
	xone	cost	40	lim1	1	
	xone	lim2	1	dem1	1	
	xtwo	cost	50	lim1	1	
	xtwo	lim3	2	dem2	1	
rhs						
	rhs1	lim1	1000	lim2	700	
	rhs1	lim3	1100	dem1	300	
	rhs1	dem2	300			
enddata						

Notice how the program individually lists all of variables and their coefficients with the particular constraint.

## Dualizing a File

In Python, generating the dual program is rather simple. Instead of creating a whole new program, one can simply take the known .lp file, import it after importing the lpsolve package, and run the following code:

```
[obj, x, duals, return] = lpsolve('get_solution', lp)
```

This code will then print the objective function value, the dual objective function value, and the duals.

In order to get a file of the dual program, access the directory in which the .lp or .mps file is contained, then import the lpsolve library and run the following code.

```
from lpsolve55 import *
lp = lpsolve('read_LP', 'My_Lp.lp')
lpsolve('solve', lp)
dlp = lpsolve('copy_lp', lp)
lpsolve('dualize_lp', dlp)
lpsolve('write_lp', dlp, 'My_DLp.lp')
```



This will return a file into the same directory as the file that My\_Lp.lp is under the name My\_DLp.lp.

## References

1. ↑ "Lpsolve reference guide." Lpsolve 5.5 Referance Guide, 1999, [lpsolve.sourceforge.net/5.5/](http://lpsolve.sourceforge.net/5.5/). Accessed 15 Mar. 2017.  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Programming\\_Files&oldid=273](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Programming_Files&oldid=273)"

- 
- This page was last modified on 28 March 2017, at 18:20.
  - This page has been accessed 13,931 times.

# Queue-Based Strategy to Achieve Maximum Stable Rate in Multi-user Network

From CU Denver Optimization Student Wiki

In this project, we will explore a strategy determined by linear programming to stabilize queues in a multicast communication network. The strategy will maximize the communication rate with no loss of packets.

This project is for MATH5593 Linear Programming at the University of Colorado-Denver, Fall 2019. Student contributors: John McFarlane and Sajjad Nassirpour

## Contents

- 1 Abstract
- 2 Motivation
- 3 Queue-based strategy
- 4 System Stabilizing
- 5 Link to GitHub page
- 6 References

## Abstract

Nowadays, multi-user wireless networks are gaining more and more popularity due to their high efficiency. For instance, in LTE cellular networks, transmitters can deliver its packets to some users at the same time and service different users with the same infrastructure. There are a lot of communication models which are used in multi-user networks. In this project, we focus on multicast channel model which means transmitters should deliver packets to all users and we need a communication strategy to deliver all packets successfully. In this project, we use a queue-based strategy to handle the packets in our scheme and use network codes probabilistically to ensure packets are delivered to all users. Most ideal studies on network coding assume infinite queue length. We constrain queues to finite length and consider more practical multi-user networks with bounded queue lengths.

In the new stable queuing model, each packet in a queue is associated with an index set indicating users that still require the packet. Our objective is to maximize the input rate under the queue stability constraint that no packet can be dropped from overflowing the queues. We will first introduce some basic and important concepts of multi-user networks, will formulate as a linear programming problem, and propose a network coding-based packet scheduling scheme that finds the optimal solution. Finally, the simulation results with AMPL corroborate our method.

# Motivation

## Example of Multi-user Network

In our multicast transmission model (Figure 1), we want to maximize the rate of packet delivery and transmit all packets to our users with zero packet losses. This project will focus on transmissions inside a subnet of the larger internet, where there is only one packet source transmitting to a given number of receiver clients (see blue box).

Assume that the transmitter has a finite number of packets it can store in buffers for later transmission. This problem is the impetus for the communication strategy discussed in this project.

Assume that the  $N$  channels between the transmitter and each of the  $N$  receivers are packet erasure channels. That is, there are  $N$  independent probabilities  $\epsilon_i$  that a packet sent from the transmitter is not received at a given receiver and there are  $N$  independent probabilities  $\gamma_i = 1 - \epsilon_i$  that a packet sent from the transmitter is received at a given receiver. Assume that the  $N$  channel erasure probabilities can be observed at the transmitter. Assume the channels are slow-fading. That is, assume the channels have a long coherence time compared to the time they are in use.

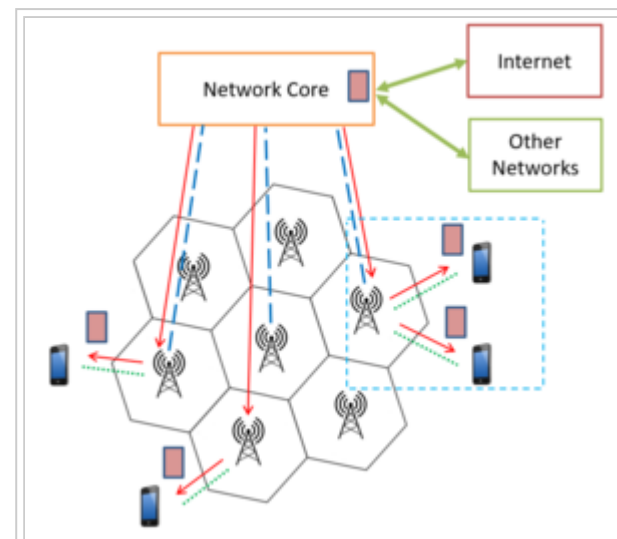


Figure 1: Multi-user Network

## Multicast Transmission Model

Assume packets are stochastically (randomly) generated at the source of the system with a variable rate  $\lambda_{in} = \mathbb{E}[A_0(t)]$  (Figure 2). Assume that the packet arrivals rate is stationary, meaning that it does not change over time.

Newly arriving packets at the transmitter are placed in the initial queue. Call the initial queue in the system queue 0 to indicate that none of the receivers in the system have received the packets in this queue. All receivers in the system require these packets and the system manages additional queues (discussed in section "Queue-based strategy" below) according to where the packet has been received.

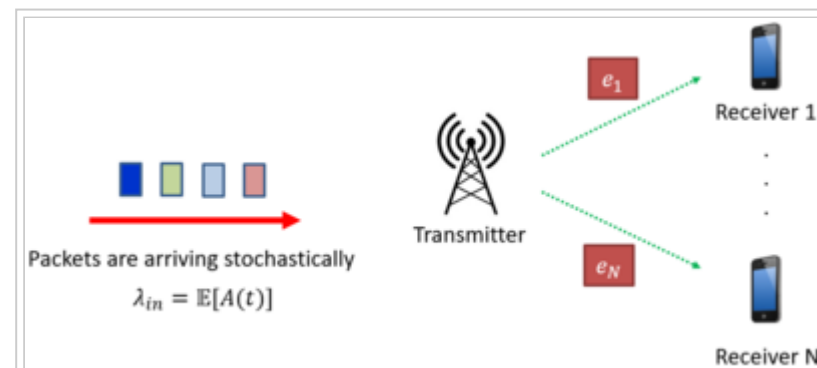


Figure 2: Source Packet Generation

Because our goal is to deliver packets to all receivers,  $\lambda_{out} = \lambda_{in} = \lambda$  and we wish to maximize  $\lambda$  (Figure 3). The transmitter in this system has finite buffer lengths, so the transmitter must transmit the packets in a manner that is most efficient given transmission success probabilities from the transmitter to each receiver. The transmission strategy in this project uses queue-based packet management.

# Queue-based strategy

## Details Of Queue-based strategy

The queues managed at the transmitter are defined by the set of users which still need to receive the packets. When a packet moves to another queue where it is still required by a number of receivers, it must still be retransmitted using network coding.

The queues/index sets can be described as follows:

$Q_0$   
The initial queue in the system that indicates that all receivers {1, 2, 3} in the system require the packets in this queue. As soon as a packet is transmitted and successfully received at any queue, the packet will leave this queue.

$Q_1$   
A queue in the system that indicates that receivers {1, 2} in the system require the packets in this queue. This queue can be reached by packets from  $Q_0$  if the former  $Q_0$  packets are received at receiver 3.

$Q_2$   
A queue in the system that indicates that receivers {2, 3} in the system require the packets in this queue. This queue can be reached by packets from  $Q_0$  if the former received at receiver 1.

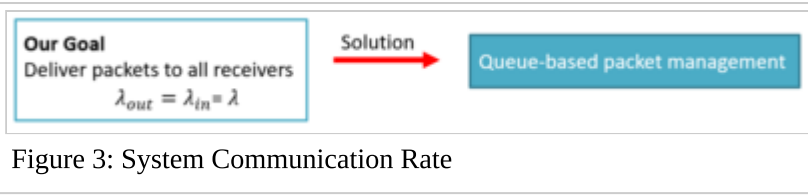


Figure 3: System Communication Rate

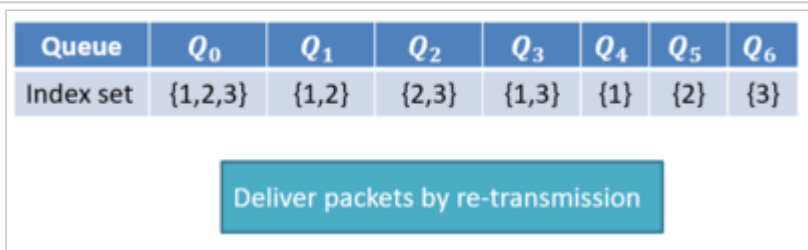


Figure 4: Packet Queues in System

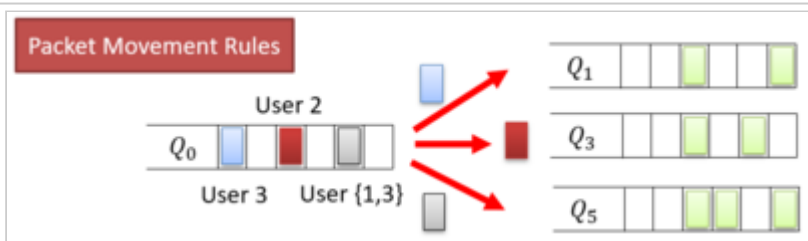


Figure 5: Queue Movement for Packets

$Q_3$

A queue in the system that indicates that receivers  $\{1, 3\}$  in the system require the packets in this queue. This queue can be reached by packets from  $Q_0$  if the former  $Q_0$  packets are received at receiver 2.

$Q_4$

A queue in the system that indicates that receiver  $\{1\}$  in the system requires the packets in this queue. This queue can be reached by packets from  $Q_0$  if the former  $Q_0$  packets are received at receivers  $\{2, 3\}$ , from  $Q_1$  if the former  $Q_1$  packets are received at receiver 2, or from  $Q_3$  if the former  $Q_3$  packets are received at receiver 3.

$Q_5$

A queue in the system that indicates that receiver  $\{2\}$  in the system requires the packets in this queue. This queue can be reached by packets from  $Q_0$  if the former  $Q_0$  packets are received at receivers  $\{1, 3\}$ , from  $Q_1$  if the former  $Q_1$  packets are received at receiver 1, or from  $Q_2$  if the former  $Q_2$  packets are received at receiver 3.

$Q_6$

A queue in the system that indicates that receiver  $\{3\}$  in the system requires the packets in this queue. This queue can be reached by packets from  $Q_0$  if the former  $Q_0$  packets are received at receivers  $\{1, 2\}$ , from  $Q_2$  if the former  $Q_2$  packets are received at receiver 2, or from  $Q_3$  if the former  $Q_3$  packets are received at receiver 1.

For each of these queues, the packets can leave the system entirely if they are transmitted and received at the received indices associated with their queues. The probability of transition between sub-queues at any given transmission period is dependent upon how often packets are selected from the sub-queues and the channel state parameters.

Network coding is used to combine packets utilizing the fact that some of the combined packets are known at the receiver and the relevant packets can be decoded from the combined packets. For the network coding, the index sets must be mutually exclusive (i.e. no intersection between any two index sets) to ensure the received packet is instantly decodable at all intended users. Also, the union of the index sets should be a full user set so that every network coded packet provides new information to as many users as possible.

### Assigning Network Coding based on Queue-based Strategy

The listed network codes are formed because the combined receiver indices of the queues sets contain receivers 1-3 (reference figure 4 for which receivers are needed for each queue). Scheduling methods must be applied for choosing network codes to combine packets with the correct probabilities  $P_i$  so that the queues will not grow boundlessly over time.

Random Coding	Probability Of Use	Responsible Queues
Network Coding # 0	P0	$Q_0$
Network Coding # 1	P1	$Q_4, Q_5, Q_6$
Network Coding # 2	P2	$Q_1, Q_6$
Network Coding # 3	P3	$Q_2, Q_4$
Network Coding # 4	P4	$Q_3, Q_5$

Figure 6: System Network Codes

# System Stabilizing

## Problem of Queue-based strategy

If the system does not devote a high enough percentage of the time transmitting from a given queue, packets can accumulate and backlog in the queue. If the number of backlogged packets grows too large, the queues overflow and packets are dropped.

Formulate the strategy for selecting network codes for transmission (determined with a focus on which queues make up the network code) into a linear programming problem by solving which we gain the maximum input rate (optimal stable input rate) while the transmitter sub-queues are guaranteed to be stable. Under the queue stability constraint, the input rate equals the services rate, which is also the network throughput. Therefore, the maximum stable input rate provides the maximum achievable network throughput when bounded queuing system is considered.

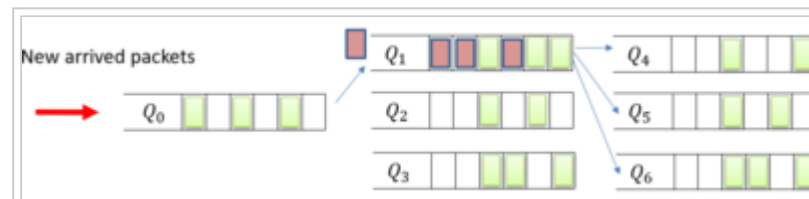


Figure 7: Dropping Packets from Finitely-sized Queues

## Queue Stability Analysis

$$\text{Number of packets in queues} = U_i(t + 1) = \max[U_i(t) - \mu_i(t), 0] + A_i(t)$$

$U_i(t)$  : number of packets inside the queue  $i$  at time  $t$

$\mu_i(t)$  : number of packets left the queue  $i$  at time  $t$

$A_i(t)$  : number of packets arrived at queue  $i$  at time  $t$

## Network Stability

A network is strongly stable if all individual queues of the network are strongly stable.

## Linear Program for selecting optimal transmission probabilities

```

maximize:
    λ

subject to:
    Q0 : λ - P0 · (e1 · e2 · γ3 + e1 · γ2 · γ3 + γ1 · γ2 · γ3 + γ1 · e2 · γ3 + e2 · e3 · γ1 + γ1 · γ2 · e3 + e1 · e3 · γ2) ≤ 0;
    Q1 : P0 · e1 · e2 · γ3 - P2 · (γ1 · γ2 + e1 · γ2 + γ1 · e2) ≤ 0;
    Q2 : P0 · e2 · e3 · γ1 - P3 · (e2 · γ3 + γ2 · γ3 + e3 · γ2) ≤ 0;
    Q3 : P0 · e1 · e3 · γ2 - P4 · (γ1 · γ3 + e1 · e3 + γ1 · e3) ≤ 0;
    Q4 : P2 · e1 · γ2 + P0 · e1 · γ2 · γ3 + P4 · e1 · γ3 - (P1 + P3) · γ1 ≤ 0;
    Q5 : P2 · e2 · γ1 + P0 · e2 · γ1 · γ3 + P3 · e2 · γ3 - (P1 + P4) · γ2 ≤ 0;
    Q6 : P3 · e3 · γ2 + P0 · e3 · γ1 · γ2 + P4 · e3 · γ1 - (P1 + P2) · γ3 ≤ 0;
    Σ Pi = 1

Where λ = arrival/departure rate, Qi = queue i,
Pi = probability of transmitting using channel code i,
ei = transmission erasure probability for channel between transmitter and receiver i,
γi = 1 - ei = transmission success probability for channel between transmitter and receiver i

```

## Examples: Optimal solutions for the Linear Program

Figure 8 shows optimal solutions for the LP given certain error probabilities. Looking at these intuitively:

- Row 1: With an lower overall error probability, the system will most often choose to transmit from  $Q_0$ ;
- Row 2: Increasing the error probability for all channels, there will be a higher probability of retransmission from queues with one remaining receiver;
- Row 3: Since the channel between the transmitter and receiver 3 is weaker,  $Q_6$  will grow faster than other queues so  $P_2$  ( $Q_1$  and  $Q_6$ ) most be selected more frequently;
- Row 4: similar to Row 2 example, again, since the overall error increases. Increase  $P_3$  because that is the network code that contains the set with receivers 2 and 3.

$\epsilon_1$	$\epsilon_2$	$\epsilon_3$	$\lambda_{max}$	$P_0$	$P_1$	$P_2$	$P_3$	$P_4$
0.1	0.1	0.1	0.9	0.9009009	0.7452907E-01	0.8190008E-02	0.8190008E-02	0.8190008E-02
0.2	0.2	0.2	0.8	0.8064516	0.1129032	0.2688172E-01	0.2688172E-01	0.2688172E-01
0.1	0.1	0.2	0.8	0.8016032	0.6146244E-01	0.1074877	0.1472332E-01	0.1472332E-01
0.1	0.2	0.2	0.8	0.8032129	0.1404393	0.1311368E-01	0.3012048E-01	0.1311368E-01

Figure 8: Solutions for different error probabilities

**Extension to the Linear Program** This LP to find the optimal solution, while it works in this case, grows exponentially in complexity if receivers are added to the system. The number of queues is  $2^N - 1$  and the number of possible codes is a Bell number.

# Link to GitHub page

Our AMPL code and presentation Pdf file are available at <https://github.com/SajjadNassirpour/Stable-Queue-based-network->

## References

N. Moghadam and H. Li, “A new wireless multicast queuing design using network coding and data-flow model,” IEEE Commun. Lett., vol. 20, no. 8, pp. 1603–1606, Aug. 2016 [[1] (<https://ieeexplore.ieee.org/iel7/4234/5534602/07469846.pdf>)]  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Queue-Based\\_Strategy\\_to\\_Achieve\\_Maximum\\_Stable\\_Rate\\_in\\_Multi-user\\_Network&oldid=2505](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Queue-Based_Strategy_to_Achieve_Maximum_Stable_Rate_in_Multi-user_Network&oldid=2505)"

- 
- This page was last modified on 5 December 2019, at 12:20.
  - This page has been accessed 3,830 times.



# Quinsen Joel

From CU Denver Optimization Student Wiki

I am a MEng in GIS and Geomatics student in the Civil Engineering Department at the University of Colorado Denver. I completed my Bachelor of Arts in Geography at the University of Colorado Denver where I was first introduced to GIS and geospatial data. I am currently working as an intern under the Geospatial Data Science group at the National Renewable Energy Laboratory (NREL). I'm interested in learning how mathematical and computational techniques can connect with Geography.

I am currently taking Network Flows with Steffen Borgwardt and contributed the following project: Assignment of Reservoirs for Pumped Hydro Storage Systems

## Link to Github Page

Assignment of Pumped Hydro Storage Reservoirs Presentation (<https://github.com/qjoel6398/Assignment-of-Pumped-Hydro-Storage-Reservoirs%7C>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Quinsen\\_Joel&oldid=2845](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Quinsen_Joel&oldid=2845)"

Category: Contributors

- 
- This page was last modified on 5 May 2020, at 21:23.
  - This page has been accessed 1,050 times.

# Rachel Snyder

From CU Denver Optimization Student Wiki

## About Me

Hi! I'm Rachel Snyder, and I am a first year PhD student at the University of Denver Colorado. I am interested in discrete mathematics, particularly enumerative combinatorics and graph theory.

## Projects

I am currently working on the Catalan Numbers.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Rachel\\_Snyder&oldid=4880](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Rachel_Snyder&oldid=4880)"

Category: Contributors

- 
- This page was last modified on 16 April 2025, at 14:47.
  - This page has been accessed 20 times.

# Rebecca Robinson

From CU Denver Optimization Student Wiki

## Contents

- 1 About Me
  - 1.1 Education
  - 1.2 Hobbies
- 2 My Contributions to the Wiki



## About Me

My name is Rebecca Robinson and I am in my third year in the Applied Mathematics Ph.D program at the University of Colorado - Denver. I specialize in graph theory, and I am currently working on a Master's project with Dr. Stephen Hartke involving really cool graph coloring stuff.

## Education

I obtained my B.S. in Mathematics (Concentration in Abstract Mathematics) with minors in Biology and Chemistry from the University of Michigan - Flint in 2017. My capstone project involved using Markov chains to model the viral-like spread of internet memes. I also completed an Honors thesis about chromatic polynomials.

## Hobbies

Outside of mathematics, I enjoy playing board games with friends, taking my four-year-old black lab River out to breweries, and checking out the Denver area craft beer scene. I am also known for having a different hair color almost every month.

## My Contributions to the Wiki

In Fall of 2019, I worked on An Integer Linear Programming Approach to Graph Coloring with Megan Duff.

In Spring of 2020, I worked on Sollin's Algorithm for Minimum Spanning Trees.

In Spring of 2021, I worked with Drew Horton on combating inequality in access to fresh foods and produce in our project, Hungry for Equality: Fighting Food Deserts.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Rebecca\\_Robinson&oldid=3337](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Rebecca_Robinson&oldid=3337)"

Category: Contributors

- This page was last modified on 5 May 2021, at 18:46.
- This page has been accessed 5,021 times.

# Reducing Crimes with a Cleaner Solution

From CU Denver Optimization Student Wiki

## Introduction:

The Traveling Salesman Problem(TSP) is one of the most famous integer programming algorithm. The concept is simple, find the quickest/cheapest/shortest path to visit a certain amount of locations. The solutions become far from simple, especially when the amount of locations increases to larger numbers. We plan to find that solution for a very specific case. The city of Denver has a large amount of parks in which we could apply TSP algorithms to. Our goal is to set up routes for anyone who wants to, or has to, clean the parks of Denver.

## Abstract:

Parks have always been a place for communities to come together. In fact, several studies have shown that a clean park can help reduce the crime of the area in and around the park. We will use this knowledge to develop plans to clean and maintain parks more efficiently. The solution involves solving a famous integer program called “The Traveling Salesmen Problem.” We plan to implement Christofide’s algorithm to find a route for city workers, volunteers, or any person who cares about the environment to visit and clean our parks in a quick, efficient manner. The data sets that we will use about park size and location will be taken from the Data to Policy website. Distances will be calculated using the taxicab geometry through online software. Results will be presented at the semiannual Data to Policy conference.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Reducing\\_Crimes\\_with\\_a\\_Cleaner\\_Solution&oldid=1952](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Reducing_Crimes_with_a_Cleaner_Solution&oldid=1952)"

- 
- This page was last modified on 21 March 2019, at 12:12.
  - This page has been accessed 855 times.

# Return To School Success In Times of COVID

From CU Denver Optimization Student Wiki

This project is by Alyssa Newman, and is for the Integer Programming course in Spring of 2021.

## Abstract

A big issue currently facing not just Colorado, but the whole world is how to send children back to face-to-face learning after a year or remote learning. Schools are looking at how to make improvements to the school over this summer to make next years' experience safe and as successful as possible. In this project I am developing an optimization model that will aid schools in making difficult decisions on what policies to implement and changes to make in their school. This model will use integer programming techniques to make suggestions on policies and changes. This model will take into account many things, including the budget restrictions the school has, and the short term and long-term benefits to students learning and experience. The school administrators using the this tool will be able to specify which policies they are considering and also put in information like their budget to get suggestions tailored to them.

## GitHub links

All information, including code, slides, and 5 minute Data to Policy video can be found at <https://github.com/ANewman94/Return-To-School-Success-In-Times-of-COVID>

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Return\\_To\\_School\\_Success\\_In\\_Times\\_of\\_COVID&oldid=3308](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Return_To_School_Success_In_Times_of_COVID&oldid=3308)"

- 
- This page was last modified on 4 May 2021, at 11:39.
  - This page has been accessed 955 times.

# Sajjad Nassirpour

From CU Denver Optimization Student Wiki

This is Sajjad Nassirpour. I am a first year PhD student at electrical engineering department at CU Denver. My major field is Information Theory in wireless communication systems. There are a lot of optimization problems in Information Theory which are considered as Linear programming problems. In fall 2019, I decide to work with John McFarlane on a problem in wireless communication involving the queue-based strategy to achieve maximum stable throughput in multi-user topology, the link is Queue-Based Strategy to Achieve Maximum Stable Rate in Multi-user Network

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sajjad\\_Nassirpour&oldid=2213](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sajjad_Nassirpour&oldid=2213)"

Category: Contributors

- 
- This page was last modified on 15 November 2019, at 11:57.
  - This page has been accessed 912 times.

# Sam's Hauling and Vehicle Routing Problems

From CU Denver Optimization Student Wiki

Sam's Hauling is a mini dumpster rental service for the greater Denver area. As described at [www.samshauling.com](http://www.samshauling.com) (<http://www.samshauling.com>), the lease dumpsters for private use such as homeowners, remodelers, contractors, and roofers. When the dumpsters are filled, Sam's Hauling also unloads the dumpsters at a landfill.

Every day, Sam's Hauling has to schedule times to pick up or drop off these dumpsters for all the requests people have made the next day. When done by hand, this takes a lot of time. Thus, software to efficiently schedule their routes could improve their efficiency.

In this project, we experiment with a particular simplified formulation of their problem and test the problem's tractability with the commercial optimization software Cplex. The goals of this project are to demonstrate how mixed integer programs arise in practice and to gain experience solving integer programs with software libraries. The code I used can be found at Sams code.

## Contents

- 1 Problem Context
  - 1.1 Traveling Salesman Problem
  - 1.2 Vehicle Routing Problem
- 2 Tractability
- 3 Model Assumptions
  - 3.1 Description
  - 3.2 Simplifications
- 4 Formulation
  - 4.1 Previous Heuristic
  - 4.2 Constraints
  - 4.3 Inventory Constraints
  - 4.4 Objective
- 5 Computational Results
  - 5.1 Implementation Details
    - 5.1.1 Problem Generation
    - 5.1.2 Source Code
    - 5.1.3 Local Search
    - 5.1.4 CPLEX API
    - 5.1.5 Example output
  - 5.2 Runtimes



## Problem Context

There are various well-known problems that can be used to model most aspects of this problem including the Vehicle Routing Problem (VRP) and the Pickup up Delivery Problem. In this section, we will discuss the background and tractability of the VRP and related problems.

### Traveling Salesman Problem

We will first note that the VRP is a generalization of the Traveling Salesman Problem (TSP), first posed in the 1800s by W. R. Hamilton. We construct a directed graph  $G = (V, E)$  where nodes in  $V$  are locations (or stops) that must be visited and the edges in  $E$  are routes between the edges. If we assign a cost  $c_e$  to edge  $e$  for each edge in  $E$  and use  $\delta^\pm(S)$  to denote the set of edges directed out of or into any node within  $S \subseteq V$ , then the TSP can be formulated as

$$\begin{aligned} \min_{x_e, e \in E} \quad & \sum_{e \in E} c_e x_e \\ \sum_{e \in \delta^+(\{i\})} x_e &= 1 \quad \forall i \in V \\ \sum_{e \in \delta^-(\{i\})} x_e &= 1 \quad \forall i \in V \\ \sum_{e \in \delta^+(S)} x_e &\geq 1 \quad \forall \emptyset \subsetneq S \subset V \\ x_e &\in \{0, 1\} \quad \forall e \in E \end{aligned}$$

The decision variables  $x_e$  indicate if edge  $e$  is used in the path. The first constraints ensure that each node is visited once, while the second are called "sub-tour elimination" constraints and ensure that there are no cycles that are not connected to the rest of the cycle. Note that this last type of constraint implies an exponential number of constraints: one for each subset of  $V$ . With only these variables, the TSP cannot be formulated with fewer constraints. However, there is another formulation due to Miller, Tucker, and Zemlin using only quadratic number of constraints by introducing auxiliary variables  $u_i \forall i \in V$  and instead requiring

$$u_i - u_j + 1 \leq n(1 - x_{ij}) \quad \forall (i, j) \in E$$

instead of the sub-tour elimination constraints.

### Vehicle Routing Problem

The vehicle routing problem is a generalization of the traveling salesman problem in that it allows several salesmen to visit each node of the graph. Each salesman is usually required to start and end at a single node called the "depot" which is usually chosen to be node 0 for convenience. Within a VRP given data stating a number of vehicles or drivers and several locations that need to be visited by the drivers. We then ask to minimize the amount of time required to visit these locations, although different objectives

Let  $T$  denote the number of drivers that are available to service the stops. If a function  $r : 2^V \rightarrow Z_+$  representing the minimum number of vehicles required to visit any subset of the vertices is known, then Dantzig, Fulkerson, and Johnson showed the VRP can be formulated as

$$\begin{aligned}
 & \min_{x_e, e \in E} \sum_{e \in E} c_e x_e \\
 & \sum_{e \in \delta^+(\{i\})} x_e = 1 \quad \forall i \in V \setminus \{0\} \\
 & \sum_{e \in \delta^-(\{i\})} x_e = 1 \quad \forall i \in V \setminus \{0\} \\
 & \sum_{e \in \delta^+(\{0\})} x_e = T \\
 & \sum_{e \in \delta^-(\{0\})} x_e = T \\
 & \sum_{e \in \delta^+(S)} x_e \geq r(S) \quad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset \\
 & x_e \in \{0, 1\} \quad \forall e \in E
 \end{aligned}$$

This time, the first two constraints require that each node aside from the depot must be visited once, while constraints the next two require that exactly  $T$  must leave and enter the depot. Finally, the second to last constraints replace the sub-tour elimination constraints.

There are several extensions to the VRP that may be relevant here. The first is the Vehicle Routing problem with pick up and delivery. In this problem, rather than finding the minimum distance to visit all nodes, we are asked to move goods from pick up locations to drop off locations. Another is the Vehicle Routing Problem with Time Windows, in which nodes are assigned a range of values for which the vehicles can visit each node. This is the case for Sam's Hauling. The last that we briefly mention is the capacitated Vehicle Routing Problem in which vehicles of constraints on how many goods they can carry. This may have bearing on our problem if we modeled all parts of our problem.

## Tractability

Traveling salesman problems and Vehicle routing problems are notoriously difficult, NP-hard problems. I found that Cplex begins to take several minutes to solve traveling salesman problems as problem size reaches around 20 nodes. One of the well-known libraries for solving TSPs is TSPLIB which has solved an 85,900 city TSP, although optimality for solving a TSP of this size cannot be expected in general.

A common technique for solving these problems is to use heuristics including ant colony optimization, tabu search, genetic algorithms, simulated annealing, scatter search and particle search among many others. Many of these techniques can use a distance function between known solutions, or a local search to improve the quality of solutions found. One way of performing local searches is to define operations for moving between solutions and a neighborhood about a given solution that includes all other solutions that are can be reached from the current solution within a specified number of operations.

One common choice of operations to move from solution to solution that has proven to be useful for the TSP is  $n$ -opts. An  $n$ -opt is constructed by breaking a cycle at  $n$  edges of a solution and recombining these. These can be used within more complicated heuristics or even branch and bound to provide warm starts and feasible solutions with strong global upper bounds on the solution. For example, a 2-opt is formed by removing two edges and flipping the order of the middle path. If one part is the nodes ..., A, B, C, D, ... and the other part is ..., E, F, G, H, ... these can then be exchanged to ..., A, B, F, E, ... and ..., C, D, G, H, ...

To the right, there is an example of one 2-opt. The cycle 9, 10, 11, 12, 13, 14, is replaced with 9, 12, 11, 10, 13, 14 to find a better cycle. Notice that all cycles are within an  $n$ -opt of eachother where  $n = |V|$ .

## Model Assumptions

### Description

There are several details of Sam's Hauling that would be required to provide good schedules for Sam's Hauling. To start considering these, we first describe the logistics of Sam's Hauling.

Sam's Hauling leases four different types of dumpsters: six, nine, twelve, and sixteen-yard dumpsters. Customers can rent any of these dumpsters and will give Sam's Hauling a request to either pick up, deliver, or replace a dumpster of the given size. In this case, a replace can be thought of as both a delivery followed by a pick up. Additionally, these requests may have associated time windows when the customer needs the request to be serviced.

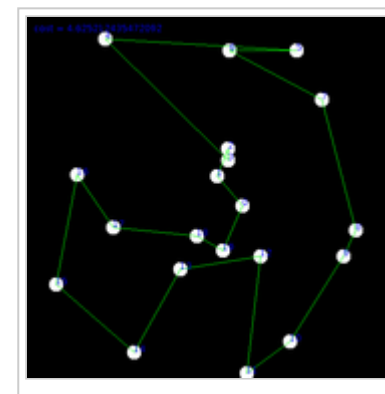
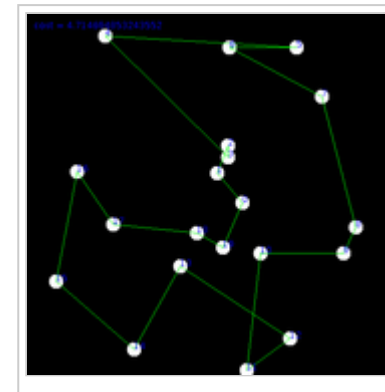
The typical number of requests per day is on the order of **30** to **40**. Sam's Hauling has eight drivers to service these requests with ten trucks of various sizes. Only the largest truck type can service dumpster requests of the largest size, while some locations can only be serviced by the smallest truck as it is more maneuverable.

The dumpsters retrieved from the pick ups must be dumped at one of four landfills throughout Denver, which can have varying wait times or unloading fees. At each landfill, Sam's Hauling owns space to store their small supply of unused dumpsters after pick ups that are ready to be delivered to future customers.

The number of dumpsters at each landfill is called the landfill's inventory, so inventory constraints require that the number of dumpsters at each landfill must remain positive. (We can pick up a dumpster from a landfill that has no dumpsters.) The inventory constraints are of particular concern as we have developed a heuristic that can find good solutions to the routing problem, but which required many more available dumpsters at each depot. This heuristic did not perform well when including these inventory constraints. Note that the schedule may ask that some drivers make trips between dumpsters with only the intention being to take move a dumpster from one landfill to another so that a subsequent driver can pick it up.

A good model may also include some stochastic components. For example, upon servicing some pick ups the dumpsters may require additional cleaning time if, for example, it has been graffitied. Also, drive times between destination and wait times at landfills may vary randomly across the day.

Finally, the last modeling complication we mention is that of nesting dumpsters. It is sometimes possible to nest empty dumpsters inside of each other to allow multiple deliveries after visiting a given landfill.



## Simplifications

Although we have a model that describes many of these aspects, we will only consider the simplest of models as input for Cplex. In particular, we will make all of the following assumptions:

- We only consider a single dumpster size

Considering more dumpster sizes is not much harder, but complicates the model unnecessarily by adding several indices.

- We assume only one type of truck capable of servicing all requests

This also allows a one-to-one mapping between drivers and trucks.

- We assume no time windows
- The time between stops is constant (we used a google API to determine distances between locations in Denver)

## Formulation

### Previous Heuristic

Because a heuristic I used in the past influenced the formulation provided here, we will very briefly describe it. Because each driver must visit one of the landfills no later than his third stop from a previous landfill, we found it convenient to break schedules into ``legs." Each leg begins and ends in a landfill and can optionally have a delivery followed by an optional pick up. The heuristic had three phases:

- Assign deliveries to pick ups (This is a matching problem that is easily solved.)
- Form ``legs" between landfills and deliveries and between pick ups and landfills
- Assign each leg to a particular driver's route.

Then we would consider slightly less than optimal assignments of legs or assignments of deliveries to pick ups to see if shorter routes could be created. The assignment phase did not assign any deliveries to pick ups where the cost of doing so was greater than servicing the delivery alone. Lastly, pick ups were assigned to landfills based on the inventory at that landfill.

# Constraints

The models proposed earlier in this section use decision variables for each \emph{edge} to indicate if they are taken. However, this makes several other aspects more difficult to model. Instead, we use a formulation that assigns requests to positions in a driver's schedule. This may not be the most efficient formulation as it requires many more variables, but it is more similar in structure to the heuristic that we developed earlier. We had hopes of warm starting Cplex using this heuristic so that we went with this formulation.

With eight drivers and thirty requests per day, most drivers don't visit more than eight stops. Therefore, we started by breaking each route into a sequence of  $R$  legs. Choosing a value of  $R$  that is too small will make the program not feasible if there are not enough legs to service all requests. However, there are values of  $R$  too small may force deliveries to be grouped with pick ups that should not be. Therefore care must be taken to chose  $R$  large enough to allow the optimal solution, but small enough that the problem is still tractable. For each driver, within each leg, a binary variable was created representing which path from landfill to delivery to pick up to delivery was chosen. That is all possible legs are enumerated first: if there are  $P$  pick ups,  $D$  deliveries, and  $L$  landfills there are  $M = L \cdot (D + 1) \cdot (P + 1) \cdot L$  different possible legs. (The plus ones indicate that a deliver or pick up is not visited.) We then create a boolean variable  $x_{d,r,l}$  to determine whether driver  $d \in \{1, 2, \dots, T\}$  uses leg  $l \in \{1, 2, M\}$  as his  $r \in \{1, 2, \dots, R\}$  th leg.

We can use data indicators  $p_{li} \forall 1 \leq l \leq M, 1 \leq i \leq P$  to be 1 if leg  $l$  services pick up  $i$ , and zero otherwise. We create data indicators  $d_{li} \forall 1 \leq l \leq M, 1 \leq i \leq D$  to indicate if leg  $l$  services delivery  $i$ . Lastly, we create data indicators  $s_{li} \forall 1 \leq l \leq M, 1 \leq i \leq L$  and  $f_{li} \forall 1 \leq l \leq M, 1 \leq i \leq L$  to indicate if leg  $l$  starts at or finishes at landfill  $i$ .

The last ingredient is the time. We can assign a time  $t_{dr} \geq 0 \forall 1 \leq d \leq T, 1 \leq r \leq R$  completed to each leg in a driver's route. If the time required to take leg  $l$  is given by  $c_l$ , then we can create an auxiliary variable  $u \geq 0 \forall 1 \leq d \leq T$  to simulate the maximum time take by all any driver.

Then we can finally write the constraints as follows, the set of all  $u, t_{dr}, x_{dlr}$  that satisfy

$$\begin{aligned}
\sum_{d=1}^D \sum_{l=1}^M \sum_{r=1}^R x_{dlr} p_{li} &= 1 & \forall 1 \leq i \leq P \\
\sum_{d=1}^D \sum_{l=1}^M \sum_{r=1}^R x_{dlr} d_{li} &= 1 & \forall 1 \leq i \leq D \\
\sum_{l=1}^M x_{dl,r+1} l_{li} &\leq \sum_{l=1}^M x_{dlr} s_{li} & \forall 1 \leq d \leq D, 1 \leq r \leq R-1, 1 \leq i \leq L \\
x_{dl0} &= 0 & \forall 1 \leq d \leq D, 2 \leq i \leq L \\
x_{dl,R-1} &= 0 & \forall 1 \leq d \leq D, 2 \leq i \leq L \\
\sum_{l=1}^M x_{dlr} &= 1 & \forall 1 \leq d \leq T, 1 \leq r \leq R \\
t_{dr} &\geq t_{d,r-1} + \sum_{l=1}^M x_{dlr} c_l & \forall 1 \leq d \leq D, 2 \leq r \leq R \\
u &\geq t_{d,R-1} & \forall 1 \leq d \leq T \\
t_{dr} &\geq 0 & \forall 1 \leq d \leq D, 1 \leq r \leq R \\
u &\geq 0 \\
x_{dlr} &\in \{0, 1\} & \forall 1 \leq d \leq T, 1 \leq l \leq M, 1 \leq r \leq R
\end{aligned}$$

will be denoted as  $F$ .

The first two constraints require that each pick up and deliver is visited exactly once. Next, we dictate that a driver cannot leave a landfill he did not enter in the last leg. Then we ensure that each driver must start and end at the first depot. Lastly, before considering time constraints, we ensure a driver can only do one thing during each leg.

The first time constraint says that the completion time of the  $d$ th driver on his  $r$ th leg must be greater than his time at the  $r - 1$  stop plus the time taken on the  $r$ th leg, while the third to last constraint requires that the total time  $u$  must be bigger than any time taken by any driver.

I am not proud of the constraints and variables used, but this is a somewhat simple way to formulate the problem. It would likely be better to formulate the problem with edges between each stop. Another major problem with this formulation is that it has symmetry issues. Note that if a driver stays at a landfill for leg  $r$ , then that driver could have stayed at another landfill for any other leg. Not only this, but all drivers are indistinguishable.

Many of these constraints are what are known as specially ordered sets or *SOS* - 1 constraints, which are typically nicer, as the depth of a branch and bound tree only needs to be logarithmic in the number of variables. These kinds of constraints require only one of a set of binary variables to be turned on. This may let the optimization library make better branches. One example of how this kind of information could be useful is in the following: if  $x_i, 1 \leq i \leq 10$  are binary variables such that

$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} = 1$ , a naive approach would be to branch on  $x_1 = 0$  or  $x_1 = 1$ , and continue in the fashion. However, this would lead to a complete enumeration of all possible solutions, it may be much better to branch on  $x_1 + x_2 + x_3 + x_4 + x_5 = 1$  or  $x_1 + x_2 + x_3 + x_4 + x_5 = 0$  to limit the number of branches. This indication that  $x_i$  are in an  $SOS - 1$  constraint may allow the library to make smarter branches.

## Inventory Constraints

The original plan was to also consider inventory constraints. These constraints are very problematic because they apply to all drivers based on the time. One approach would be to first solve the problem without inventory constraints. Given a solution, we could simply check what times the inventories become negative and add constraints to ensure that all inventories are positive at that time. For example, if it is a solution to the previous problem becomes infeasible at times  $v_k$  for some set times  $v_k, 1 \leq k \leq K$ , we could try to rule these solutions out in the following manner. Suppose that the initial inventories are given by  $I_i^s \forall 1 \leq i \leq L$ . We introduce nuisance variables  $b_{drk} \forall 1 \leq d \leq T, 1 \leq r \leq R, 1 \leq k \leq K$  to signify if the  $r$ th leg of driver  $d$  is completed before time  $v_k$ . We can ensure this by stipulating that

$$t_{dr} \leq v_k + b_{drk}B$$

where  $B$  is an arbitrarily large number. Then we create inventory variables  $I_{ki}$  for each time  $k$  at each landfill  $i$ . These are integers but do not need to be explicitly declared as such as they will be forced to be integral within the program. We can create data  $m_{li} \forall 1 \leq l \leq M, 1 \leq i \leq L$  representing the change in inventory at landfill  $i$  from performing leg  $l$ . For example, if leg  $l$  picks up a dumpster from landfill  $i$  and delivers it before pick up a dumpster on the way to landfill  $j$ , then  $m_{li} = -1$  and  $m_{lj} = +1$ . Then inventories can be tracked with the constraint

$$I_{ki} = I_i^s + \sum_{d=1}^D \sum_{l=1}^M \sum_{r=1}^R x_{dlr} b_{drk} m_{li} \quad \forall 1 \leq k \leq K, 1 \leq i \leq L$$

The last piece is to ensure that the inventories are never negative:

$$I_{ki} \geq 0 \quad \forall 1 \leq k \leq K, 1 \leq i \leq L.$$

However, there is a major problem with this approach: it introduces quadratic terms between  $x_{dlr} b_{drk}$ . There are ways to enforce linearity because these are both binary variables, but this formulation is already out of hand without including inventory constraints. Another caveat is that this does not make a distinction between a dumpster being taken at the \emph{beginning} of the leg and a new dumpster being delivered at the \emph{end} of the leg.

The inventory constraints make most otherwise feasible solutions infeasible, as the  $I_i^s$  are usually small. Because these constraints cut off much of the feasible region, these could be nice to include within a search algorithm to narrow the number of solutions to be tested. However, adding them to a linear program formulation makes the problem much more complicated and does not imply a speed up.

## Objective

There are many different considerations when choosing the objective. One thing that might be nice to consider is the end of day inventories: if we leave a landfill with no dumpsters, then scheduling the next day may prove to be difficult. Also, in practice, we really wish to maximize the number of stops we are capable of visiting, subject to finishing the stops in one day.

Initially, we chose to minimize the time taken to service all requests, that is  $\min\{u : (u, t_{dr}, x_{dlr}) \in F\}$ . However, this has some drawbacks. First of all, it introduces much more symmetry: the objective does not change for several different solutions where the driver who takes the most time does not change his route. Also, the cost in time is simply the sum of all times:  $\sum_{d=1}^T t_{d,R-1}$  so that if we could make this smaller without affecting the longest time spent by any driver. However, simply minimizing this assigns nearly all stops to a single driver. Therefore, we introduced a parameter  $\lambda$  and minized a weighted sum of both of these expressions:  $\min\{\lambda T u + \sum_{d=1}^T t_{d,R-1} : (u, t_{dr}, x_{dlr}) \in F\}$ . In our code, we assigned  $\lambda = 2$ .

## Computational Results

### Implementation Details

#### Problem Generation

To learn how to use Cplex, we created randomly generated problem sets, with locations being chosen with a uniform distance over  $[0, 1]^2$  distances given by the euclidean norm. We considered values of  $P$  between 3 and 15, values of  $D$  between 3 and 15, values of  $L$  between 2 and 4, values of  $R$  between 5 and 10.

#### Source Code

The source code I used has most likely changed since I edited this, but a snapshot containing most of the work is contained at Sams code.

#### Local Search

Initially, I used the object  $\min\{u : (u, t_{dr}, x_{dlr}) \in F\}$ , and I found several solutions that were visibly less than optimal by changing only a few vertices. However, upon further examination, I realized that these were only improvements for drivers whose time was not the maximum time. Upon changing the objective to  $\min\{\lambda T u + \sum_{d=1}^T t_{d,R-1} : (u, t_{dr}, x_{dlr}) \in F\}$ , I was no longer able to find simple solutions to the solutions found by Cplex.

My attempts are found in LocalSearch.java. The local search operation was to change any two sequential landfills within a route while swapping any two delivers or two swapping any two pick ups. Unfortunately, swapping any two delivers or swapping any two pick ups at the same time was too expensive. I had hoped that by starting a random search, and sequentially improving the solution by considering all these changes would provide a good warm start to the cplex optimizer. However, the cplex solver was good

Typesetting math: 100%

a few seconds, I was not able to improve the solutions Cplex had.



While solving the MIP, cplex maintains a pool of feasible solutions currently found. Although this may not be the proper approach, I used the populate method to give breaks while solving for me to access these. However, this did not reduce the optimality gap as quickly as simply calling the solve method.

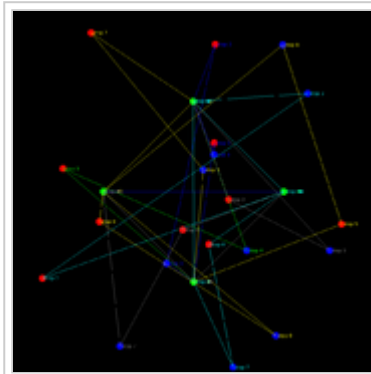
Example output

Shown to the right, is example output from the cplex solver. Red circles correspond to deliveries, blue circles are pick ups, and green circles are landfills. The lines between these circles are the routes chosen by cplex, and the landfill all the way to the right is the beginning depot.

Runtimes

Cplex was not able to solve this formulation for realistic problem sizes. I found that with  $L = 2, P = 5, D = 5, R = 5$ , cplex was able to solve to optimality within 65 seconds. If I let cplex run for one minute, the optimality gaps found by cplex were:

$L$	$P$	$D$	$R$	$T$	objective	number of binaries	number of rows	Optimality Gap \%
2	5	5	5	5	25.15	2880	90	0.01
2	10	5	5	5	25.1002	5280	95	4.94
2	5	10	5	5	24.3465	5280	95	5.15
4	10	10	5	5	26.4809	33880	140	19.77
4	15	15	8	8	43.8683	213009	342	27.19



Example output of the cplex solver

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sam%27s\\_Hauling\\_and\\_Vehicle\\_Routing\\_Problems&oldid=544](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sam%27s_Hauling_and_Vehicle_Routing_Problems&oldid=544)"

- This page was last modified on 3 May 2017, at 12:56.
- This page has been accessed 9,543 times.

# Sam's Hauling and Vehicle Routing Problems

From CU Denver Optimization Student Wiki  
(Redirected from Sam's Hauling and Vehicle Routing)

Sam's Hauling is a mini dumpster rental service for the greater Denver area. As described at [www.samshauling.com](http://www.samshauling.com) (<http://www.samshauling.com>), the lease dumpsters for private use suck as homeowners, remodelers, contractors, and roofers. When the dumpsters are filled, Sam's Hauling also unloads the dumpsters at a landfill.

Every day, Sam's Hauling has to schedule times to pick up or drop off these dumpsters for all the requests people have made the next day. When done by hand, this takes a lot of time. Thus, software to efficiently schedule their routes could improve their efficiency.

In this project, we experiment with a particular simplified formulation of their problem and test the problem's tractability with the commercial optimization software Cplex. The goals of this project are to demonstrate how mixed integer programs arise in practice and to gain experience solving integer programs with software libraries. The code I used can be found at Sams code.

## Contents

- 1 Problem Context
  - 1.1 Traveling Salesman Problem
  - 1.2 Vehicle Routing Problem
- 2 Tractability
- 3 Model Assumptions
  - 3.1 Description
  - 3.2 Simplifications
- 4 Formulation
  - 4.1 Previous Heuristic
  - 4.2 Constraints
  - 4.3 Inventory Constraints
  - 4.4 Objective
- 5 Computational Results
  - 5.1 Implementation Details
    - 5.1.1 Problem Generation
    - 5.1.2 Source Code
    - 5.1.3 Local Search
    - 5.1.4 CPLEX API
    - 5.1.5 Example output
  - 5.2 Run times

## Problem Context

There are various well-known problems that can be used to model most aspects of this problem including the Vehicle Routing Problem (VRP) and the Pickup up Delivery Problem. In this section, we will discuss the background and tractability of the VRP and related problems.

## Traveling Salesman Problem

We will first note that the VRP is a generalization of the Traveling Salesman Problem (TSP), first posed in the 1800s by W. R. Hamilton. We construct a directed graph  $G = (V, E)$  where nodes in  $V$  are locations (or stops) that must be visited and the edges in  $E$  are routes between the edges. If we assign a cost  $c_e$  to edge  $e$  for each edge in  $E$  and use  $\delta^\pm(S)$  to denote the set of edges directed out of or into any node within  $S \subseteq V$ , then the TSP can be formulated as

$$\begin{aligned} \min_{x_e, e \in E} \quad & \sum_{e \in E} c_e x_e \\ \sum_{e \in \delta^+(\{i\})} x_e &= 1 \quad \forall i \in V \\ \sum_{e \in \delta^-(\{i\})} x_e &= 1 \quad \forall i \in V \\ \sum_{e \in \delta^+(S)} x_e &\geq 1 \quad \forall \emptyset \subsetneq S \subset V \\ x_e &\in \{0, 1\} \quad \forall e \in E \end{aligned}$$

The decision variables  $x_e$  indicate if edge  $e$  is used in the path. The first constraints ensure that each node is visited once, while the second are called "sub-tour elimination" constraints and ensure that there are no cycles that are not connected to the rest of the cycle. Note that this last type of constraint implies an exponential number of constraints: one for each subset of  $V$ . With only these variables, the TSP cannot be formulated with fewer constraints. However, there is another formulation due to Miller, Tucker, and Zemlin using only quadratic number of constraints by introducing auxiliary variables  $u_i \forall i \in V$  and instead requiring

$$u_i - u_j + 1 \leq n(1 - x_{ij}) \quad \forall (i, j) \in E$$

instead of the sub-tour elimination constraints.

## Vehicle Routing Problem

The vehicle routing problem is a generalization of the traveling salesman problem in that it allows several salesmen to visit each node of the graph. Each salesman is usually required to start and end at a single node called the "depot" which is usually chosen to be node 0 for convenience. Within a VRP given data stating a number of vehicles or drivers and several locations that need to be visited by the drivers. We then ask to minimize the amount of time required to visit these locations, although different objectives will be discussed as well.

Let  $T$  denote the number of drivers that are available to service the stops. If a function  $r : 2^V \rightarrow Z_+$  representing the minimum number of vehicles required to visit any subset of the vertices is known, then Dantzig, Fulkerson, and Johnson showed the VRP can be formulated as

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \sum_{e \in \delta^+(\{i\})} x_e &= 1 \quad \forall i \in V \setminus \{0\} \\ \sum_{e \in \delta^-(\{i\})} x_e &= 1 \quad \forall i \in V \setminus \{0\} \\ \sum_{e \in \delta^+(\{0\})} x_e &= T \\ \sum_{e \in \delta^-(\{0\})} x_e &= T \\ \sum_{e \in \delta^+(S)} x_e &\geq r(S) \quad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset \\ x_e &\in \{0, 1\} \quad \forall e \in E \end{aligned}$$

This time, the first two constraints require that each node aside from the depot must be visited once, while constraints the next two require that exactly  $T$  must leave and enter the depot. Finally, the second to last constraints replace the sub-tour elimination constraints.

There are several extensions to the VRP that may be relevant here. The first is the Vehicle Routing problem with pick up and delivery. In this problem, rather than finding the minimum distance to visit all nodes, we are asked to move goods from pick up locations to drop off locations. Another is the Vehicle Routing Problem with Time Windows, in which nodes are assigned a range of values for which the vehicles can visit each node. This is the case for Sam's Hauling. The last that we briefly mention is the capacitated Vehicle Routing Problem in which vehicles of constraints on how many goods they can carry. This may have bearing on our problem if we modeled all parts of our problem.

## Tractability

Traveling salesman problems and Vehicle routing problems are notoriously difficult, NP-hard problems. I found that Cplex begins to take several minutes to solve traveling salesman problems as problem size reaches around 20 nodes. One of the well-known libraries for solving TSPs is TSPLIB which has solved an 85,900 city TSP, although

Typesetting math: 100%

a TSP of this size cannot be expected in general.

A common technique for solving these problems is to use heuristics including ant colony optimization, tabu search, genetic algorithms, simulated annealing, scatter search and particle search among many others. Many of these techniques can use a distance function between known solutions, or a local search to improve the quality of solutions found. One way of performing local searches is to define operations for moving between solutions and a neighborhood about a given solution that includes all other solutions that are can be reached from the current solution within a specified number of operations.

One common choice of operations to move from solution to solution that has proven to be useful for the TSP is  $n$ -opts. An  $n$ -opt is constructed by breaking a cycle at  $n$  edges of a solution and recombining these. These can be used within more complicated heuristics or even branch and bound to provide warm starts and feasible solutions with strong global upper bounds on the solution. For example, a **2**-opt is formed by removing two edges and flipping the order of the middle path. If one part is the nodes ..., A, B, C, D, ... and the other part is ..., E, F, G, H, ... these can then be exchanged to ..., A, B, F, E, ... and ..., C, D, G, H, ...

To the right, there is an example of one **2**-opt. The cycle 9, 10, 11, 12, 13, 14, is replaced with 9, 12, 11, 10, 13, 14 to find a better cycle. Notice that all cycles are within an  $n$ -opt of eachother where  $n = |V|$ .

## Model Assumptions

### Description

There are several details of Sam's Hauling that would be required to provide good schedules for Sam's Hauling. To start considering these, we first describe the logistics of Sam's Hauling.

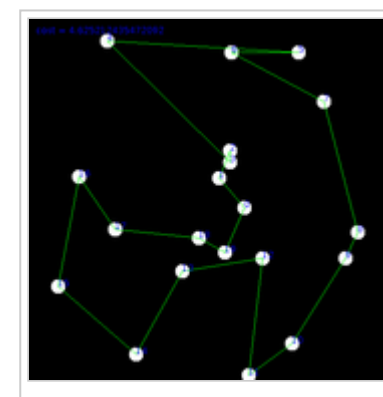
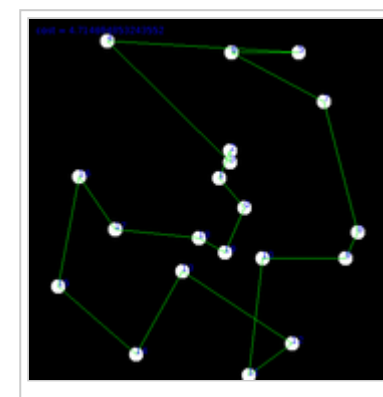
Sam's Hauling leases four different types of dumpsters: six, nine, twelve, and sixteen-yard dumpsters. Customers can rent any of these dumpsters and will give Sam's Hauling a request to either pick up, deliver, or replace a dumpster of the given size. In this case, a replace can be thought of as both a delivery followed by a pick up. Additionally, these requests may have associated time windows when the customer needs the request to be serviced.

The typical number of requests per day is on the order of **30** to **40**. Sam's Hauling has eight drivers to service these requests with ten trucks of various sizes. Only the largest truck type can service dumpster requests of the largest size, while some locations can only be serviced by the smallest truck as it is more maneuverable.

The dumpsters retrieved from the pick ups must be dumped at one of four landfills throughout Denver, which can have varying wait times or unloading fees. At each landfill, Sam's Hauling owns space to store their small supply of unused dumpsters after pick ups that are ready to be delivered to future customers.

The number of dumpsters at each landfill is called the landfill's inventory, so inventory constraints require that the number of dumpsters at each landfill must remain positive. (We can pick up a dumpster from a landfill that has no dumpsters.) The inventory constraints are of particular concern as we have developed a heuristic that can find good solutions to the routing problem, but which required many more available dumpsters at each depot. This heuristic did not perform well when including these inventory constraints. Note that the schedule may ask that some drivers make trips between dumpsters with only the intention being to take move a dumpster from one landfill to another so that a subsequent driver can pick it up.

A good model may also include some stochastic components. For example, upon servicing some pick ups the dumpsters may require additional cleaning time if, for example, it has been graffitied. Also, drive times between destination and wait times at landfills may vary randomly across the day.



Finally, the last modeling complication we mention is that of nesting dumpsters. It is sometimes possible to nest empty dumpsters inside of each other to allow multiple deliveries after visiting a given landfill.

## Simplifications

Although we have a model that describes many of these aspects, we will only consider the simplest of models as input for Cplex. In particular, we will make all of the following assumptions:

- We only consider a single dumpster size

Considering more dumpster sizes is not much harder, but complicates the model unnecessarily by adding several indices.

- We assume only one type of truck capable of servicing all requests

This also allows a one-to-one mapping between drivers and trucks.

- We assume no time windows
- The time between stops is constant (we used a google API to determine distances between locations in Denver)

## Formulation

### Previous Heuristic

Because a heuristic I used in the past influenced the formulation provided here, we will very briefly describe it. Because each driver must visit one of the landfills no later than his third stop from a previous landfill, we found it convenient to break schedules into ``legs." Each leg begins and ends in a landfill and can optionally have a delivery followed by an optional pick up. The heuristic had three phases:

- Assign deliveries to pick ups (This is a matching problem that is easily solved.)
- Form ``legs" between landfills and deliveries and between pick ups and landfills
- Assign each leg to a particular driver's route.

Then we would consider slightly less than optimal assignments of legs or assignments of deliveries to pick ups to see if shorter routes could be created. The assignment phase did not assign any deliveries to pick ups where the cost of doing so was greater than servicing the delivery alone. Lastly, pick ups were assigned to landfills based on the

inventory at that landfill.

## Constraints

The models proposed earlier in this section use decision variables for each \emph{edge} to indicate if they are taken. However, this makes several other aspects more difficult to model. Instead, we use a formulation that assigns requests to positions in a driver's schedule. This may not be the most efficient formulation as it requires many more variables, but it is more similar in structure to the heuristic that we developed earlier. We had hopes of warm starting Cplex using this heuristic so that we went with this formulation.

With eight drivers and thirty requests per day, most drivers don't visit more than eight stops. Therefore, we started by breaking each route into a sequence of  $R$  legs. Choosing a value of  $R$  that is too small will make the program not feasible if there are not enough legs to service all requests. However, there are values of  $R$  too small may force deliveries to be grouped with pick ups that should not be. Therefore care must be taken to chose  $R$  large enough to allow the optimal solution, but small enough that the problem is still tractable. For each driver, within each leg, a binary variable was created representing which path from landfill to delivery to pick up to delivery was chosen. That is all possible legs are enumerated first: if there are  $P$  pick ups,  $D$  deliveries, and  $L$  landfills there are  $M = L \cdot (D + 1) \cdot (P + 1) \cdot L$  different possible legs. (The plus ones indicate that a deliver or pick up is not visited.) We then create a boolean variable  $x_{d,r,l}$  to determine whether driver  $d \in \{1, 2, \dots, T\}$  uses leg  $l \in \{1, 2, M\}$  as his  $r \in \{1, 2, \dots, R\}$  th leg.

We can use data indicators  $p_{li} \forall 1 \leq l \leq M, 1 \leq i \leq P$  to be 1 if leg  $l$  services pick up  $i$ , and zero otherwise. We create data indicators  $d_{li} \forall 1 \leq l \leq M, 1 \leq i \leq D$  to indicate if leg  $l$  services delivery  $i$ . Lastly, we create data indicators  $s_{li} \forall 1 \leq l \leq M, 1 \leq i \leq L$  and  $f_{li} \forall 1 \leq l \leq M, 1 \leq i \leq L$  to indicate if leg  $l$  starts at or finishes at landfill  $i$ .

The last ingredient is the time. We can assign a time  $t_{dr} \geq 0 \forall 1 \leq d \leq T, 1 \leq r \leq R$  completed to each leg in a driver's route. If the time required to take leg  $l$  is given by  $c_l$ , then we can create an auxiliary variable  $u \geq 0 \forall 1 \leq d \leq T$  to simulate the maximum time take by all any driver.

Then we can finally write the constraints as follows, the set of all  $u, t_{dr}, x_{drl}$  that satisfy

$$\begin{aligned}
\sum_{d=1}^D \sum_{l=1}^M \sum_{r=1}^R x_{dlr} p_{li} &= 1 & \forall 1 \leq i \leq P \\
\sum_{d=1}^D \sum_{l=1}^M \sum_{r=1}^R x_{dlr} d_{li} &= 1 & \forall 1 \leq i \leq D \\
\sum_{l=1}^M x_{dl,r+1} l_{li} &\leq \sum_{l=1}^M x_{dlr} s_{li} & \forall 1 \leq d \leq D, 1 \leq r \leq R-1, 1 \leq i \leq L \\
x_{dl0} &= 0 & \forall 1 \leq d \leq D, 2 \leq i \leq L \\
x_{dl,R-1} &= 0 & \forall 1 \leq d \leq D, 2 \leq i \leq L \\
\sum_{l=1}^M x_{dlr} &= 1 & \forall 1 \leq d \leq T, 1 \leq r \leq R \\
t_{dr} &\geq t_{d,r-1} + \sum_{l=1}^M x_{dlr} c_l & \forall 1 \leq d \leq D, 2 \leq r \leq R \\
u &\geq t_{d,R-1} & \forall 1 \leq d \leq T \\
t_{dr} &\geq 0 & \forall 1 \leq d \leq D, 1 \leq r \leq R \\
u &\geq 0 \\
x_{dlr} &\in \{0, 1\} & \forall 1 \leq d \leq T, 1 \leq l \leq M, 1 \leq r \leq R
\end{aligned}$$

will be denoted as  $F$ .

The first two constraints require that each pick up and deliver is visited exactly once. Next, we dictate that a driver cannot leave a landfill he did not enter in the last leg. Then we ensure that each driver must start and end at the first depot. Lastly, before considering time constraints, we ensure a driver can only do one thing during each leg.

The first time constraint says that the completion time of the  $d$ th driver on his  $r$ th leg must be greater than his time at the  $r - 1$  stop plus the time taken on the  $r$ th leg, while the third to last constraint requires that the total time  $u$  must be bigger than any time taken by any driver.

I am not proud of the constraints and variables used, but this is a somewhat simple way to formulate the problem. It would likely be better to formulate the problem with edges between each stop. Another major problem with this formulation is that it has symmetry issues. Note that if a driver stays at a landfill for leg  $r$ , then that driver could have stayed at another landfill for any other leg. Not only this, but all drivers are indistinguishable.

Many of these constraints are what are known as specially ordered sets or *SOS* - 1 constraints, which are typically nicer, as the depth of a branch and bound tree only needs to be logarithmic in the number of variables. These kinds of constraints require only one of a set of binary variables to be turned on. This may let the optimization library make better branches. One example of how this kind of information could be useful is in the following: if  $x_i, 1 \leq i \leq 10$  are binary variables such that



$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} = 1$ , a naive approach would be to branch on  $x_1 = 0$  or  $x_1 = 1$ , and continue in the fashion. However, this would lead to a complete enumeration of all possible solutions, it may be much better to branch on  $x_1 + x_2 + x_3 + x_4 + x_5 = 1$  or  $x_1 + x_2 + x_3 + x_4 + x_5 = 0$  to limit the number of branches. This indication that  $x_i$  are in an  $SOS - 1$  constraint may allow the library to make smarter branches.

## Inventory Constraints

The original plan was to also consider inventory constraints. These constraints are very problematic because they apply to all drivers based on the time. One approach would be to first solve the problem without inventory constraints. Given a solution, we could simply check what times the inventories become negative and add constraints to ensure that all inventories are positive at that time. For example, if it is a solution to the previous problem becomes infeasible at times  $v_k$  for some set times  $v_k, 1 \leq k \leq K$ , we could try to rule these solutions out in the following manner. Suppose that the initial inventories are given by  $I_i^s \forall 1 \leq i \leq L$ . We introduce nuisance variables  $b_{drk} \forall 1 \leq d \leq T, 1 \leq r \leq R, 1 \leq k \leq K$  to signify if the  $r$ th leg of driver  $d$  is completed before time  $v_k$ . We can ensure this by stipulating that

$$t_{dr} \leq v_k + b_{drk}B$$

where  $B$  is an arbitrarily large number. Then we create inventory variables  $I_{ki}$  for each time  $k$  at each landfill  $i$ . These are integers but do not need to be explicitly declared as such as they will be forced to be integral within the program. We can create data  $m_{li} \forall 1 \leq l \leq M, 1 \leq i \leq L$  representing the change in inventory at landfill  $i$  from performing leg  $l$ . For example, if leg  $l$  picks up a dumpster from landfill  $i$  and delivers it before pick up a dumpster on the way to landfill  $j$ , then  $m_{li} = -1$  and  $m_{lj} = +1$ . Then inventories can be tracked with the constraint

$$I_{ki} = I_i^s + \sum_{d=1}^D \sum_{l=1}^M \sum_{r=1}^R x_{dlr} b_{drk} m_{li} \quad \forall 1 \leq k \leq K, 1 \leq i \leq L$$

The last piece is to ensure that the inventories are never negative:

$$I_{ki} \geq 0 \quad \forall 1 \leq k \leq K, 1 \leq i \leq L.$$

However, there is a major problem with this approach: it introduces quadratic terms between  $x_{dlr} b_{drk}$ . There are ways to enforce linearity because these are both binary variables, but this formulation is already out of hand without including inventory constraints. Another caveat is that this does not make a distinction between a dumpster being taken at the \emph{beginning} of the leg and a new dumpster being delivered at the \emph{end} of the leg.

The inventory constraints make most otherwise feasible solutions infeasible, as the  $I_i^s$  are usually small. Because these constraints cut off much of the feasible region, these could be nice to include within a search algorithm to narrow the number of solutions to be tested. However, adding them to a linear program formulation makes the problem much more complicated and does not imply a speed up.

## Objective

There are many different considerations when choosing the objective. One thing that might be nice to consider is the end of day inventories: if we leave a landfill with no dumpsters, then scheduling the next day may prove to be difficult. Also, in practice, we really wish to maximize the number of stops we are capable of visiting, subject to finishing the stops in one day.

Initially, we chose to minimize the time taken to service all requests, that is  $\min\{u : (u, t_{dr}, x_{dlr}) \in F\}$ . However, this has some drawbacks. First of all, it introduces much more symmetry: the objective does not change for several different solutions where the driver who takes the most time does not change his route. Also, the cost in time is simply the sum of all times:  $\sum_{d=1}^T t_{d,R-1}$  so that if we could make this smaller without affecting the longest time spent by any driver. However, simply minimizing this assigns nearly all stops to a single driver. Therefore, we introduced a parameter  $\lambda$  and minized a weighted sum of both of these expressions:  $\min\{\lambda T u + \sum_{d=1}^T t_{d,R-1} : (u, t_{dr}, x_{dlr}) \in F\}$ . In our code, we assigned  $\lambda = 2$ .

## Computational Results

### Implementation Details

#### Problem Generation

To learn how to use Cplex, we created randomly generated problem sets, with locations being chosen with a uniform distance over  $[0, 1]^2$  distances given by the euclidean norm. We considered values of  $P$  between 3 and 15, values of  $D$  between 3 and 15, values of  $L$  between 2 and 4, values of  $R$  between 5 and 10.

#### Source Code

The source code I used has most likely changed since I edited this, but a snapshot containing most of the work is contained at Sams code.

#### Local Search

Initially, I used the object  $\min\{u : (u, t_{dr}, x_{dlr}) \in F\}$ , and I found several solutions that were visibly less than optimal by changing only a few vertices. However, upon further examination, I realized that these were only improvements for drivers whose time was not the maximum time. Upon changing the objective to  $\min\{\lambda T u + \sum_{d=1}^T t_{d,R-1} : (u, t_{dr}, x_{dlr}) \in F\}$ , I was no longer able to find simple solutions to the solutions found by Cplex.

My attempts are found in LocalSearch.java. The local search operation was to change any two sequential landfills within a route while swapping any two delivers or two swapping any two pick ups. Unfortunately, swapping any two delivers or swapping any two pick ups at the same time was too expensive. I had hoped that by starting a random search, and sequentially improving the solution by considering all these changes would provide a good warm start to the cplex optimizer. However, the cplex solver was good

Typesetting math: 100%

a few seconds, I was not able to improve the solutions Cplex had.

While solving the MIP, cplex maintains a pool of feasible solutions currently found. Although this may not be the proper approach, I used the populate method to give breaks while solving for me to access these. However, this did not reduce the optimality gap as quickly as simply calling the solve method.

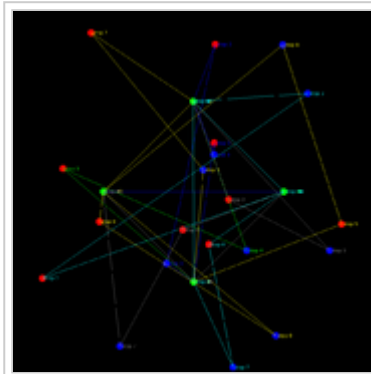
Example output

Shown to the right, is example output from the cplex solver. Red circles correspond to deliveries, blue circles are pick ups, and green circles are landfills. The lines between these circles are the routes chosen by cplex, and the landfill all the way to the right is the beginning depot.

Runtimes

Cplex was not able to solve this formulation for realistic problem sizes. I found that with  $L = 2, P = 5, D = 5, R = 5$ , cplex was able to solve to optimality within 65 seconds. If I let cplex run for one minute, the optimality gaps found by cplex were:

$L$	$P$	$D$	$R$	$T$	objective	number of binaries	number of rows	Optimality Gap \%
2	5	5	5	5	25.15	2880	90	0.01
2	10	5	5	5	25.1002	5280	95	4.94
2	5	10	5	5	24.3465	5280	95	5.15
4	10	10	5	5	26.4809	33880	140	19.77
4	15	15	8	8	43.8683	213009	342	27.19



Example output of the cplex solver

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sam%27s\\_Hauling\\_and\\_Vehicle\\_Routing\\_Problems&oldid=544](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sam%27s_Hauling_and_Vehicle_Routing_Problems&oldid=544)"

- This page was last modified on 3 May 2017, at 12:56.
- This page has been accessed 9,542 times.

# Sams code

From CU Denver Optimization Student Wiki

This page contains a snapshot of my source code for experimenting with Cplex for Sam's Hauling

## Contents

- 1 Solution.java
- 2 Problem.java
- 3 DepotArc.java
- 4 Location.java
- 5 LocalSearch.java
- 6 Painter.java
- 7 SecondMain.java

## Solution.java

```
package second;

import ilog.concert.IloException;
import ilog.concert.IloLinearNumExpr;
import ilog.concert.IloNumVar;
import ilog.cplex.IloCplex;
import ilog.cplex.IloCplex.UnknownObjectException;

import java.io.IOException;

public class Solution
{
    Problem problem;
    Leg[][] legs;
    double lambda = 2;

    IloNumVar totalTime;
    double totalTimeValue;

    public Solution(Problem problem)
    {
        this.problem = problem;

        legs = new Leg[problem.numTrucks][];
        for (int i = 0; i < legs.length; i++)
        {
            legs[i] = new Leg[problem.maximumLegs];
```

```

        for (int j = 0; j < legs[i].length; j++)
            legs[i][j] = new Leg();
    }
}

public Solution(Solution solution)
{
    this.problem = solution.problem;
    this.totalTime = solution.totalTime;
    this.totalTimeValue = solution.totalTimeValue;

    legs = new Leg[problem.numTrucks][];
    for (int i = 0; i < legs.length; i++)
    {
        legs[i] = new Leg[problem.maximumLegs];
        for (int j = 0; j < legs[i].length; j++)
        {
            legs[i][j] = new Leg();
            legs[i][j].assign(solution.legs[i][j]);
        }
    }
}

public String toString()
{
    StringBuilder builder = new StringBuilder();

    for (int driver = 0; driver < legs.length; driver++)
    {
        builder.append("Driver ").append(driver).append('\t');
        for (int stop = 0; stop < legs[driver].length; stop++)
        {
            builder.append(stop).append(':').append(legs[driver][stop]);
        }
        builder.append('\n');
    }

    return builder.toString();
}

public double getCost()
{
    double maxTime = 0;
    double sumOfTimes = 0;

    for (int driver = 0; driver < legs.length; driver++)
    {
        double driverTime = 0;
        for (int stop = 0; stop < legs[driver].length; stop++)
        {
            driverTime += problem.arcs[legs[driver][stop].depotIndex].cost;
        }

        maxTime = Math.max(maxTime, driverTime);
        sumOfTimes += driverTime;
    }

    return legs.length * lambda * maxTime + sumOfTimes;
}

public static Solution createRandomSolution(Problem problem)
{

```

```

Solution solution = new Solution(problem);

for (int pickup = 0; pickup < problem.pickups.length; pickup++)
{
    while (true)
    {
        int driver = SecondMain.RANDOM.nextInt(solution.legs.length);
        int leg = SecondMain.RANDOM.nextInt(solution.legs[driver].length);
        if (solution.legs[driver][leg].pickup >= 0)
            continue;

        solution.legs[driver][leg].pickup = pickup;
        break;
    }
}
for (int delivery = 0; delivery < problem.deliveries.length; delivery++)
{
    while (true)
    {
        int driver = SecondMain.RANDOM.nextInt(solution.legs.length);
        int leg = SecondMain.RANDOM.nextInt(solution.legs[driver].length);
        if (solution.legs[driver][leg].delivery >= 0)
            continue;

        solution.legs[driver][leg].delivery = delivery;
        break;
    }
}
for (int driver = 0; driver < solution.legs.length; driver++)
{
    for (int leg = 0; leg < solution.legs[driver].length; leg++)
    {
        Leg l = solution.legs[driver][leg];
        if (l.delivery < 0 && l.pickup < 0)
            continue;
        l.landfill = SecondMain.RANDOM.nextInt(problem.landfills.length);
    }
}

return solution;
}

public void validate()
{
    problem.validate();

    for (int pickup = 0; pickup < problem.pickups.length; pickup++)
    {
        int count = 0;
        for (int driver = 0; driver < legs.length; driver++)
            for (int stop = 0; stop < legs[driver].length; stop++)
            {
                if (legs[driver][stop].pickup != pickup)
                    continue;
                count++;
            }
        if (count != 1)
            throw new RuntimeException("Found pickup " + pickup + " " + count + " times.");
    }
    for (int delivery = 0; delivery < problem.deliveries.length; delivery++)
    {
        int count = 0;

```

```

        for (int driver = 0; driver < legs.length; driver++)
            for (int stop = 0; stop < legs[driver].length; stop++)
            {
                if (legs[driver][stop].delivery != delivery)
                    continue;
                count++;
            }
        if (count != 1)
            throw new RuntimeException("Found delivery " + delivery + " " + count + " times.");
    }
    for (int driver = 0; driver < legs.length; driver++)
    {
        if (problem.arcs[legs[driver][0].depotIndex].landfill1 != 0)
            throw new RuntimeException("Driver does not start at the depot");
        if (problem.arcs[legs[driver][legs[driver].length-1].depotIndex].landfill2 != 0)
            throw new RuntimeException("Driver does not end at the depot");

        for (int stop = 1; stop < legs[driver].length; stop++)
            if (problem.arcs[legs[driver][stop].depotIndex].landfill1 != legs[driver][stop-1].landfill)
                throw new RuntimeException("Driver does not start where he left");
    }
}

public void solveWithCplex() throws IloException, IOException
{
    IloCplex cplex = new IloCplex();

    totalTime = cplex.numVar(0, Double.MAX_VALUE, "total_time");

    for (int driver = 0; driver < legs.length; driver++)
        for (int stop = 0; stop < legs[driver].length; stop++)
            legs[driver][stop].createVariables(cplex, problem, "driver_" + driver + "_" + stop + "_");

    // has to pick something
    for (int driver = 0; driver < legs.length; driver++)
        for (int stop = 0; stop < legs[driver].length; stop++)
            legs[driver][stop].addCanOnlyDoOneThingConstraint(cplex, driver, stop);

    // has to stop and start
    for (int driver = 0; driver < legs.length; driver++)
    {
        legs[driver][0].createStartsConstraints(cplex, problem, driver);
        legs[driver][legs[driver].length-1].createEndsConstraints(cplex, problem, driver);
    }

    // has to leave current depot
    for (int driver = 0; driver < legs.length; driver++)
        for (int stop = 1; stop < legs[driver].length; stop++)
            legs[driver][stop].hasToLeaveCurrentDepotConstraint(cplex, problem, driver, stop, legs[driver][stop-1]);

    // has to visit each pickup
    for (int pickup = 0; pickup < problem.pickups.length; pickup++)
    {
        IloLinearNumExpr v = cplex.linearNumExpr();
        for (int driver = 0; driver < legs.length; driver++)
            for (int stop = 0; stop < legs[driver].length; stop++)
                legs[driver][stop].pickupHasToBeVisitedConstraintConstraint(v, problem, pickup);
        cplex.addEq(1, v, "someone_needs_to_visit_pickup_" + pickup);
    }
}

```

```

// has to visit each delivery
for (int delivery = 0; delivery < problem.deliveries.length; delivery++)
{
    IloLinearNumExpr v = cplex.linearNumExpr();
    for (int driver = 0; driver < legs.length; driver++)
        for (int stop = 0; stop < legs[driver].length; stop++)
            legs[driver][stop].deliverHasToBeVisitedConstraintConstraint(v, problem, delivery);
    cplex.addEq(1, v, "someone_needs_to_visit_delivery_" + delivery);
}

// must set the time
for (int driver = 0; driver < legs.length; driver++)
    for (int stop = 0; stop < legs[driver].length; stop++)
        legs[driver][stop].hasToHaveTheTime(cplex, problem, driver, stop, stop == 0 ? null : legs[driver][stop-1]);
for (int driver = 0; driver < legs.length; driver++)
    legs[driver][legs[driver].length-1].timeMustBeLessThanMaximum(cplex, driver, totalTime);

IloLinearNumExpr obj = cplex.linearNumExpr();

obj.addTerm(lambda * problem.numTrucks, totalTime);

for (int driver=0; driver<legs.length; driver++)
    obj.addTerm(1, legs[driver][legs[driver].length-1].time);

cplex.addMinimize(obj);

cplex.exportModel("/home/yz/Documents/Source/eclipse-java/workspace/output/foobar.lp");

for (int count = 0; count < 20; count++)
{
    cplex.setParam(IloCplex.IntParam.TimeLimit, 60);
    if (cplex.populate())
    {
        System.out.println("Solution status: " + cplex.getStatus());
        double tolerance = cplex.getParam(IloCplex.Param.MIP.Tolerances.Integrality);
        // if (cplex.getValue(open[j]) >= 1 - tolerance)

        int nsols = cplex.getSolnPoolNsols();
        for (int sol = 0; sol < nsols; sol++)
        {
            Solution otherSolution = new Solution(this);
            otherSolution.assign(cplex, sol);
            otherSolution.validate();

            String filename = "/home/yz/Documents/Source/eclipse-java/workspace/output/test_"
                + String.format("%04d_", count)
                + String.format("%04d", sol) + ".png";
            otherSolution.write(cplex, filename, sol);

            System.out.println(otherSolution);
            LocalSearch localSearch = new LocalSearch(otherSolution);
            localSearch.searchAllLandfills();
        }
    }
}
cplex.end();
}

public void compareCosts(IloCplex cplex)
{
    double maxTime = 0;
    double sumOfTimes = 0;

```



```

for (int driver = 0; driver < legs.length; driver++)
{
    double driverTime = 0;
    for (int stop = 0; stop < legs[driver].length; stop++)
    {
        double thisCost = problem.arcs[legs[driver][stop].depotIndex].cost;
        driverTime += problem.arcs[legs[driver][stop].depotIndex].cost;

        if (Math.abs(driverTime - legs[driver][stop].finishTime) > 1e-4)
        {
            System.err.println("driver " + driver + " stop " + stop);
            System.err.println("\t\tmy time: " + driverTime);
            System.err.println("\t\ttheir time: " + legs[driver][stop].finishTime);
        }
    }

    maxTime = Math.max(maxTime, driverTime);
    System.out.println("My    max time for driver " + driver + ": " + driverTime);
    System.out.println("Their max time for driver " + driver + ": " + legs[driver][legs[driver].length-1].finishTime);
    sumOfTimes += driverTime;
}
System.out.println("My    max time : " + maxTime);
System.out.println("Their max time : " + totalTimeValue);

System.out.println("objective value: " + (legs.length * lambda * maxTime + sumOfTimes));
}

public void write(IloCplex cplex, String filename, int index) throws IOException, IOException
{
    System.out.println("Optimal value: " + cplex.getObjValue(index));
    System.out.println("My computed objective: " + getCost());
    compareCosts(cplex);
    System.out.println("Wrote to " + filename);
    Painter.draw(this, filename);
}

public void assign(Solution other)
{
    for (int driver = 0; driver < legs.length; driver++)
        for (int stop = 0; stop < legs[driver].length; stop++)
            legs[driver][stop].assign(other.legs[driver][stop]);
}

public void assign(IloCplex cplex, int index) throws UnknownObjectException, IOException
{
    totalTimeValue = cplex.getValue(totalTime, index);

    for (int driver = 0; driver < legs.length; driver++)
        for (int stop = 0; stop < legs[driver].length; stop++)
        {
            legs[driver][stop].assign(problem, cplex, index);

            if (stop > 0)
            {
                double nCost = problem.arcs[ legs[driver][stop ].depotIndex].cost;
                double pCost = cplex.getValue(legs[driver][stop-1].time, index);
                double cCost = cplex.getValue(legs[driver][stop ].time, index);

                if (pCost + nCost > cCost)
                {
                    System.out.println("driver: " + driver);

```

```

        System.out.println("stop: " + stop );
        System.out.println("nCost: " + nCost );
        System.out.println("pCost: " + pCost );
        System.out.println("cCost: " + cCost );
        System.out.println("assigned to: " + cplex.getValue(legs[driver][stop].arcIdx[legs[driver][stop].depotIndex], index));
        System.out.println("variable name: " + legs[driver][stop].arcIdx[legs[driver][stop].depotIndex].getName());
        System.out.println("index: " + legs[driver][stop].depotIndex);

        throw new RuntimeException("uh oh!!!!");
    }
}

public static class Leg
{
    int depotIndex = -1;
    IloNumVar[] arcIdx;
    IloNumVar time;

    int start = -1;
    int delivery = -1;
    int pickup = -1;
    int landfill = -1;

    double finishTime;

    public String toString()
    {
        StringBuilder builder = new StringBuilder();

        builder.append('[');
        builder.append("start: ").append(String.format("%03d", start)).append(',');
        builder.append("delivery:").append(String.format("%03d", delivery)).append(',');
        builder.append("pickup: ").append(String.format("%03d", pickup)).append(',');
        builder.append("landfill:").append(String.format("%03d", landfill)).append(',');
        builder.append("time: ").append(String.format("%08f", finishTime));
        builder.append(']');

        return builder.toString();
    }

    public void set(Problem problem, int l1, int delivery, int pickup, int l2)
    {
        this.start = l1;
        this.delivery = delivery;
        this.pickup = pickup;
        this.landfill = l2;
        depotIndex = problem.getIndex(l1, delivery, pickup, l2);
    }

    public void assign(Leg leg)
    {
        this.depotIndex = leg.depotIndex;
        this.arcIdx = leg.arcIdx;
        this.time = leg.time;
        this.start = leg.start;
        this.delivery = leg.delivery;
        this.pickup = leg.pickup;
        this.landfill = leg.landfill;
        this.finishTime = leg.finishTime;
    }
}

```

```

public void createVariables(IloCplex cplex, Problem problem, String prefix) throws IloException
{
    arcIdx = cplex.boolVarArray(problem.arcs.length);

    for (int i = 0; i < arcIdx.length; i++)
        arcIdx[i].setName(prefix + "arc_" + i);

    time = cplex.numVar(0, Double.MAX_VALUE, prefix + "_time");
}

public void addCanOnlyDoOneThingConstraint(IloCplex cplex, int driver, int stop) throws IloException
{
    IloLinearNumExpr arc = cplex.linearNumExpr();
    for (int i = 0; i < arcIdx.length; i++)
        arc.addTerm(1, arcIdx[i]);
    cplex.addEq(1, arc, "driver_" + driver + "_does_one_thing_for_leg_" + stop);
}

public void pickupHasToBeVisitedConstraint(IloLinearNumExpr v, Problem problem, int pickup) throws IloException
{
    for (int arc = 0; arc < arcIdx.length; arc++)
        if (problem.arcs[arc].pickup == pickup)
            v.addTerm(1, arcIdx[arc]);
}

public void deliverHasToBeVisitedConstraint(IloLinearNumExpr v, Problem problem, int deliver) throws IloException
{
    for (int arc = 0; arc < arcIdx.length; arc++)
        if (problem.arcs[arc].delivery == deliver)
            v.addTerm(1, arcIdx[arc]);
}

public void createStartsConstraints(IloCplex cplex, Problem problem, int driver) throws IloException
{
    for (int arc = 0; arc < arcIdx.length; arc++)
        if (problem.arcs[arc].landfill1 != 0)
            arcIdx[arc].setUB(0);
}

public void createEndsConstraints(IloCplex cplex, Problem problem, int driver) throws IloException
{
    for (int arc = 0; arc < arcIdx.length; arc++)
        if (problem.arcs[arc].landfill2 != 0)
            arcIdx[arc].setUB(0);
}

public void hasToLeaveCurrentDepotConstraint(IloCplex cplex, Problem problem, int driver, int stop, Leg previous) throws IloException
{
    for (int landfill = 0; landfill < problem.landfills.length; landfill++)
    {
        IloLinearNumExpr ended = cplex.linearNumExpr();
        for (int arc = 0; arc < previous.arcIdx.length; arc++)
            if (problem.arcs[arc].landfill2 == landfill)
                ended.addTerm(1, previous.arcIdx[arc]);
        IloLinearNumExpr started = cplex.linearNumExpr();
        for (int arc = 0; arc < arcIdx.length; arc++)
            if (problem.arcs[arc].landfill1 == landfill)
                started.addTerm(1, arcIdx[arc]);

        cplex.addEq(started, ended, "driver_" + driver + "_on_leg_" + stop + "_leaves_" + landfill + "_if_he_is_there");
    }
}

```

```

public void hasToHaveTheTime(IloCplex cplex, Problem problem, int driver, int stop, Leg previous) throws IloException
{
    IloLinearNumExpr timeToFinishThisLeg = cplex.linearNumExpr();

    if (previous != null)
        timeToFinishThisLeg.addTerm(1, previous.time);

    for (int arc = 0; arc < arcIdx.length; arc++)
        timeToFinishThisLeg.addTerm(problem.arcs[arc].cost, arcIdx[arc]);

    cplex.addLe(timeToFinishThisLeg, time, "driver_" + driver + "_must_complete_stop_" + stop + "_after_in_order");
}

public void timeMustBeLessThanMaximum(IloCplex cplex, int driver, IloNumVar maxTime) throws IloException
{
    cplex.addLe(time, maxTime, "driver_" + driver + "_must_finish_before_max_time");
}

public void assign(Problem problem, IloCplex cplex, int index) throws UnknownObjectException, IloException
{
    depotIndex = -1;

    for (int arc = 0; arc < arcIdx.length && depotIndex < 0; arc++)
    {
        if (index < 0 && cplex.getValue(arcIdx[arc]) > .5)
            depotIndex = arc;
        if (index >= 0 && cplex.getValue(arcIdx[arc], index) > .5)
            depotIndex = arc;
    }

    if (depotIndex < 0)
        throw new RuntimeException("No value set!");

    start    = problem.arcs[depotIndex].landfill1;
    delivery = problem.arcs[depotIndex].delivery;
    pickup   = problem.arcs[depotIndex].pickup;
    landfill  = problem.arcs[depotIndex].landfill2;

    finishTime = cplex.getValue(time, index);
}
}
}

```

## Problem.java

```

package second;

public class Problem
{
    Location[] deliveries;
    Location[] pickups;
    Location[] landfills;

    double[][] l2l; // landfill to landfill

```

```

double[][] l2p; // landfill to pickups
double[][] l2d; // landfill to delivery

double[][] d2l; // delivery to landfill
double[][] d2p; // delivery to pickups

double[][] p2l; // pickup to landfill

int numTrucks;

DepotArc[] arcs;
// Map<DepotArc, Integer> arcToIdx = new HashMap<>();
int[][][][] arcToIdx;

int maximumLegs;

public void setDepotArcs()
{
    arcs = new DepotArc[landfills.length * (deliveries.length + 1) * (pickups.length + 1) * landfills.length];

    arcToIdx = new int[landfills.length][][][];
    int idx = 0;
    for (int landfill1 = 0; landfill1 < landfills.length; landfill1++) {
        arcToIdx[landfill1] = new int[deliveries.length+1][][];
        for (int deliver = -1; deliver < deliveries.length; deliver++) {
            arcToIdx[landfill1][deliver+1] = new int[pickups.length+1][];
            for (int pickup = -1; pickup < pickups.length; pickup++) {
                arcToIdx[landfill1][deliver+1][pickup+1] = new int[landfills.length];
                for (int landfill2 = 0; landfill2 < landfills.length; landfill2++) {
                    arcs[idx] = new DepotArc(this, landfill1, deliver, pickup, landfill2);
                    arcToIdx[landfill1][deliver+1][pickup+1][landfill2] = idx;
                    idx++;
                }
            }
        }
    }
}

public int getIndex(int l1, int d, int p, int l2)
{
    return arcToIdx[l1][d+1][p+1][l2];
}

public void validate()
{
    for (int arc = 0; arc < arcs.length; arc++)
        if (getIndex(arcs[arc].landfill1, arcs[arc].delivery, arcs[arc].pickup, arcs[arc].landfill2) != arc)
            throw new RuntimeException("did not match!");
}

public static Problem generateRandomProblem()
{
    int numDeliveries = 10;
    int numPickups = 10;
    int numLandfills = 4;
    int numTrucks = 5;

    int maximumLegs = 10;

    Problem problem = new Problem();

```

```

        problem.deliveries = Location.createLocations(numDeliveries);
        problem.pickups = Location.createLocations(numPickups);
        problem.landfills = Location.createLocations(numLandfills);

        problem.l2l = Location.createDistanceMatrix(problem.landfills, problem.landfills);
        problem.l2p = Location.createDistanceMatrix(problem.landfills, problem.pickups);
        problem.l2d = Location.createDistanceMatrix(problem.landfills, problem.deliveries);

        problem.d2l = Location.createDistanceMatrix(problem.deliveries, problem.landfills);
        problem.d2p = Location.createDistanceMatrix(problem.deliveries, problem.pickups);

        problem.p2l = Location.createDistanceMatrix(problem.pickups, problem.landfills);

        for (int i = 0; i < numLandfills; i++)
        {
            problem.landfills[i].x = .5 + .25 * Math.cos(i * 2 * Math.PI / numLandfills);
            problem.landfills[i].y = .5 + .25 * Math.sin(i * 2 * Math.PI / numLandfills);
        }

        problem.numTrucks = numTrucks;
        problem.maximumLegs = maximumLegs;

        problem.setDepotArcs();

        return problem;
    }
}

```

## DepotArc.java

```

package second;

public class DepotArc implements Comparable<DepotArc>
{
    int landfill1;
    int delivery;
    int pickup;
    int landfill2;

    double cost;

    public DepotArc(Problem p, int landfill1, int delivery, int pickup, int landfill2)
    {
        this.landfill1 = landfill1;
        this.delivery = delivery;
        this.pickup = pickup;
        this.landfill2 = landfill2;

        if (delivery < 0)
        {
            if (pickup < 0)
                cost = p.l2l[landfill1][landfill2];
            else
                cost = p.l2p[landfill1][pickup] + p.p2l[pickup][landfill2];
        }
    }
}

```

```

    }
    else
    {
        if (pickup < 0)
            cost = p.l2d[landfill1][delivery] + p.d2l[delivery][landfill2];
        else
            cost = p.l2d[landfill1][delivery] + p.d2p[delivery][pickup] + p.p2l[pickup][landfill2];
    }
}

public String toString()
{
    StringBuilder builder = new StringBuilder();

    builder.append('[');
    builder.append("L:").append(landfill1).append(',');
    builder.append("D:").append(delivery).append(',');
    builder.append("P:").append(pickup).append(',');
    builder.append("L:").append(landfill2).append(',');
    builder.append(']');

    return builder.toString();
}

public boolean equals(Object other)
{
    if (!(other instanceof DepotArc))
    {
        return false;
    }
    return equals((DepotArc) other);
}

public boolean equals(DepotArc o)
{
    return landfill1 == o.landfill1 &&
        pickup == o.pickup &&
        delivery == o.delivery &&
        landfill2 == o.landfill2;
}

public int hashCode() {
    return toString().hashCode();
}

@Override
public int compareTo(DepotArc o) {
    int cmp;
    cmp = Integer.compare(landfill1, o.landfill1);
    if (cmp != 0) return cmp;
    cmp = Integer.compare(delivery, o.delivery);
    if (cmp != 0) return cmp;
    cmp = Integer.compare(pickup, o.pickup);
    if (cmp != 0) return cmp;
    cmp = Integer.compare(landfill2, o.landfill2);
    if (cmp != 0) return cmp;
    return 0;
}
}

```

# Location.java

```
package second;

public class Location
{
    public double x, y;

    public Location(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public static Location generateRandomLocation()
    {
        return new Location(SecondMain.RANDOM.nextDouble(), SecondMain.RANDOM.nextDouble());
    }

    public static Location[] createLocations(int size)
    {
        Location[] locations = new Location[size];

        for (int i = 0; i < locations.length; i++)
            locations[i] = generateRandomLocation();

        return locations;
    }

    public static double[][] createDistanceMatrix(Location[] from, Location[] to)
    {
        double[][] ret = new double[from.length][to.length];
        for (int i = 0; i < from.length; i++)
        {
            for (int j = 0; j < to.length; j++)
            {
                double dx = from[i].x - to[j].x;
                double dy = from[i].y - to[j].y;
                ret[i][j] = Math.sqrt(dx * dx + dy * dy);
            }
        }
        return ret;
    }
}
```

# LocalSearch.java

```
package second;

import second.Solution.Leg;

public class LocalSearch {

    Solution solution;
```



Solution oldSolution;

```
public LocalSearch(Solution solution)
{
    this.oldSolution = solution;
    this.solution = new Solution(solution);
}

public class OneMod
{
    double oldCost;

    boolean swapP;
    int pdriver1;
    int pstop1;
    int pdriver2;
    int pstop2;

    boolean swapD;
    int ddriver1;
    int dstop1;
    int ddriver2;
    int dstop2;

    int ldriver;
    int lstop;

    public void reset()
    {
        oldCost = oldSolution.getCost();

        swapP = false;
        pdriver1 = 0;
        pdriver2 = 0;
        pstop1 = 0;
        pstop2 = 0;

        swapD = false;
        ddriver1 = 0;
        ddriver2 = 0;
        dstop1 = 0;
        dstop2 = 0;

        ldriver = 0;
        lstop = 0;
    }

    public void check()
    {
        checkL1();
    }

    public void checkL1()
    {
        if (lstop == 0)
        {
            checkL2();
        }
        else
        {
            Leg lcurr1 = solution.legs[ldriver][lstop];
            for (int landfill = 0; landfill < solution.problem.landfills.length; landfill++)
```

```

        {
            if (landfill == lcurr1.start)
                continue;

            Leg lprev1 = solution.legs[ldriver][lstop-1];
            lprev1.set(solution.problem, lprev1.start, lprev1.delivery, lprev1.pickup, landfill);
            lcurr1.set(solution.problem, landfill, lcurr1.delivery, lcurr1.pickup, lcurr1.landfill);
            checkL2();
        }
    }
}

public void checkL2()
{
    if (lstop == solution.legs[ldriver].length-1)
    {
        checkL3();
    }
    else
    {
        Leg lcurr1 = solution.legs[ldriver][lstop];
        for (int landfill = 0; landfill < solution.problem.landfills.length; landfill++)
        {
            Leg lnext1 = solution.legs[ldriver][lstop+1];

            if (landfill == lcurr1.landfill)
                continue;

            lcurr1.set(solution.problem, lcurr1.landfill, lcurr1.delivery, lcurr1.pickup, landfill);
            lnext1.set(solution.problem, landfill, lnext1.delivery, lnext1.pickup, lnext1.landfill);
            checkL3();
        }
    }
}

public void checkL3()
{
    solution.assign(oldSolution);

    if (swapD)
    {
        if (ddriver1 == ddriver2 && dstop1 == dstop2)
            return;

        Leg dleg1 = solution.legs[ddriver1][dstop1];
        Leg dleg2 = solution.legs[ddriver2][dstop2];
        int d1 = dleg1.delivery;
        int d2 = dleg2.delivery;

        if (d1 < 0 && d2 < 0)
            return;

        dleg1.set(solution.problem, dleg1.start, d2, dleg1.pickup, dleg1.landfill);
        dleg2.set(solution.problem, dleg2.start, d1, dleg2.pickup, dleg2.landfill);
    }

    if (swapP)
    {
        if (pdriver1 == pdriver2 && pstop1 == pstop2)
            return;

        Leg pleg1 = solution.legs[pdriver1][pstop1];
        Leg pleg2 = solution.legs[pdriver2][pstop2];
    }
}

```

```

        int p1 = pleg1.pickup;
        int p2 = pleg2.pickup;

        if (p1 < 0 && p2 < 0)
            return;

        pleg1.set(solution.problem, pleg1.start, pleg1.delivery, p2, pleg1.landfill);
        pleg2.set(solution.problem, pleg2.start, pleg2.delivery, p1, pleg2.landfill);
    }

    solution.validate();

    double newCost = solution.getCost();
    if (newCost < oldCost)
    {
        System.out.println("Found better cost: old=" + oldCost + ", new cost=" + newCost);
    }
}

@Override
public String toString() {
    return "OneMod [oldCost=" + oldCost + "\n\tswapP=" + swapP + "\n\tpdriver1=" + pdriver1 + "\n\tpstop1=" + pstop1
        + "\n\tpdriver2=" + pdriver2 + "\n\tpstop2=" + pstop2 + "\n\tswapD=" + swapD + "\n\tddriver1=" + ddriver1
        + "\n\tdstop1=" + dstop1 + "\n\tddriver2=" + ddriver2 + "\n\tdstop2=" + dstop2 + "\n\tldriver=" + ldriver
        + "\n\tlstop=" + lstop + "]";
}

}

public void searchAllLandfills()
{
    long startTime = System.currentTimeMillis();

    OneMod oneMod = new OneMod();
    oneMod.oldCost = oldSolution.getCost();

    oneMod.swapD = false;
    oneMod.swapP = false;
    for (oneMod.ldriver = 0; oneMod.ldriver < solution.legs.length; oneMod.ldriver++)
    for (oneMod.lstop = 0; oneMod.lstop < solution.legs[oneMod.ldriver].length; oneMod.lstop++)
    {
        oneMod.check();
    }

    if (false)
    {
        oneMod.swapD = false;
        oneMod.swapP = true;
        for (oneMod.ldriver = 0; oneMod.ldriver < solution.legs.length; oneMod.ldriver++)
        for (oneMod.lstop = 1; oneMod.lstop < solution.legs[oneMod.ldriver].length; oneMod.lstop++)

        for (oneMod.pdriver1 = 0; oneMod.pdriver1 < solution.legs.length; oneMod.pdriver1++)
        for (oneMod.pstop1 = 0; oneMod.pstop1 < solution.legs[oneMod.pdriver1].length; oneMod.pstop1++)
        for (oneMod.pdriver2 = 0; oneMod.pdriver2 < solution.legs.length; oneMod.pdriver2++)
        for (oneMod.pstop2 = 0; oneMod.pstop2 < solution.legs[oneMod.pdriver2].length; oneMod.pstop2++)
        {
            oneMod.check();
        }

        oneMod.swapD = true;
        oneMod.swapP = false;
    }
}

```

```

for (oneMod.ldriver = 0; oneMod.ldriver < solution.legs.length; oneMod.ldriver++)
for (oneMod.lstop = 1; oneMod.lstop < solution.legs[oneMod.ldriver].length; oneMod.lstop++)

for (oneMod.ddriver1 = 0; oneMod.ddriver1 < solution.legs.length; oneMod.ddriver1++)
for (oneMod.dstop1 = 0; oneMod.dstop1 < solution.legs[oneMod.ddriver1].length; oneMod.dstop1++ )
for (oneMod.ddriver2 = 0; oneMod.ddriver2 < solution.legs.length; oneMod.ddriver2++)
for (oneMod.dstop2 = 0; oneMod.dstop2 < solution.legs[oneMod.ddriver2].length; oneMod.dstop2++ )
{
    oneMod.check();
}

}

long endTime = System.currentTimeMillis();

System.out.println("Local search took " + (endTime - startTime) / 1000.0 + "s");
}
}

```

## Painter.java

```

package second;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.OutputStream;
import java.nio.file.Files;
import java.nio.file.Paths;

import javax.imageio.ImageIO;

import second.Solution.Leg;

public class Painter
{
    private static final int WIDTH = 1000;
    private static final int HEIGHT = 1000;
    private static final int CIRCLE_RADIUS = 10;

    private static final Color[] DRIVER_COLORS = {
        Color.blue,
        Color.yellow,
        Color.green,
        Color.GRAY,
        Color.cyan,
        Color.ORANGE,
        Color.pink
    };

    private static Color getDriverColor(int driver)
    {
        if (driver < DRIVER_COLORS.length)

```

```

        {
            return DRIVER_COLORS[driver];
        }

        return SecondMain.createRandomColor();
    }

    public static void drawCircle(Graphics2D g, Location l)
    {
        g.fillOval(mapX(l.x) - CIRCLE_RADIUS,
                   mapY(l.y) - CIRCLE_RADIUS, 2 * CIRCLE_RADIUS, 2 * CIRCLE_RADIUS);
    }

    public static void drawLocations(Graphics2D g, Color c, Location[] l)
    {
        g.setColor(c);
        for (Location loc : l)
            drawCircle(g, loc);
    }

    public static void drawLine(Graphics2D g, Location l)
    {
    }

    public static void drawProblem(Graphics2D g, Problem p)
    {
        drawLocations(g, Color.red, p.deliveries);
        drawLocations(g, Color.blue, p.pickups);
        drawLocations(g, Color.green, p.landfills);
    }

    public static void drawLine(Graphics2D g, Location loc1, Location loc2)
    {
        g.drawLine(mapX(loc1.x), mapY(loc1.y), mapX(loc2.x), mapY(loc2.y));
    }

    public static void draw(Solution solution, String file) throws IOException
    {
        BufferedImage image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
        Graphics2D g = image.createGraphics();

        g.setColor(Color.black);
        g.fillRect(0, 0, WIDTH, HEIGHT);

        drawProblem(g, solution.problem);

        int driver = 0;
        for (Leg[] route : solution.legs)
        {
            Color driverColor = getDriverColor(driver++);

            double time = 0;
            int stopNumber = 1;

            Location prev = solution.problem.landfills[0];
            for (Leg leg : route)
            {
                if (leg.delivery >= 0)
                {
                    Location next = solution.problem.deliveries[leg.delivery];
                    g.setColor(driverColor);
                    drawLine(g, prev, next);
                }
            }
        }
    }

```

```

        g.setColor(Color.white);
        g.drawString("stop " + stopNumber++, mapX(next.x)+5, mapY(next.y)+5);
        prev = next;
    }
    if (leg.pickup >= 0)
    {
        Location next = solution.problem.pickups[leg.pickup];
        g.setColor(driverColor);
        drawLine(g, prev, next);

        g.setColor(Color.white);
        g.drawString("stop " + stopNumber++, mapX(next.x)+5, mapY(next.y)+5);
        prev = next;
    }
    if (leg.landfill >= 0)
    {
        Location next = solution.problem.landfills[leg.landfill];
        g.setColor(driverColor);
        drawLine(g, prev, next);

        g.setColor(Color.white);
        g.drawString("stop " + stopNumber++, mapX(next.x)+5, mapY(next.y)+5);
        prev = next;
    }
}

g.setColor(driverColor);
drawLine(g, prev, solution.problem.landfills[0]);
}

//
//
g.drawString("cost = " + solution.getCost(), 10, 20);

try (OutputStream newOutputStream = Files.newOutputStream(Paths.get(file));)
{
    ImageIO.write(image, "png", newOutputStream);
}

}

private static int mapX(double x)
{
    return (int) (WIDTH * x);
}
private static int mapY(double y)
{
    return (int) (HEIGHT * y);
}
}
}

```

## SecondMain.java

```

package second;

import ilog.concert.IloException;

import java.awt.Color;

```

```
import java.io.IOException;
import java.util.Random;

public class SecondMain
{
    public static final Random RANDOM = new Random(1776);
    public static Color createRandomColor()
    {
        return new Color(RANDOM.nextInt(255), RANDOM.nextInt(255), RANDOM.nextInt(255));
    }

    public static void main(String[] args) throws IOException, IOException
    {
        Problem problem = Problem.generateRandomProblem();
        Solution solution = Solution.createRandomSolution(problem);

        solution.solveWithCplex();
    }
}
```

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sams\\_code&oldid=413](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sams_code&oldid=413)"

- This page was last modified on 29 April 2017, at 20:21.
- This page has been accessed 1,759 times.

# Sam's Hauling and Vehicle Routing

From CU Denver Optimization Student Wiki

(Redirected from Sams)

Redirect page

 Sam's Hauling and Vehicle Routing Problems

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sam%27s\_Hauling\_and\_Vehicle\_Routing&oldid=545"

---

- This page was last modified on 3 May 2017, at 12:56.
- This page has been accessed 1,114 times.



# Sandra Robles

From CU Denver Optimization Student Wiki

Hello!

I'm a 2nd year M.S. Stats student with an interest in optimization. I took a 7 year break between undergrad and grad school, and have been working in Data Science since.

During the Fall semester of 2020, Hope Haygood & I worked on Denver Fire Response Distances for a project in our Linear Programming class.

During the Spring 2021 semester, I am working with Michael Burgher and Collin Powell on Vaccine Distribution for a project in our Integer Programming class.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sandra\\_Robles&oldid=3103](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sandra_Robles&oldid=3103)"

Category: Contributors

- 
- This page was last modified on 12 April 2021, at 09:43.
  - This page has been accessed 705 times.

# Semidefinite Programming

From CU Denver Optimization Student Wiki

Semidefinite programming is a generalization of linear programming. Much like linear programming, semidefinite programming is concerned with the optimization of a linear objective function over an affine subspace. The main difference is that the affine subspace has the additional constraint that it must be contained within the space of positive semidefinite matrices, denoted as  $\mathbb{S}^n$ .

## Contents

- 1 Similarities to Linear Programming
- 2 Restrictions on  $X$ 
  - 2.1 Positive Semidefinite
- 3 Solving the SDP
- 4 Geometry of the SDP
  - 4.1 Difference from Linear Programming

## Similarities to Linear Programming

There are many similarities between semidefinite programming and linear programming. Their standard formulation look vary similar, the main difference being the use of matrices instead of vectors for the variables, constraints, and objective function. For example, look at this formulation of a semidefinite program:

$$\begin{array}{ll} LP : & \min \quad c \cdot x \\ & s.t. \quad a_i \cdot x = b_i, \quad \forall i \in [m] \\ & \quad \quad x \in \mathbb{R}_+^n \end{array} \qquad \begin{array}{ll} SDP : & \min \quad C \bullet X \\ & s.t. \quad A_i \bullet X = b_i, \quad \forall i \in [m] \\ & \quad \quad X \succeq 0 \end{array}$$

Where  $X, C, A_k$  are matrices in  $\mathbb{R}^n$ . The similarity to linear programs is apparent, with the exception that instead of element-wise vector multiplication, there is element-wise matrix multiplication, or multiple vectors being multiplied in a single operation. From these, we can construct a linear program, demonstrating how linear programs are special cases of semidefinite programs. Let  $C, A_k \forall k$  be strictly diagonal matrices. That is,  $C_{ij} = 0 \quad \forall i \neq j$  and  $A_{k,ij} = 0 \quad \forall i \neq j$ . This will leave us with the only interesting values for each constraint and for the objective function being where  $i = j$ , so we can identify each component simply by one identifier. This will result in  $\sum_i c_i x_i$  for the objective function, and a similar looking sum for each constraint. This is simply a linear function.

# Restrictions on $X$

The variable matrix  $X$  must be a positive semidefinite matrix, which is how this type of program gets its name.

## Positive Semidefinite

A positive semidefinite matrix  $X$  is any matrix such that  $v^T \bullet X \bullet v \geq 0$  for any vector  $v$  in  $\mathbb{R}^n$ . A result of this particular definition is that the eigenvalues for the matrix  $X$  are all non-negative. If the eigenvalues are all positive, then the matrix is positive definite and  $v^T \bullet X \bullet v > 0$ . The constraint given to the matrix that any  $v$  will result in a non-negative value provides the program with the ability to linearly constrain a non-linear function. This will be demonstrated later in the geometries section. Note that any diagonally dominant matrix with positive values down the main diagonal is a positive semidefinite matrix, which can be helpful when attempting to find these matrices.

## Solving the SDP

Solvers use interior point methods to solve semidefinite programs. Even though all of the matrices in the constraint functions are linear matrices, the simplex method does not work for solving semidefinite programs. This is because the definition of a positive semidefinite matrix is  $v^T X v \geq 0$  for all  $v$ . The restriction that it must hold for all  $v$  implies an infinite number of constraints, which will be demonstrated below. Because of this, the simplex method does not work because it can take up to an infinite number of steps, which is computationally expensive to say the least. Therefore, interior point methods are used to solve SDPs because we can guarantee convergence to an optimal vertex within machine precision of the true optimal vertex. Because semidefinite programs are solved on computers, as interior point methods for problems large enough to be interesting would not be feasibly solvable by hand, machine precision near the optimal vertex is all we would get from a simplex method. Therefore, interior point methods are the preferred solving technique for SDPs.

## Geometry of the SDP

The feasible set of a SDP is an affine subspace intersected with the cone of semidefinite matrices in  $\mathbb{R}^{n \times n}$ , or  $\mathbb{S}^n$ . The affine subspace is the subspace in  $\mathbb{R}^{n \times n}$  that contains all each linear constraint after the element-wise operation  $A \cdot X$  takes place. The semidefinite cone is the collection of all positive semidefinite matrices, and the intersection of these two subspaces creates another affine subspace in  $\mathbb{S}^n$ .

## Difference from Linear Programming

In linear programming, each constraint generates a bisecting hyperplane, separating the vector space into two half-spaces. One of these half spaces contains feasible points for the constraint and the other does not. The same is true for semidefinite programs. The main difference is that there are a finite number of linear constraints in a linear program, and an infinite number of linear constraints in a semidefinite program. This is because a matrix is positive semidefinite if  $v^T X v \succeq 0$  for all  $v$  in  $\mathbb{R}^n$ . because there are an infinite number of constraints. take the example below:

Let  $X$  be the matrix  $\begin{bmatrix} x_1 & 1 \\ 1 & x_2 \end{bmatrix}$  and the vector  $v$  be  $\begin{bmatrix} 1 \\ a \end{bmatrix}$ . Then using every value of  $a$  provides a scalar of every possible vector in  $\mathbb{R}^2$  except any scalar of the vector  $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ .

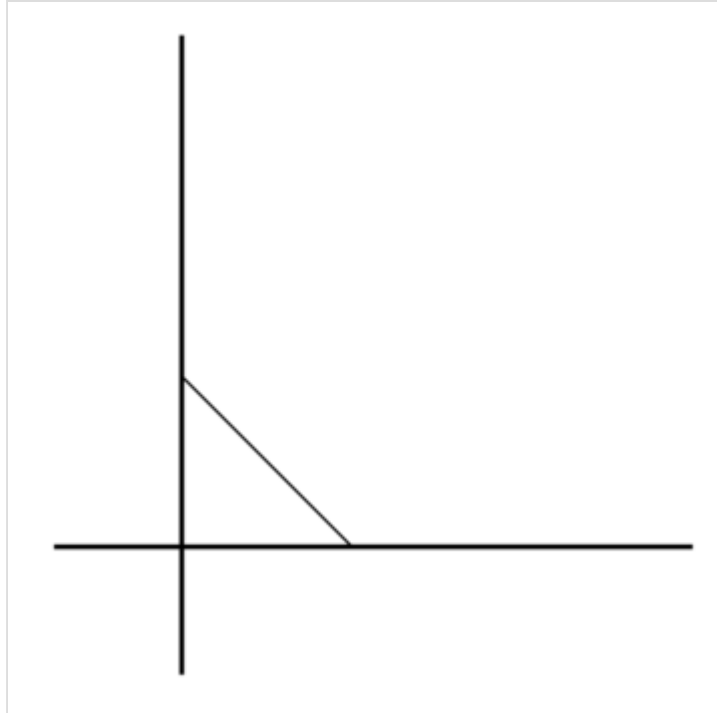
This is what we need to test if the matrix is positive semidefinite. Doing the matrix multiplication for  $v = \begin{bmatrix} 0 \\ a \end{bmatrix}$ ,  $v^T X v = a^2 x_2$ , which is always positive. For all other

cases, where  $v$  is some scalar of  $\begin{bmatrix} 1 \\ a \end{bmatrix}$ , the multiplication gives us

$$\begin{aligned} 0 \leq v^T X v &= v^T \begin{bmatrix} x_1 & 1 \\ 1 & x_2 \end{bmatrix} \begin{bmatrix} 1 \\ a \end{bmatrix} \\ &= v^T \begin{bmatrix} x_1 + a \\ 1 + a \cdot x_2 \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ a \end{bmatrix}^T \begin{bmatrix} x_1 + a \\ 1 + a \cdot x_2 \end{bmatrix} \\ &= (x_1 + a) + (a + a^2 x_2). \end{aligned}$$

Which gives us the constraint  $\frac{1}{a} x_1 + a x_2 \geq -2$ .

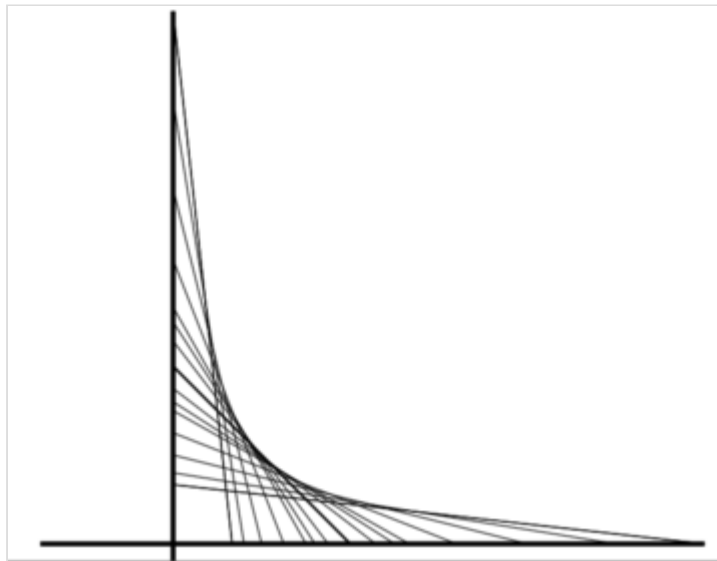
Plugging in different values for  $a$  gives different constraints. For example, if  $a$  is positive, then the constraints are completely contained in the part of the vector space outside the first orthant. This makes them redundant constraints and therefore not interesting. When  $a$  is negative, however, we get constraints that pass through the first orthant. for example, let  $a = -1$ . We end up with the constraint  $x_1 + x_2 \geq 2$ , which looks like this:



When  $a = -3, -2, -\frac{1}{2}, -\frac{1}{3}$ , then we get constraints that look like this:



And when we substitute in more real values for  $a$ , we can see that the set of linear constraints approaches the function  $x_2 = \frac{1}{x_1}$ , which represents a non-linear function.



This is also an example of how non-linear programs can be expressed as a semidefinite program.

We can also characterize positive semidefinite matrices by their **leading principal minors**:  $M$  is a positive semidefinite matrix if and only if all leading principle minors are non-negative. As the  $k$ th leading principle minor is the determinant of the first  $k$  rows and columns, we have the following two inequalities:

$$x_1 \geq 0,$$

and

$$\begin{vmatrix} x_1 & 1 \\ 1 & x_2 \end{vmatrix} \geq 0.$$

In other words, we need  $x_1 \geq 0$  and  $x_1 x_2 - 1 \geq 0$ , implying that the feasible region for our SDP is in fact  $x_2 \geq \frac{1}{x_1}$  for non-negative  $x_1$ , as expected.

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Semidefinite\_Programming&oldid=729"

- This page was last modified on 5 December 2017, at 14:48.
- This page has been accessed 9,103 times.

# Separable Convex Cost Network Flow Problems

From CU Denver Optimization Student Wiki

This page was created by Eric Olberding and Valentinas Sungaila.

The PDF of the presentation can be found at <https://github.com/eric072891/NetworkFlowsProject>.

In a minimum cost network flow problem, we have a graph with nodes and arcs. We can send things along the arcs from one node to another. Typically, there are some constraints on how much can be sent from or to each node. Sending something along an arc costs something. Separable convex cost network flow problems are minimum cost network flow problems where the cost is convex.

In this project, applications of convex cost flow problems are covered along with algorithms to solve some special cases of the problem. We describe two different types of convex cost functions: piecewise linear, and concise functions. Then, two algorithms for solving linear cost network flow problems are adapted to solve the piecewise linear cost network flow problems. The approximation of a concise convex function with a piecewise linear function and the modification of an algorithm to solve this problem is also covered.

## Contents

- 1 Separable Cost
- 2 Convex Cost
- 3 Applications
  - 3.1 Road Networks
  - 3.2 Area Transfers in Communication Networks
- 4 Two Types of Convex Cost Functions
- 5 Network Transformation for Piecewise Linear Cost Functions
- 6 Residual Network
- 7 Pseudopolynomial Time Algorithms
- 8 Polynomial Time Algorithm (Capacity Scaling Algorithm)
- 9 Reference

## Separable Cost

A separable cost problem is one in which the total cost can be written as a linear combination of the costs on each arc.  $\sum_{(i,j) \in A} C_{ij}(x_{ij})$

The following is an example of a network that does not have separable cost. Let the network have two arcs  $x_1, x_2$ . The total cost in this network is  $x_1 + 2x_1x_2 + x_2$ . This

Typesetting math: 100%

es not have separable cost.



# Convex Cost

In a separable convex cost setting, the cost of sending  $x_{ij}$  units of flow along an arc is  $C_{ij}(x_{ij})$ . A function is convex if  $f(\lambda a + (1 - \lambda)b) \leq \lambda f(a) + (1 - \lambda)f(b)$  for  $\lambda \in (0, 1)$ . This means that any line connecting two points on the graph of the function lay above the function. The function  $C_{ij}(x)$  must satisfy this, as it is convex. If we have the set of arcs  $A$  in the network, the total cost for a given flow in the network is  $\sum_{(i,j) \in A} C_{ij}(x_{ij})$ .

## Applications

### Road Networks

In road networks, as the number of cars increases, the road becomes more congested which leads to delay in travel time. This can be modeled as a function of the flow  $x$  on that road.

$$Delay = \frac{\alpha x}{(u - x)}$$

Where  $x$  is the flow of traffic,  $u$  is the theoretical road capacity, and  $\alpha$  is a constant. As  $x$  which is the flow increases and approaches  $u$  the delay on the road increases which makes the delay function on each road segment a convex function. When looking for the flow plan that gives the minimum overall delay in traffic flow, the sum of all the delays in road segments creates a convex cost network flow model.

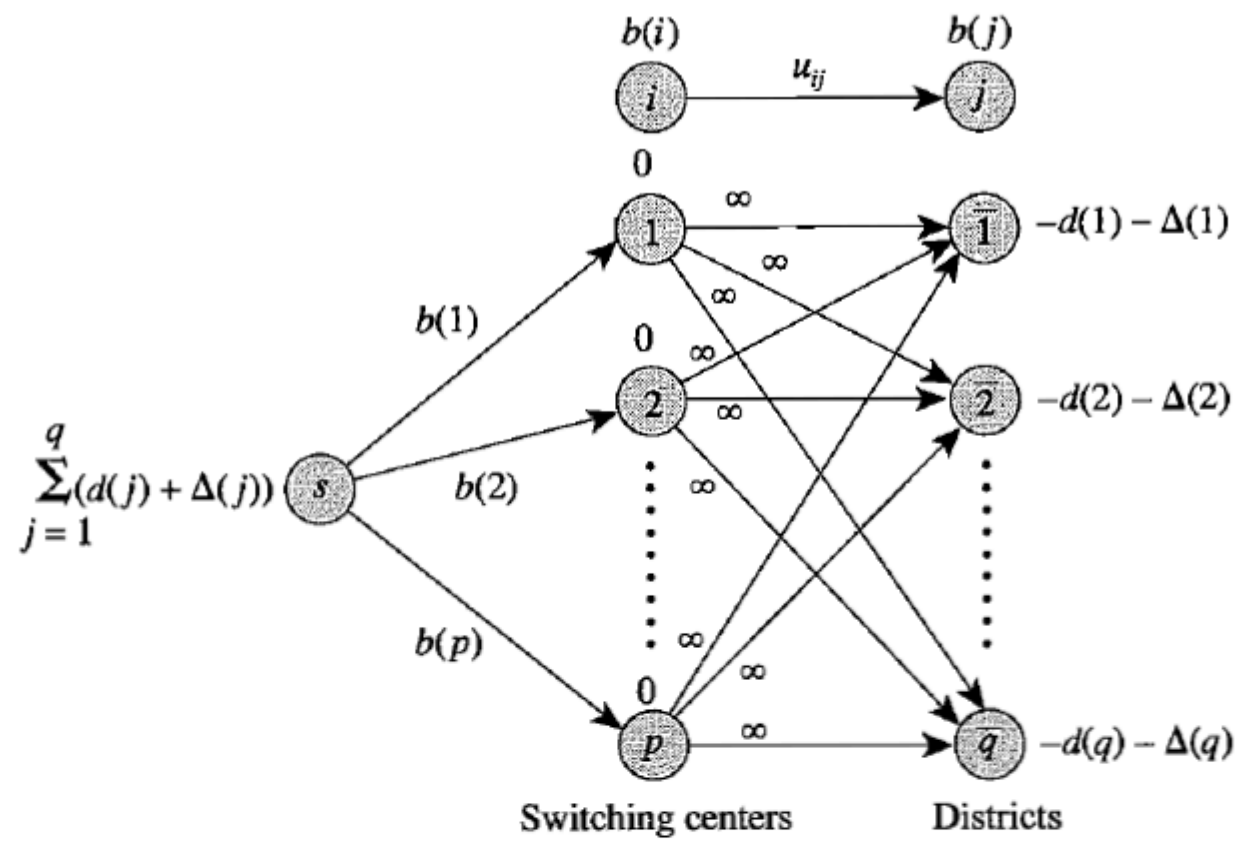
Another example of a convex cost network flow model for traffic flow deals with the behavioral assumption that people on the road system will travel from their starting point to their destination with minimum delay. Here delay can be represented as  $C_{ij}(x_{ij})$  on arc  $(i, j)$  as a function of arc's flow  $x_{ij}$  for each person on the road. Since multiple people are on the road and each persons flow depends on others the objective function is shown below.

$$\sum_{(i,j) \in A} \int_0 C_{ij}(y) dy$$

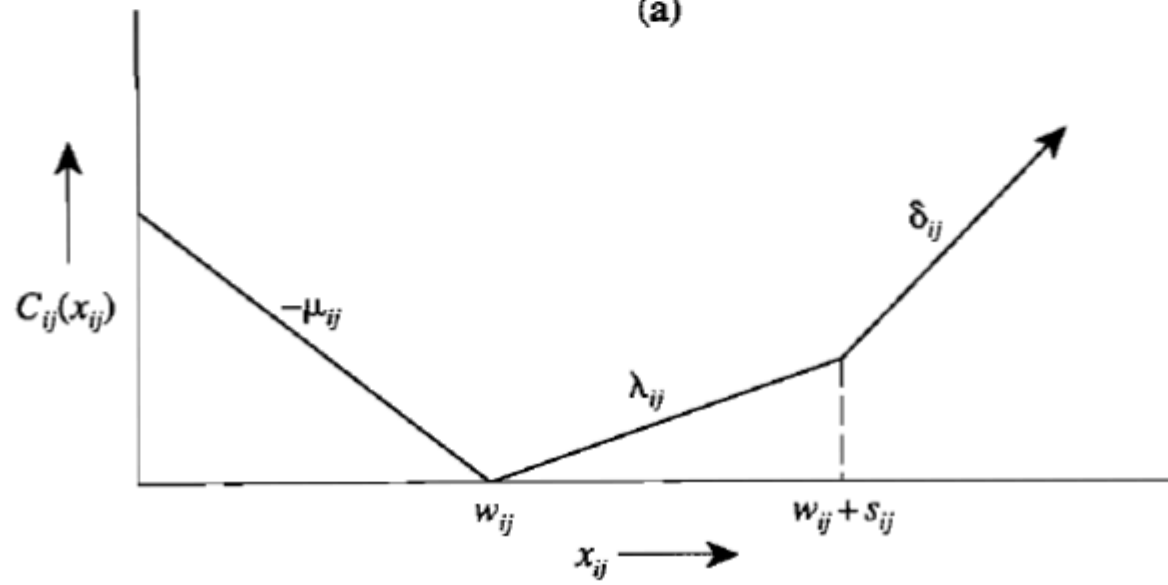
If the delay function is non-decreasing, then the function of each variable  $x_{ij}$  within the summation is convex. Each person will travel along a shortest path with total delay cost  $C_{ij}(x_{ij})$  on the arcs of the path which leads to a convex cost.

### Area Transfers in Communication Networks

In the past equipping every phone with a routing ability was expensive. Thus requiring switch centers to be used when routing phone calls. A convex cost network flow problem arises to see how many customers need to be assigned to switch centers to minimize cost. Area transfers in communication networks can route calls between people in a switching center and between switching centers as shown bellow.



(a)



(b)

Here  $d(j)$  represents the current demand of district  $j$  or the number of lines going to a switching center.  $b(i)$  represents the capacity of switch center  $i$  or the number of lines the center can handle.  $w_{ij}$  represents the number of lines currently in use between center  $i$  and district  $j$ .  $s_{ij}$  represents the spare lines connecting to the switch center  $i$  to district  $j$  at a cost of  $\lambda_{ij}$  per line.  $\delta_{ij}$  represents a cost for an additional line beyond the spare ones and the assumption is  $\delta_{ij} > \lambda_{ij}$ .  $\mu_{ij}$  represents the cost to disconnect a line from switch center  $i$  to district  $j$  and connect it to another switch center.

A demand for lines at each district can be defined as  $d(j) + \delta(j)$ . Figure **a**, shows how this communication network would look like. Figure **b** shows the cost of flow on arc  $(i, j)$ . If the switch center  $i$  supplies  $w_{ij}$  lines to district  $j$  then no cost is incurred. But when demand starts exceeding  $w_{ij}$  then the cost increases in a convex fashion first with  $\lambda_{ij}$  and then  $\delta_{ij}$ .

## Two Types of Convex Cost Functions

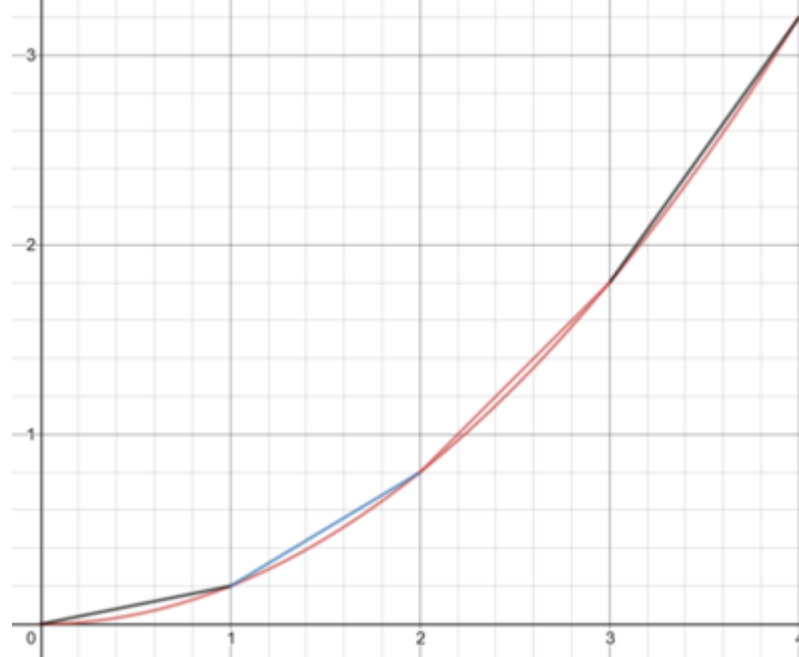
In this project we cover two types of convex functions. We cover concise functions, like  $C_{ij}(x_{ij}) = 10x_{ij}^2$ , that can be written using  $O(1)$  pieces of information.

We also examine piecewise linear costs. In these problems, the cost per unit (slope of the total cost function) for a single arc will be piecewise constan. Suppose we have an arc with a capacity of 2, where a different cost is incurred for each unit of flow. The first unit of flow along this arc may incur a cost of 5, while the second unit of flow incurs a cost of 7. This would be written,

$$C_{ij}(x_{ij}) = \begin{cases} 5x, & \text{if } 0 \leq x_{ij} < 1 \\ 7x + 5, & \text{if } 1 \leq x_{ij} \leq 2 \end{cases}$$

Note that, as these piecewise linear functions are convex, the cost of sending more flow along an arc can only ever increase.

Concise functions can be approximated with piecewise linear functions. Below, we show an approximation of the concise function  $C(x) = \frac{1}{5}x^2$  with 4 linear segments.



If we assume that the feasible solutions to a separable convex cost problem are integral, then we lose nothing by approximating the concave function with a piecewise linear convex cost function. Usually, the integer optimal solution will be worse than the continuous optimal solution. However, this isn't a major issue. We can approximate the continuous optimal solution to any desired degree.

To do this, we replace  $x_{ij}$  with  $y_{ij}/M$ , where  $M$  is some large integer of our choosing. Then, for an integer optimal solution  $y_{ij}^*$ , the solution  $x_{ij}^* = y_{ij}^*/M$  is within  $1/M$  of the optimal flow for the original continuous problem. We are "stretching" the function. Integer solutions in the new function correspond to fractional solutions in the original problem.

This approximation does have implications for the running time of algorithms, as we outline in the next section.

## Network Transformation for Piecewise Linear Cost Functions

One way of solving separable piecewise linear convex cost problems, is to convert the corresponding network into one that can be solved with typical linear minimum cost flow problems. For an arc from  $i$  to  $j$  with cost function consisting of  $p$  linear segments, we replace it with  $p$  arcs between  $i$  and  $j$ . Note that, for a piecewise linear cost function, the cost per unit of extra flow corresponds to the slope of the line segment. We now have  $p$  arcs, where the cost on each arc corresponds to the slope of the line segment. To illustrate with an example, suppose we have an arc with capacity 4 such that the cost per unit flow is  $c_1$  on  $[0, 1)$ ,  $c_2$  on  $[1, 2)$ , and  $c_3, c_4$  similar. Suppose that we send 2 units of flow along the arc. The transformed network looks like the following.



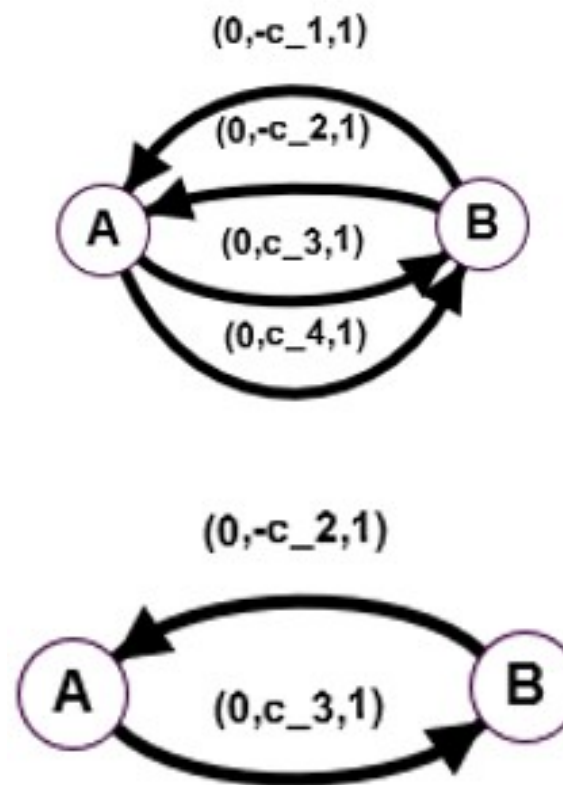
For the first unit of flow, we completely traverse the line segment in the cost function with slope  $c_1$ . This corresponds to sending one unit along the arc with cost  $c_1$ . We send another unit of flow, completely traversing the line segment with cost per unit (slope)  $c_2$ . Again, the arc corresponding to this line segment is at capacity.

For convex cost functions that were already piecewise, this transformation does not seriously affect running time. For each arc with  $p$  segments, we already needed to specify information proportional to  $p$ . For concise functions, however, we add information that isn't implicit in the problem setup. The better the approximation we use, the more line segments we have. This means more added arcs and more information to store. Furthermore, if we analyze the running time of an algorithm on the transformed network for a concise function, the running time will depend on the number of pieces; these are independent of the original data.

We also note that we have a graph with multi-arcs! If we are searching for a shortest path, we may search through arcs that we are guaranteed not to use in an augmenting step. An augmenting step involves sending extra flow along an arc. Since the cost function is convex, the slope of the cost function can only increase. This means that  $c_1 < c_2 < c_3 < c_4$  in the example above. If we have currently sent two units of flow along this arc, we would only ever send an additional unit of flow along the arc with cost  $c_3$ . Thus, we need to somehow disregard the arc with cost  $c_4$  until it is needed. To do this, we use a modification of a residual network.

## Residual Network

For the transformed network above, we can construct a modification of the typical residual network that saves computation. We use the fact that, for a given level of flow  $x_{ij}$ , we only need the arcs with costs equal to the slopes of the line segment from one less than the current flow to the current flow and the line segment between one more than the current flow and the current flow. For the forward arc  $(i, j)$ , we have the arc with cost  $C_{ij}(x_{ij} + 1) - C_{ij}(x_{ij})$ . Note that we are using the total cost here, so the difference corresponds to the slope. Similarly, we include the backwards arc  $(j, i)$  with cost  $C_{ij}(x_{ij} - 1) - C_{ij}(x_{ij})$ . The capacities of these arcs corresponds to the amount of flow we can send forward or back before needing to change the cost per unit of the arc. The picture below contains the normal residual network for the example in the previous section and the modified residual network. In that example, we had already sent two units of flow.



We only need the arcs that correspond to spending the least when sending flow and saving the most when deciding to no longer send flow.

## Pseudopolynomial Time Algorithms

Once we have the modified residual network above, we can use typical linear min cost flow algorithms to solve our problem. For the cycle-canceling algorithm, after finding a feasible flow we search for a negative cycle in the modified residual network. We then augment flow along this negative cycle to reduce the total cost in the original network. Note that this modified residual network will have arcs with costs and capacities that change as we augment flow. Because of this, a cycle may still be negative even after augmenting as much flow as possible along that cycle. If the cycle is still negative, it would be a waste to search for a negative cycle again. To avoid this, we continue to augment flow along the same negative cycle until it is no longer negative.

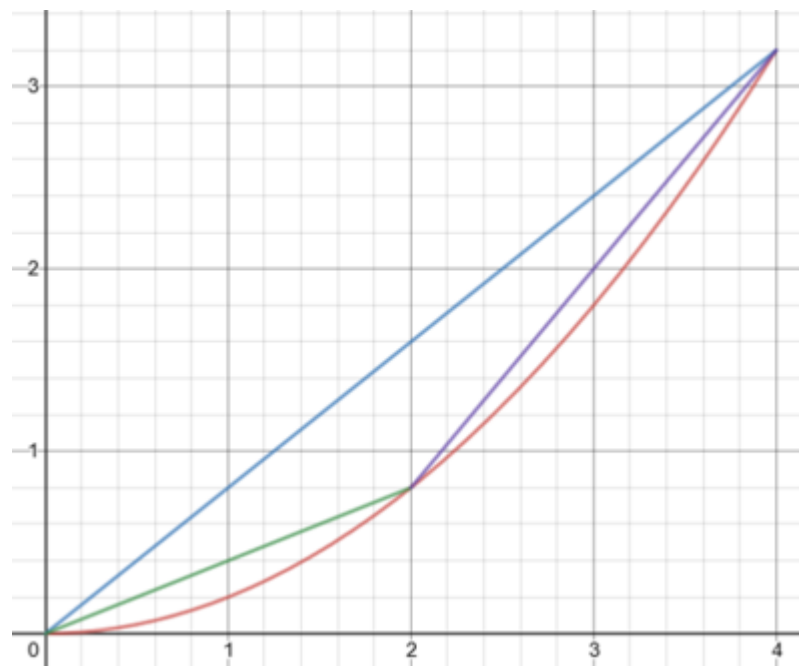
The shortest successive paths algorithm can similarly be applied to the modified residual network. Similar to before, we may end up augmenting flow along the same path multiple times, as the capacity of an arc is "reset" when increasing the cost of the arc in the augmenting path. In fact, in the worst case, this algorithm may only augment 1 unit of flow per iteration.

# Polynomial Time Algorithm (Capacity Scaling Algorithm)

In this section, we describe an improvement upon the shortest successive paths algorithm mentioned above. We describe a modification that prevents sending small increments of flow repeatedly. The main idea is to start with course approximations of the concise function and push large amounts of excess to deficit nodes and then refine the approximation over the course of the algorithm while sending smaller and smaller amounts. We first approximate the concise function with one linear segment. The maximum capacity of this segment is a power of two. It is the largest power of two less than or equal to the original maximum capacity of the arc. Let  $\Delta$  be the size of the interval on which one of the linear segments is defined. For the different approximations, we have corresponding residual networks  $G(x, \Delta)$  defined in the residual networks section.

We note above that, in the first phase,  $\Delta$  will be the largest power of two less than or equal to the maximum capacity. We initialize the node potential and flow to be 0. Now, define  $E(\Delta), D(\Delta)$  as the nodes with excess at least  $\Delta$  and at most  $-\Delta$ , respectively. Choose a pair of nodes from these sets and find the shortest path between them. Push  $\Delta$  units of flow from the excess node to the deficit node. Update the residual network. Do this until either  $E(\Delta)$  or  $D(\Delta)$  are empty.

Once they are empty, replace  $\Delta$  with  $\Delta/2$  and repeat the process. At the start of each scaling phase  $\Delta$ , we may have to add or subtract  $\Delta$  units of flow to an arc to make the arc satisfy the reduced cost optimality conditions. In the graph below, if we start the phase  $\Delta = 2$  with a flow of 0, the slope of the adjacent line segment,  $c_{ij}$  is smaller. This means that the reduced cost for  $(i, j)$ ,  $c_{ij}^\pi = c_{ij} - \pi(i) + \pi(j)$  is smaller and may be negative. We also see that if we increase the flow by  $\Delta = 2$ , the cost per unit flow on the forward arc  $(i, j)$  (slope of the line segment to the right of  $x_{ij} = 2$ ) is larger and thus the arc satisfies the reduced cost optimality conditions.



Within each scaling phase, the reduced cost optimality condition is preserved as we are essentially using the successive shortest paths algorithm. This means that, after we have scaled  $\Delta$  down to one and gotten rid of all excess, we will have an optimal solution.

# Reference

Ravindra Ahuja, Thomas Magnanti, and James Orlin, Network Flows, MIT Press (1993)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Separable\\_Convex\\_Cost\\_Network\\_Flow\\_Problems&oldid=2900](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Separable_Convex_Cost_Network_Flow_Problems&oldid=2900)"

---

- This page was last modified on 7 May 2020, at 13:59.
- This page has been accessed 3,967 times.



# Shadow price

From CU Denver Optimization Student Wiki

In mathematical programming, **shadow prices** are a measure of the sensitivity of a program with respect to a constraint. This sensitivity the relative change in the objective function value when the right hand side value of a constraint is changed. Consider the following linear example:

Friction Tire Company makes 2 kinds of tires, a sport tire and an all season tire. They sell the sport tire for \$50 per tire and the all season tire for \$40 per tire to their various distributors. Due to constraints on the size of their warehouse, they can only store up to 1,000 kg of their main rubber compound each week before they get their next shipment. Likewise, they can only have up to 700 kg of their all season compound and 1,100 kg of their sport compound. Each sport tire takes one kg of the main compound and 2 kg of the sport compound to make, and each all season tire takes one kg of the main compound and one kg of the all season compound. They know that they will need to produce at least 300 of each tire, and they can rely on their marketing department to create demand for whatever else they produce. Before Friction can find the shadow prices, they need to model and solve this program.

$$\begin{array}{llllll} \text{max} & 40x_1 & + & 50x_2 & & \\ \text{s.t} & x_1 & + & x_2 & \leq & 1,000 \\ & x_1 & & & \leq & 700 \\ & & & 2x_2 & \leq & 1,100 \\ & x_1 & , & x_2 & \geq & 300 \end{array}$$

It is easy to verify that the optimal point is (450, 550), or that Friction should produce 550 sport tires and 450 all season tires to maximize the profit. That maximized profit will be \$45,500.

With the model and an optimal point, the shadow prices can be found, as well as how they affect the objective function value. These are calculated component-wise, or one constraint at a time. Increasing or decreasing the right hand side (RHS) value of each component one marginal unit at a time and then re-solving the program will give you the new objective function value. The difference between these two values, or the change in objective function value, is the shadow price. Imagine that Friction Tire could store 1,100 kg of their main rubber compound instead of 1,000 and they have to take shipments in 100 kg increments, what would the difference in profit be? Start with the first constraint and increase the RHS value to 1,100. Solving the new model gives an objective function value of 49,500. This gives an increased profit of \$4,000 per 100 kg of increased storage. Because this example is linear, the shadow price for the primal problem on constraint 1 is  $\frac{4,000}{100}$ , or 40. This means that for each increased unit on the right hand side of the first constraint, the objective function value will increase by 40.

This is relatively simple for linear problems, but it gets a bit more complex for non-linear problems. Non-linear programs require the partial derivative, or gradient ( $\nabla f$ ), of the objective function value with respect to the constraint to find the instantaneous shadow price. This should make sense for anyone with a background in calculus, as derivatives simply describe rates of change. The simplified way of finding the shadow price ( $\lambda$ ) in this way is with the formula:

$$\lambda = \frac{\partial f}{\partial g}$$

This is obtained with the Lagrangian, which is explained in more depth on its own page. The gradient of the objective function with respect to the constraints is a vector equal to the optimal solution of the dual problem for linear programs. The same is true for the gradient of the dual program.<sup>[1]</sup><sup>[2]</sup>

## Referances

- ↑ Rardin, Ronald L. Indtroduction to Optimization in Operations Research. 2nd ed., Pearson, 2016
- ↑ Vanderbei, Robert J. Linear programming. Springer US, 2014. International Series in Operations Research & Management Science 196. Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Shadow\_price&oldid=247"

- This page was last modified on 21 March 2017, at 13:40.
- This page has been accessed 12,084 times.

# Shortest Path Routing Algorithms

From CU Denver Optimization Student Wiki

I have worked on the single source shortest path routing algorithm and all pair shortest path routing algorithm.

## Abstract

The shortest path algorithms have many important applications ranging from Google Map, network routing protocols, VLSI chip placement to prediction of infection risk and exploiting arbitrage opportunities in currency exchange. The purpose of this project is to study the single source shortest path algorithm (due to Dijkstra) and the all pair shortest path routing algorithm (due to Floyd and Warshall) in a graph. Both the algorithms assume that the graph is connected, the network topology and edge costs are known and nodes perform computation independent of each other. Dijkstra's algorithm assumes that there is no negative edge cost while the Floyd Warshall's algorithm assumes that there is no negative edge cycle. Dijkstra's algorithm iteratively finds the shortest path by updating the estimated distance of the unvisited nodes that are connected to the newly discovered node. This is a greedy algorithm which uses two key observations: (a) the subpath of any shortest path is itself a shortest path; (b) the triangle inequality can lead to a better distance estimate. The running time of the algorithm is  $O(V^2)$  ( $V$ : the number of nodes) and can be improved to  $O(V \log V)$  by using a Fibonacci heap. Floyd Warshall's algorithm is a dynamic programming approach in which the intermediate results are stored in a table. This uses the following key observation: for a shortest path between  $i$  and  $j$ , there are two cases. In the first case, the intermediate node is not on the path leading to the shortest path of length  $d_{ij}^{(m-1)}$  whereas in the second case, the intermediate node is on the path resulting in the shortest path of length  $d_{im}^{(m-1)} + d_{mj}^{(m-1)}$ . The algorithm runs in  $O(V^3)$ . While Dijkstra's algorithm can be invoked for every node to obtain the all pair shortest path, the algorithm will not accept a negative weight.

## Github link

<https://github.com/amitsengupta1/shortest-path> Amit Sengupta

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Shortest\_Path\_Routing\_Algorithms&oldid=2624"

- This page was last modified on 1 May 2020, at 08:42.
- This page has been accessed 1,159 times.

# Shuo Feng

From CU Denver Optimization Student Wiki

Shuo is a graduate student at the University of Colorado Denver studying electrical engineering. Originally from China, completed his undergraduate education with a bachelor degree in biomedical engineering and attended University of Colorado Denver since 2014. During his studies at CU Denver, he is working with professor Jae-Do Park on the microbial fuel cells (MFCs) project, focusing on MFCs voltage optimization and energy harvesting.

- Contributions

Solving Power Flows Problems Using Network Flows Model

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Shuo\\_Feng&oldid=1131](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Shuo_Feng&oldid=1131)"

Category: Contributors

- 
- This page was last modified on 22 April 2018, at 13:30.
  - This page has been accessed 1,001 times.

# Siyuan Lin

From CU Denver Optimization Student Wiki

I am a first year master student in Applied Mathematics. I come from Shandong, China. Since my undergraduate major was Economics, so I also hold a master degree in Applied Economics from Georgetown University. I am working with Gregory Matesi. The link to our project page is: Gradient Descent Method in Solving Convex Optimization Problems

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Siyuan\\_Lin&oldid=2283](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Siyuan_Lin&oldid=2283)"

Category: Contributors

- 
- This page was last modified on 28 November 2019, at 10:57.
  - This page has been accessed 1,311 times.

# Sollin's Algorithm for Minimum Spanning Trees

From CU Denver Optimization Student Wiki

The author of this project is Rebecca Robinson.

## Contents

- 1 Introduction
- 2 Optimality Conditions
- 3 History
- 4 Algorithm
- 5 Pseudocode
- 6 Example
- 7 Complexity of the Algorithm
- 8 Correctness of Algorithm
- 9 Presentation Slides
- 10 References

## Introduction

In a network composed of nodes and arcs with costs, a **spanning tree** is a acyclic subgraph connecting all the nodes together. A **minimum spanning tree** is the spanning tree with minimum cost on the network. That is it is the spanning tree with the least sum of the costs of all the edges.

Finding the minimum spanning tree is useful in several different applications. Perhaps the most direct application is designing physical systems. For example, consider isolated villages that are connected by roads, but not yet by telephone lines. We want to decide where to put the telephone lines to be able to reach every village with using the minimum length of telephone line.

Another application of the minimum spanning tree problem is cluster analysis. Suppose that we have a network and want to split the network into  $k$  different clusters such that the total cost of all the clusters is minimized. We can take the minimum spanning tree and delete the  $k - 1$  arcs with the highest cost. The result is a forest of  $k$  trees with minimal cost.

## Optimality Conditions

For the minimum spanning tree problem, we have two different optimality conditions: *cut optimality conditions* and *path optimality conditions*. We do not show it here, but these two optimality conditions are equivalent. We describe the cut optimality conditions here.

## Cut Optimality Conditions

A spanning tree  $T^*$  is a minimum spanning tree if and only if it satisfies the following cut optimality conditions: For every arc  $(i, j) \in T^*$ ,  $c_{ij} \leq c_{kl}$  for every arc  $(k, l)$  contained in the cut formed by deleting arc  $(i, j)$  from  $T^*$ .

**Proof.** Suppose  $T^*$  is a minimum spanning tree and that  $c_{ij} > c_{kl}$  and arc  $(k, l)$  is in the cut formed by deleting arc  $(i, j)$  from  $T^*$ . Then  $T^*$  is not a minimum spanning tree since replacing  $(i, j)$  by  $(k, l)$  in  $T^*$  would yield a lower cost spanning tree.

Suppose  $T^*$  satisfies the cut optimality conditions, but  $T^*$  is not a minimum spanning tree. That is, there exists a tree  $T' \neq T^*$  that is a minimum spanning tree. Then, since  $T^*$  is not a minimum spanning tree, it must have some arc  $(i, j)$  not contained in  $T'$ . Deleting arc  $(i, j)$  from  $T^*$  creates a cut  $[S, \bar{S}]$ . We note that in a tree, if any arc is added between the nodes of the tree, it must form a cycle. Thus, if we add the arc  $(i, j)$  in to  $T'$ , we must form a cycle  $W$ .  $W$  must contain another arc  $(k, l)$  such that  $k \in S$  and  $l \in \bar{S}$ . Since  $T^*$  satisfies the cut optimality conditions,  $c_{ij} \leq c_{kl}$ . Since  $T'$  is a minimum spanning tree,  $c_{ij} \geq c_{kl}$ , otherwise  $T'$  would have contained  $(i, j)$  to begin with. Thus,  $c_{ij} = c_{kl}$ . If we replace  $(k, l)$  by  $(i, j)$  in  $T^*$ , we produce a minimum spanning tree with one more arc in common with  $T'$ . We can continue to repeat this argument to transform  $T^*$  into  $T'$ , giving that  $T^*$  is a minimum spanning tree. Q.E.D.

Note that the cut optimality conditions imply that every arc in a minimum spanning tree is a minimum cost arc across the cut defined by removing it from the tree. We can reword this by saying that for any cut, the minimum cost edge in the cut must be in the minimum spanning tree.

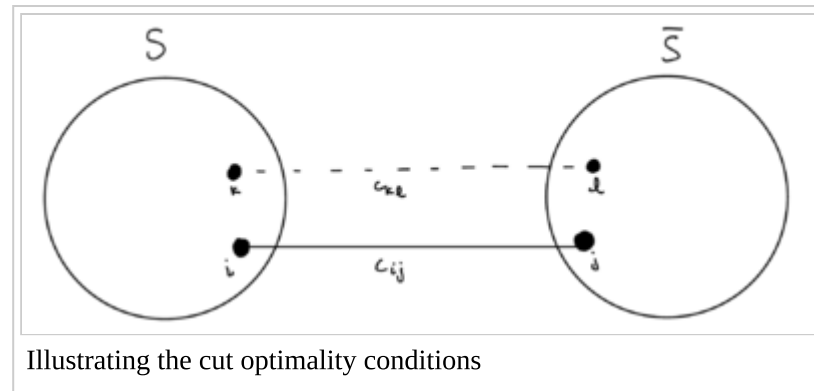
## History

More commonly known as Boruvka's algorithm, Sollin's algorithm was the first algorithm for the minimum spanning tree problem. Boruvka first published his algorithm in 1926 which he designed as a way of constructing an efficient electricity network in Moravia, a region of the Czech Republic. He published two papers in the same year with this algorithm. One was 22 pages and has been viewed as unnecessarily complicated. The other was a one page paper that has been regarded as the key paper showing Boruvka's understanding and knowledge of the problem.

It was then independently rediscovered in 1938 by French mathematician Gustave Choquet, and again in 1951 by Florek, Łukasiewicz, Perkal, Steinhaus, and Zubrzycki. Again, it was rediscovered in 1962 by Georges Sollin.

## Algorithm

Sollin's algorithm is a hybrid of Kruskal's and Prim's algorithm. In Sollin's algorithm, we maintain a collection of nodes  $N_1, N_2, \dots$  and adds arcs to this collection, a technique borrowed from Kruskal's algorithm. We also add minimum cost arcs at every iteration, a technique borrowed from Prim. It's important to note that this algorithm requires all arc costs to be distinct, but Boruvka has said "If we measure distances, we can assume that they are all different. Whether distance from Brno to Breclaw is 50 km or 50 km and 1 cm is a matter of conjecture."



Boruka's one page paper

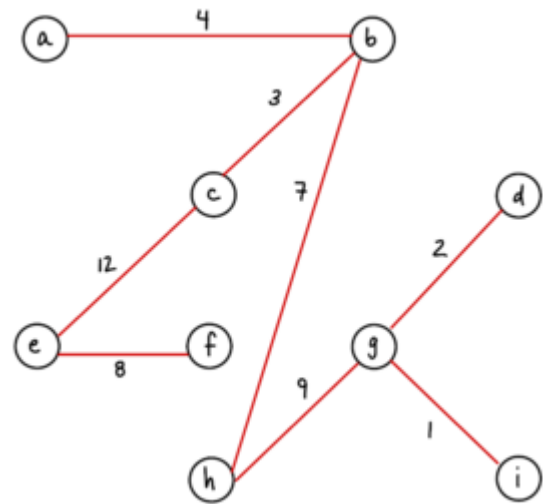
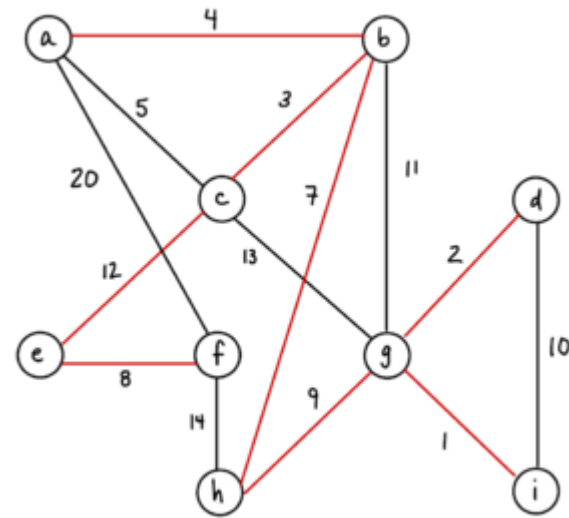
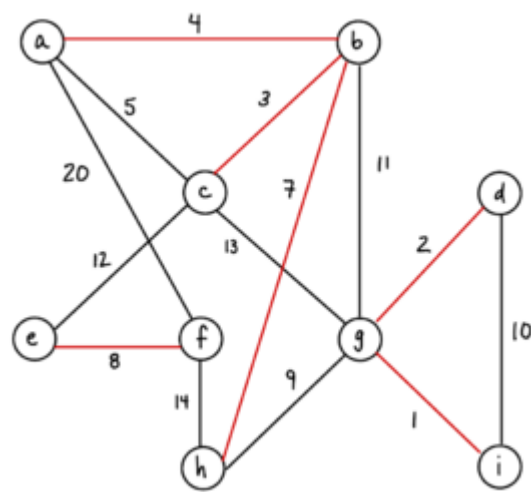
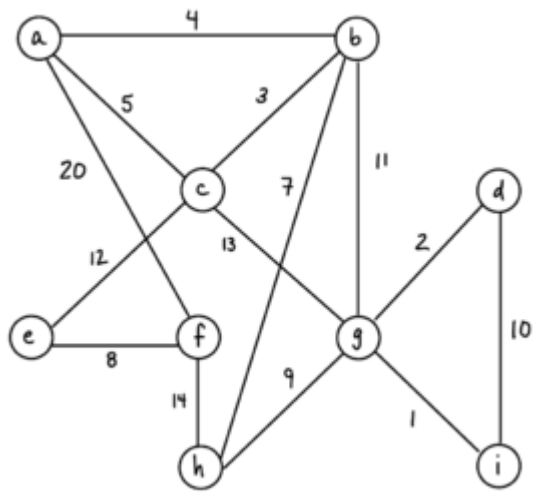
1. *nearest-neighbor*( $N_k, i_k, j_k$ ): Given a tree spanning the nodes  $N_k$ , this operation determines an arc  $(i_k, j_k)$  that is a minimum cost arc emanating from  $N_k$ . To perform this operation, we scan all the arcs in the adjacency lists of nodes in  $N_k$  and find a minimum cost among those that have one endpoint not in  $N_k$ .
2. *merge*( $i_k, j_k$ ): Given two nodes  $i_k$  and  $j_k$ , if the two nodes belong to two different trees, we merge the two trees into a single tree.

# Pseudocode

```
algorithm Sollin;
begin
  for each  $i \in N$  do  $N_i := \{i\}$ ;
   $T^* := \emptyset$ ;
  while  $|T^*| < (n - 1)$  do
    begin
      for each tree  $N_k$  do nearest-neighbor( $N_k, i_k, j_k$ );
      for each  $N_k$  do
        if nodes  $i_k$  and  $j_k$  belong to different trees then
          merge ( $i_k, j_k$ ) and update  $T^* := T^* \cup \{(i_k, j_k)\}$ ;
    end;
  end;
```



# Example



Consider the network given in the leftmost picture. The second picture gives the result after the first iteration of the algorithm.

The third picture gives the second iteration. Since after the second iteration, we only have one tree left, the algorithm terminates and we are left with the spanning tree shown on the far right with a cost of 46.

## Complexity of the Algorithm

We look at the running time of each of the two major operations that occur in Sollin's algorithm. We note that nodes are stored as a circular doubly linked list. This allows us to be able to visit each node of the tree starting from any other node. Each node gets a label such that:

1. Nodes of the same tree get the same label

Typesetting math: 100% nt trees have different labels

At the start, each label  $i$  is given to each  $i \in N$ . That is, each node is a separate tree. We can see that we check if an arc  $(i, j)$  has its endpoint in the same tree by checking the labels of  $i$  and  $j$ . Thus, *nearest-neighbor* can be performed in  $O(\sum_{i \in N} |A(i)|) = O(m)$ .

Merge operations are performed in an iterative scheme. In each iteration,

- We select an unexamined tree, say  $N_1$ . Let  $(i_1, j_1)$  be a minimum cost arc emanating from  $N_1$ .  $j_1$  may or may not be in  $N_1$ .
- Suppose the nodes in  $N_1$  have label  $\alpha$ .
- If node  $j_1$  has label  $\alpha$ , the iteration ends.
- Otherwise, scan through the nodes of the tree containing  $j_1$ , say  $N_2$ , and assign them label  $\alpha$ .
- Now consider a minimum cost arc  $(i_2, j_2)$  emanating from  $N_2$ .
- If node  $j_2$  has label  $\alpha$ , the iteration ends.
- Otherwise, scan through the nodes of the tree containing  $j_2$ , say  $N_3$  and assign them the label  $\alpha$ .
- Repeat this until eventually the iteration ends.

This procedure assigns a label to each node once, and so runs in  $O(n)$  time.

Each execution of the while loop lowers the number of trees by at least a factor of 2, which means we run the while loop  $O(\log n)$  times. This, together with the  $O(m)$  time for *nearest-neighbor* gives a total running time of  $O(m \log n)$ .

## Correctness of Algorithm

Recall that a different wording of the cut optimality conditions is that for any cut, the minimum cost edge in the cut must be in the minimum spanning tree. Let  $T^*$  be the spanning tree given by Sollin's algorithm, and let  $T$  be the minimum spanning tree. Assume that  $T^* \neq T$ . Then there is some arc  $(i, j)$  contained in  $T^*$  but not  $T$ . Delete  $(i, j)$  to obtain the cut  $[S, \bar{S}]$ . Since  $T$  is a spanning tree, it must have some arc  $(k, l) \notin T^*$  such that  $k \in S$  and  $l \in \bar{S}$ . By the algorithm,  $c_{kl} > c_{ij}$ , else  $(k, l)$  would have been chosen by the algorithm to be in  $T^*$ . But according to cut optimality conditions, the minimum spanning tree  $T$  should contain the minimum cost edge in any cut. But  $T$  does not contain  $(i, j)$  which is the minimum cost arc in  $[S, \bar{S}]$ . This is a contradiction, and so  $T^*$  given by the algorithm is the minimum spanning tree.

## Presentation Slides

Presentation slides can be accessed here: <https://github.com/rebrobin/presentation/blob/master/NetworkFlowsPresentationFinal.pdf>

## References

Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993).

Nesetril, J., Nesetrilova, H.: The Origins of Minimal Spanning Tree Algorithms – Boruvka and Jarnik. Documenta Mathematica. Extra Volume ISMP, 127–141 (2012). Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sollin%27s\\_Algorithm\\_for\\_Minimum\\_Spanning\\_Trees&oldid=2816](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Sollin%27s_Algorithm_for_Minimum_Spanning_Trees&oldid=2816)"

- This page was last modified on 5 May 2020, at 11:21.
- This page has been accessed 5,270 times.

# Solving Power Flows Problems Using Network Flows Model

From CU Denver Optimization Student Wiki

Power Grids (electricity networks) are responsible for our daily electrical energy usage. An electricity network usually includes multiple components like generators (producing energy), utilities (energy consumers), transmission lines (energy transportation), transformers, and some power electronics devices for control purposes. For electrical analysis, like demand satisfaction, costs optimization, an energy flow calculation is necessary. This is usually a non-linear optimization problem that is solved by numerical methods known as Newton-Raphson method, an optimal power flow (OPF) to calculate the optimal energy production for each generator in the grid. In this project, the goal is to investigate electricity networks under the network flows theory, finding out if the general network flows algorithms like maximum flows and minimum costs can have positive impact on optimal power flow calculations.

## Contents

- 1 Problem Simplification
- 2 14-Bus Power Grid Test System
  - 2.1 Per-Unit-System
  - 2.2 Power Flows Calculated by Numerical Methods
- 3 Power Grid Flow-Based Model
  - 3.1 Notes on Model
  - 3.2 Feasible Flows in Model
- 4 Results and Discussions
- 5 References

## Problem Simplification

The electrical flows in a power grid obey laws of physics, and typically people don't control them in the real network. The amount of the energy produced is forecasted by the previous year's data. To control the electrical flows in the electrical network, each node or some of the nodes have to be able to distribute the flow according to a given flow. Control devices, which are able to influence the electrical flow, in general, the flexible alternating current transmission systems (FACTS) are needed. In this project, electricity networks under the assumption that all of nodes are supplied with such a control device. And the test system used in the project is a 14 bus system which will be introduced in the next section.

## 14-Bus Power Grid Test System

The electricity network on the right is called a 14-bus system, since this electricity network has 14 buses.

The lines connecting the buses represent the transmission lines and are named *branches*. An arrow at a bus means that there is a power demand, consists of real power demand and reactive power demand, the power demand are also called *load*. Buses which are connected with a generator marked with G or a condenser marked with C represent the generator buses which are the *power supplies* in an electricity network. In AC a generator consists of both a real power output and a reactive power output and a condenser has only reactive power output. The specifications of this system can be find from the Washington University website<sup>[1]</sup>.

## Per-Unit-System

In practice, the high-voltage lines are commonly used in transmission networks for large distance transmissions and low-voltage lines are used in distribution network for short distance transmissions. Thus, in an electricity network there typically exist multiple voltage levels. To make the calculation in an electricity network with multiple voltage levels easier, the per-unit-system is used as a normalization. The system will be separate into different base zone but with the same normalization level in per unit system.

$$perunit = \frac{value_{real}}{value_{base}}$$

In this project the base value is set to 100 MVA.

## Power Flows Calculated by Numerical Methods

In an AC network power consists of two terms: the real power  $P$  (measured in MW) and reactive power  $Q$  (measured in MVar). When simulating an electricity network, then it is necessary to include Kirchhoff's laws. The first Kirchhoff's current law, defines that the sum over all incoming and outgoing currents at a bus are equal to zero. The second Kirchhoff's voltage law defines that the sum over all voltages in a loop are equal to zero.

$$V * (Y * V) - Sg + Sd = 0$$

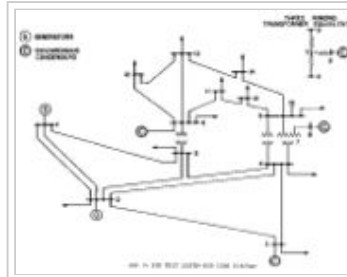
The real power demand  $Pd$  and reactive power demand  $Qd$  for each bus are given. For each generator bus in the electricity network, the power generation  $Pg$  and voltage  $V$  are given. And there is the slack bus, which we know the voltage magnitude  $V$  and voltage angle  $\theta$ . The goal is to calculate the voltage magnitudes  $V$  and voltage angles  $\theta$  at each load bus in the electricity network and voltage angles  $\theta$  for the generator buses so that the power demands are satisfied.

$$\begin{bmatrix} \frac{\partial P_1}{\partial \theta_1} & \frac{\partial P_1}{\partial V_1} & \cdots & \frac{\partial P_1}{\partial \theta_{n-1}} & \frac{\partial P_1}{\partial V_{n-1}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial P_{n-1}}{\partial \theta_1} & \frac{\partial P_{n-1}}{\partial V_1} & \cdots & \frac{\partial P_{n-1}}{\partial \theta_{n-1}} & \frac{\partial P_{n-1}}{\partial V_{n-1}} \end{bmatrix} \begin{bmatrix} \theta_2 \\ \vdots \\ \theta_n \\ V_2 \\ \vdots \\ V_n \end{bmatrix} = \begin{bmatrix} P_2 \\ \vdots \\ P_n \\ Q_2 \\ \vdots \\ Q_n \end{bmatrix}$$

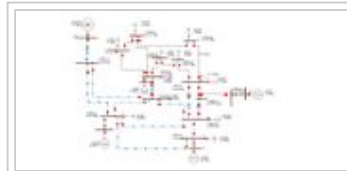
The linearized equation system is solved for the next iterations by using the next estimation for voltage magnitude  $V$  and voltage angle  $\theta$ . This iteration ends if the error lies in the tolerance  $\epsilon$ . Typical initial guesses are  $V = 1$  for voltage magnitude and  $\theta = 0$  for voltage angles. Further more, If there is a real power generation  $Pg$ , the cost function is defined by

$$C = 10 * k^2 / P_g + 20 * k$$

where  $k$  is the amount of generation.



14-bus system



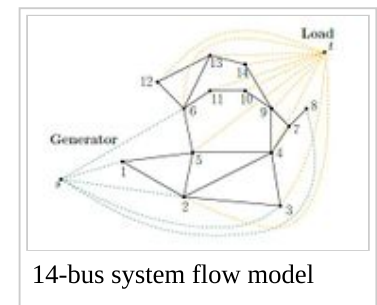
14-bus system in per unit

# Power Grid Flow-Based Model

To apply flow models on electricity networks it is necessary to transform these networks to s-t-networks with one source and one sink to generate flows. Therefore, we interpret the electricity networks with regard to graph theoretical terms and define the sources and sinks of these network structures to connect these sources and sinks to one general source s and general sink t. This transformation also simplifies the work in the graph theoretical area and provides clear mathematical descriptions. Buses connected to generators in  $G$ , which are responsible for real and reactive power, and condensers in  $C$ , which are responsible for reactive power, are connected to an additional vertex s, which is called source. Buses connected to consumers in  $L$  (also known as buses with load) are connected to an additional vertex t, called sink. Transformers in  $T$  and other components need no special modeling. The cost of each arc is the power loss if it is a transmission line, none cost for else, the graph model of the 14 bus system is shown on the right.

## Notes on Model

The arc capacity is depending on the property of the generator, condenser, transmission line and loads. In this 14-bus system, there may be more than one transmission line between two buses, as in the graph network model, we merge those lines into one arc. The capacity of this arc is the total number of all lines capacity. As the cost, transmission line power loss is actually depend on line length and voltage level. To simply it, we just consider the line length, which in this case is the same.



Mathematical Term	Components in Grid	Remarks
Nodes $N$	Buses, including one general source and one general sink	NA
Arcs $A(i, j)$	Transmission lines and additional lines from generators to source and loads to sink	Arc direction must make obey the physical law
Capacities $u(i, j)$	Maximum load of each electrical component	NA
Costs $c(i, j)$	Transmission line loss	Have same value in this project
Flows $x(i, j)$	Power flows	Only Consider real power

## Feasible Flows in Model

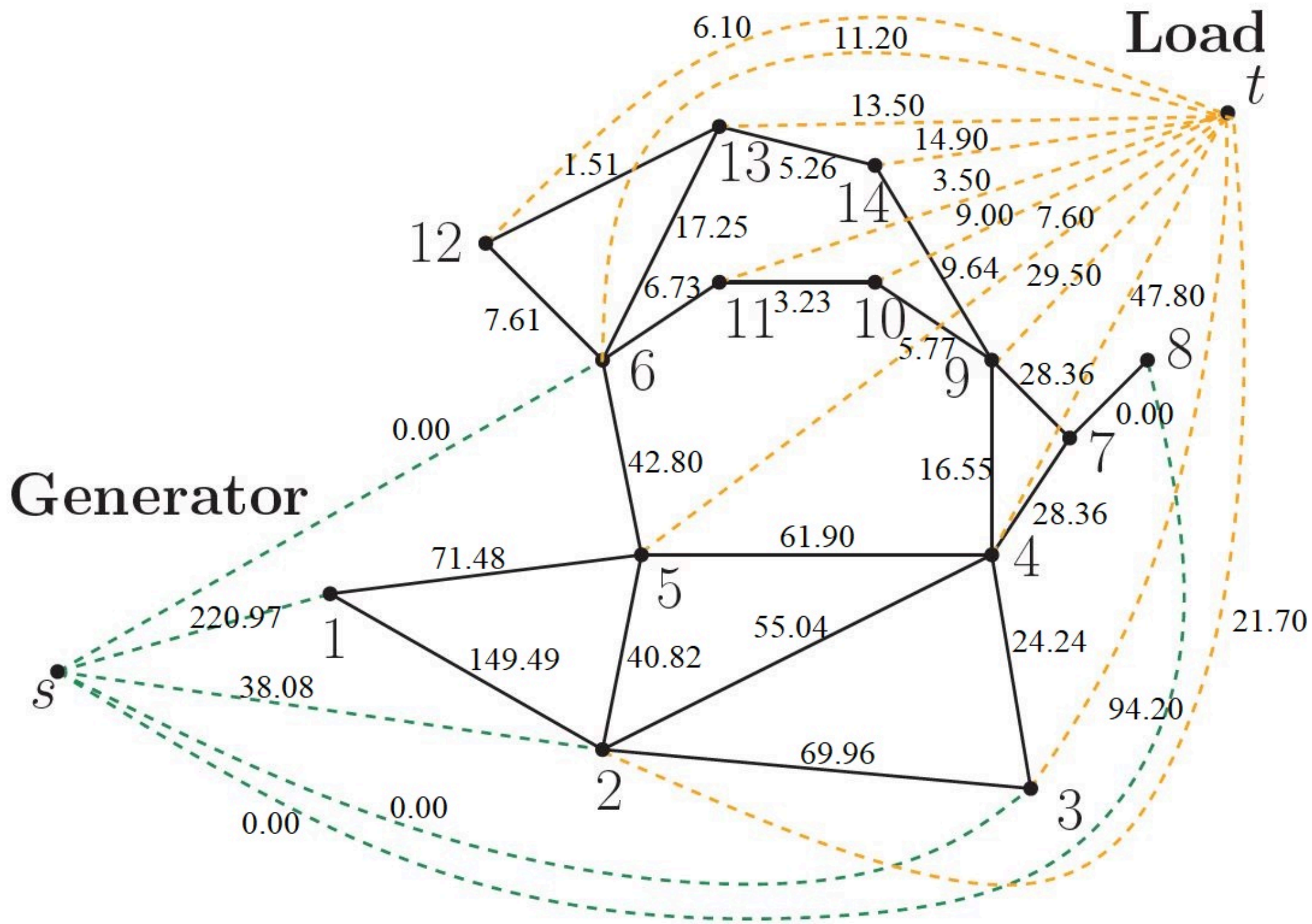
To find a feasible flow in the model, the naive approach is just use the maximum flow algorithm, augmenting along each possible path and then trace back for the flow on each arc. Although topically in a power grid we don't focus on sending as much power as possible, using this approach we can meet the physical law of electrical and keep the whole system stable.

## Results and Discussions

As the flows shows in the figure below (flows marked on the arcs), we can conclude that,

- 1) some arcs are very close to overloaded
- 2) generator distribution is not optimal although all nodes meet the physical laws

Furthermore, if we count the transmission loss into consideration, the total flow amount for real power is very close to optimal flow amount calculated by numerical method. Since the generation distribution is poor in this case, so it can not be the lowest generation cost flows.



Summarizing, network flow model not ideal to solve the complex network like a electrical network. Even with the good amount of simplifications and assumptions, it is still very hard to get the optimal power flows.

## References

- ↑ https://www2.ee.washington.edu/research/pstca/pf14/ieee14cdf.txt  
Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Solving\_Power\_Flows\_Problems\_Using\_Network\_Flows\_Model&oldid=1427"

- This page was last modified on 30 April 2018, at 12:58.
- This page has been accessed 4,868 times.



# Spectral clustering

From CU Denver Optimization Student Wiki

**Spectral clustering**<sup>[1]</sup> is a method that uses weighted graphs to partition the network into groups of different sizes and densities. Let  $G = (V, E)$  be an undirected graph with vertex set  $V = \{v_1, \dots, v_n\}$  and edge at  $E = \{(v_i, v_j) : v_i, v_j \in V\}$ . For each  $v_i$  we define the degree of  $v_i \in V$  as

$$d_i = \sum_{j=1}^n w_{ij},$$

where  $w_{ij}$  is a weight between two vertices  $v_i$  and  $v_j$ ; note that  $w_{ij} \geq 0$  and because the graph is undirected  $w_{ij} = w_{ji}$ .

## Contents

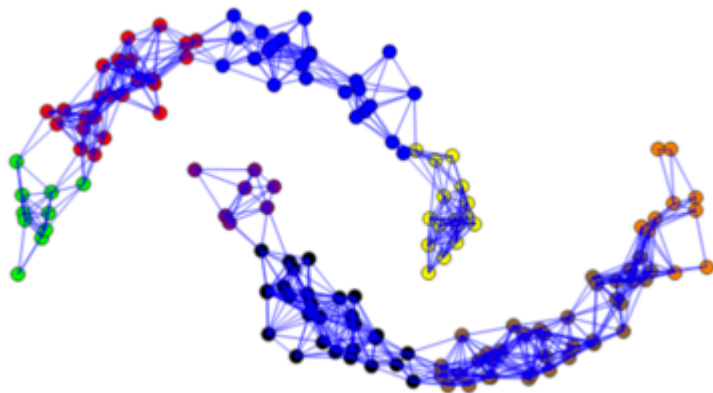
- 1 Different similarity graphs
- 2 The unnormalized graph Laplacian
- 3 Unnormalized spectral clustering algorithm
  - 3.1 Unnormalized spectral clustering algorithm
- 4 Applying spectral clustering
- 5 References

## Different similarity graphs

For a set of data points  $x_1, \dots, x_n$ , with  $x_i \in \mathbb{R}^2$  let the similarity graph be represented by  $G = (V, E)$ , where each vertex  $v_i$  in  $V$  represents a data point  $x_i$ . Given  $s_{ij} \geq 0$  i.e., some notion of similarity between all pairs of data points  $x_i$  and  $x_j$ , two vertices are connected if the similarity value is positive or larger than a certain threshold; the edge is then weighted by  $s_{ij}$ . There are several constructions to transform a given set  $x_1, \dots, x_n$  of data points with pairwise similarities  $s_{ij}$  or pairwise distances  $d_{ij}$  into a graph  $G = (V, E)$ . We will discuss two alternatives ways and give an example for each. In particular, we take the data set and classify it into different groups through spectral clustering. The results obtained for different similarity graphs with the same set of data are shown in the following Figures. These figures show the relationships among artificially generated data and allow us to observe the various advantages of different similarity graph.

### The $\varepsilon$ -neighborhood approach

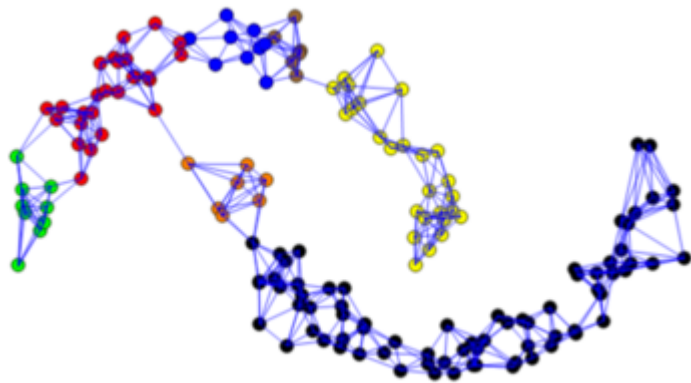
Given a positive  $\varepsilon$ , all points at a distance less than  $\varepsilon$  are connected. As the distances between all connected points are roughly on the same scale (in the sense that they are at most  $\varepsilon$  units away), weighting the edges would not incorporate more information of the data in the graph. Hence, the  $\varepsilon$ -neighborhood graph is usually represented by an unweighted graph. In the Figure below the points within a distance less than  $\varepsilon = 0.28$  are joined by an edge. It shows that the middle points on each *moon* are strongly connected, while the points in the extremes less so. In general, it is difficult to define a parameter  $\varepsilon$  that is robust, especially when the points are distributed with different distances among them depending on the space. If we choose a good parameter  $\varepsilon$ , we obtain well-defined clusters at the output of the algorithm. As the Figure illustrates the data is grouped into eight clusters represented by different colors.



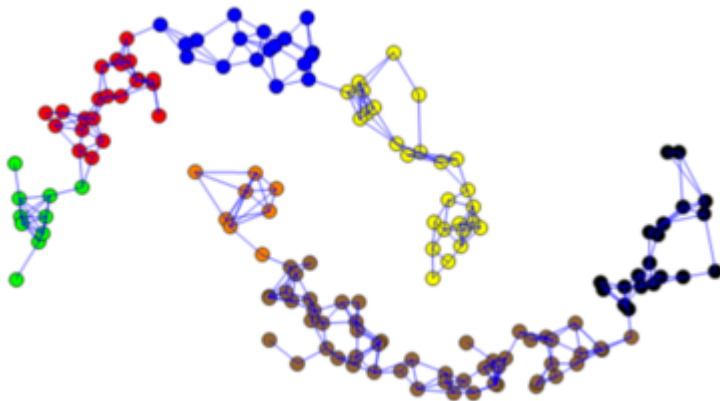
### The $k$ -nearest neighbor approach

Given a positive  $k$ , vertex  $v_i$  is connected to vertex  $v_j$  if  $v_j$  is among the  $k$ -nearest neighbors of  $v_i$ . Note that, this definition leads to a directed graph, as the neighborhood relationship is not symmetric. There are two ways of making it undirected:

- One way is to simply ignore the directions of the edges, that is we connect  $v_i$  and  $v_j$  with an undirected edge if  $v_i$  is among the  $k$ -nearest neighbors of  $v_j$  or if  $v_j$  is among the  $k$ -nearest neighbors of  $v_i$ . The resulting graph is what is usually called the  *$k$ -nearest neighbor graph*, shown in the Figure below. Vertices  $x_i$  and  $x_j$  are connected with an undirected edge if  $x_i$  is among the 6-nearest neighbors of  $x_j$  or if  $x_j$  is among the 6-nearest neighbors of  $x_i$ . The advantage of this approach is that if many points are too close, it will not generate excessive links (as can be seen at the points in the middle of the moon of the Figure). However, the  $k$ -nearest approach can break the graph into several disconnected components and give a wrong number of connected components.



- Another way is to connect vertices  $v_i$  and  $v_j$  if both  $v_i$  is among the  $k$ -nearest neighbors of  $v_j$  and  $v_j$  is among the  $k$ -nearest neighbors of  $v_i$ . The resulting graph is called the \emph{mutual  $k$ -nearest neighbor graph} (shown in the Figure below). Compared to the other approaches mutual  $k$ -nearest neighbor graph does not alter the number of connected components, being appropriate to detect clusters of different densities.



For the last two approaches, after connecting the appropriate vertices, we weight the edges by the similarity of their endpoints.

## The unnormalized graph Laplacian

Let  $W = (w_{ij})_{i,j=1,\dots,n}$  the weighted adjacency matrix of the graph and the degree matrix  $\mathcal{D}$  the diagonal matrix with the degrees  $d_1, \dots, d_n$  on the diagonal. The unnormalized graph Laplacian matrix is defined as

$$L = \mathcal{D} - W.$$

# Unnormalized spectral clustering algorithm

To be able to formalize spectral clustering, we need to define the similarity matrix. Given a set of data points  $x_1, \dots, x_n$  and some notion of similarity  $s_{ij} = s(x_i, x_j)$ , there is a similarity matrix  $S = (s_{ij})_{i,j=1,\dots,n}$  which is assumed to be symmetric and non-negative.

## Unnormalized spectral clustering algorithm

**Input:** Similarity matrix  $S \in \mathbb{R}^{n \times n}$  and number  $m$  of clusters to construct.

- Construct a similarity graph. Let  $W$  be its weighted adjacency matrix.
- Compute the first  $m$  eigenvectors  $u_1, \dots, u_m$  of  $L$ .
- Let  $U \in \mathbb{R}^{n \times m}$  be the matrix containing the vectors  $u_1, \dots, u_m$  as columns.
- Let  $y_i \in \mathbb{R}^m$  be the vector corresponding to the  $i$ -th row of  $U$ , for  $i = 1, \dots, n$
- Cluster the points  $(y_i)_{i=1,\dots,n}$  in  $\mathbb{R}^m$  into clusters  $C_1, \dots, C_m$ .

**Output:** Clusters  $A_1, \dots, A_m$  with  $A_i = \{j | y_j \in C_i\}$ .

The output of the algorithm returns the partitioned network. Points in different clusters are dissimilar to each other; that is, between-cluster similarities are minimized and within-cluster similarities are maximized.

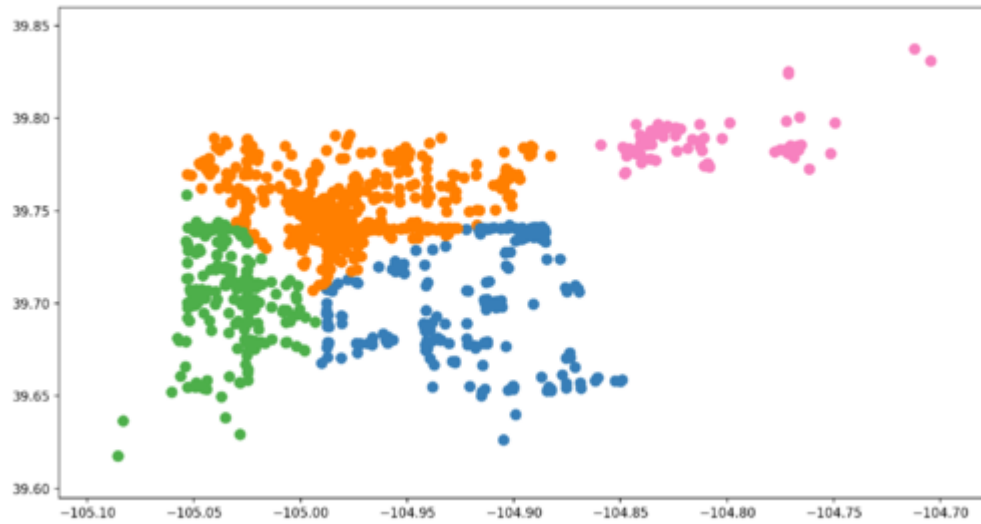
## Applying spectral clustering

Crime response planning by linear programing gives the optimal police location with regard to a set of crime locations from the Open Data Catalog<sup>[2]</sup>. The analysis in the project is focused on murder and robbery data but could be extended to other crimes. The goal is to minimize the expected distance between the crime location and the police location.

The Denver Open Data Catalog includes criminal offenses in the City and County of Denver for the previous five calendar years plus the current year to date. The data is based on the National Incident Based Reporting System (NIBRS). The data is filtered in the preprocessing stage of a Python program, avoiding crime locations outside of Denver area.

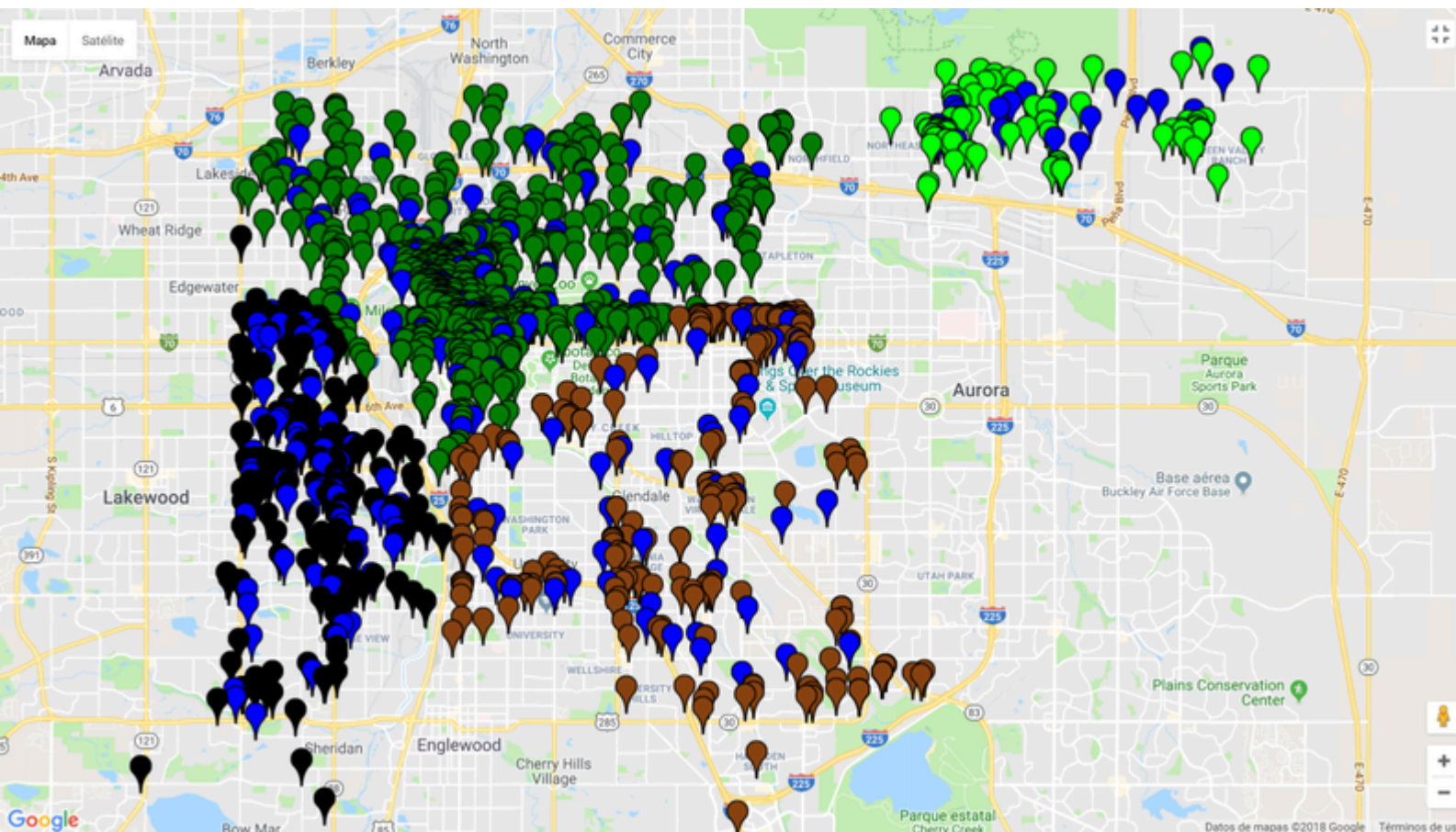
In the first part of the project, we implement and compare the Discrete Wasserstein Barycenters models<sup>[3]</sup> using murder data. Then, we included robbery to the study. This increases the amount of input data for 2016 from 52 to 1204 cases. To be able to include different types of crime and work with a bigger data set, we apply spectral clustering to partition the locations of crime incidents in the Denver area into smaller parts. Then, we apply the discrete barycenter models to determine the optimal police location in

The main result of the project is a Python code that extracts relevant data from Denver Open Data Cataloge, computes spectral clustering, implements discrete barycenters for crime data in AMPL, and provides a visualization of results using Google maps. Repository at <https://github.com/nataliavillegasfranco/barycenters>.



Spectral clustering generates a network that captures the relationships between crime locations, using a similarity matrix  $\mathcal{S} = (s_{ij})_{i,j=1,\dots,n}$  where  $s_{ij} = s(x_i, x_j)$ .

We apply spectral clustering to robbery and murder data from the year 2016 and get partition in the Figure above.



## References



# Stable Marriage Problem

From CU Denver Optimization Student Wiki

The **Stable Marriage Problem** is an example of a *matching problem*, a class of optimization problems in which the primary goal is to find some optimal way to pair up adjacent nodes in a network. In general, matching problems involve an undirected, simple network with weighted edges between pairs of vertices; the optimization involved usually relates to minimizing or maximizing the sum of the edge weights on a set of independent edges (i.e. a collection of edges in which no two edges have a common endpoint). Matching problems as a whole can be split into two separate groups: bipartite weighted matching problems, and nonbipartite matching problem; the distinction is determined by whether or not the underlying network structure is inherently bipartite. The **stable marriage problem** is an example of the former, in which we wish to maximize some function of the edges weights in a bipartite network.

## Contents

- 1 Hall's Marriage Theorem
- 2 Bipartite Weighted Matching Problem
  - 2.1 Runtime for Solution Methods
- 3 The Problem
  - 3.1 Propose-and-Reject Algorithm
- 4 Nonbipartite Cardinality Matching Problem

## Hall's Marriage Theorem

Let  $G$  be a bipartite graph with partite sets  $X$  and  $Y$ . We say that  $G$  admits an  $X$ -saturated matching if there exists a matching  $M \subset E(G)$  for which every vertex in  $X$  is an endpoint of some edge in  $M$ . Hall's Marriage Theorem is one of many theorems in graph theory described as TONCAS: 'The Obvious Necessary Condition is Also Sufficient.' In this case, the obvious necessary condition for an  $X$ -saturated matching in  $G$  is for  $|A| \leq |N(A)|$  for every  $A \subseteq X$ , where  $N(A) \subset Y$  denotes the union of the neighborhoods all of all vertices in  $A$ .

Therefore, a formal statement of Hall's Marriage Theorem is as follows: an  $X, Y$ -bigraph admits an  $X$ -saturated matching if and only if  $|A| \leq |N(A)|$  for all  $A \subseteq X$ . One well-known proof of this claim uses  $M$ -augmenting paths, which are paths in  $G$  given some matching  $M$  which alternate from unmatched edge to matched edge, where both ends of the path are unmatched. We call such a path  $M$ -augmenting because swapping the matched edges for the unmatched edges creates a larger matching, provided the path was maximal (i.e. the endpoints of the path are unmatched vertices under  $M$ ). This proof provides the basis for an algorithm to construct an  $X$ -saturated matching, if one exists in  $G$ .



# Bipartite Weighted Matching Problem

Let  $G = (N_1 \cup N_2, A)$  with  $|N_1| = |N_2| = n$  and arc weights  $c_{ij}$ . By convention, we assume that all arcs are directed from  $N_1$  to  $N_2$ , but this assumption will not be important in the **Stable Marriage Problem**. However, by generating a new source vertex with arcs to every vertex in  $N_1$ , and a new sink node which serves as the head of an arc from each of the vertices in  $N_2$ , we can transform these bipartite weighted matching problems into special versions of a flow problem. Therefore, many of the algorithms and techniques which were developed for solving flow problems can be effectively applied to bipartite weighted matching problems; however, they will generally not account for much of the assumed structure of the bipartite networks underpinning the problems. Examples of such relevant algorithms are *Successive Shortest Path Algorithm*, the *Hungarian Algorithm*, the *Relaxation Algorithm*, and the *Cost-Scaling Algorithm*.

## Runtime for Solution Methods

We denote by  $S(n, m, C)$  the time needed to solve a shortest path problem with nonnegative arc lengths. Then as the Successive Shortest Path, Hungarian, and Relaxation Algorithms are all modifications of shortest path algorithms, it can be shown that they all run in  $O(nS(n, m, C))$  time. However, the basic implementation of the Cost-Scaling Algorithm runs in  $O(nm \log(nC))$  time while a modified version which decomposes the scaling computations in the algorithm runs on  $O(\sqrt{nm} \log(nC))$  time.

## The Problem

The **stable marriage problem** can be posed via a simple story. Imagine a set  $N_1$  of men and  $N_2$  of women ( $|N_1| = |N_2| = n$ ) looking for love; they are provided portfolios on the members of the opposite sex, meet and discuss their lives, preferences, hobbies, etc. Each person then ranks all of the members of the opposite sex. We aim to determine an optimal matching, where optimal means something slightly different than most contexts. For a given matching, a man-woman pair is said to be *unstable* if they are not matched to each other but prefer each other to their current spouses; an optimal matching consists of no *unstable* pairs.

The best possible runtime for any algorithm which solves this problem will be linear in the input size, which consists of two  $n \times n$  matrices consisting of each individual's ranking of the members of the other sex. As the numbers in the matrix only have meaning as a ranking system (i.e. 2 is not `twice as good/bad as' 1), we can assume that without loss of generality that the entries consist of the integers  $1, 2, \dots, n$ . We describe an algorithm below which achieves a runtime linear in the input size.

## Propose-and-Reject Algorithm

The "propose-and-reject algorithm" is an iterative greedy algorithm which can achieve an  $N_i$ -optimal stable matching, meaning every member of  $N_i$  obtains their best-possible stable partner. To initialize the algorithm, we maintain a LIST of thus-far unmatched men, and point each man towards the the woman to whom he will next propose, "current-woman". At the start,  $LIST = N_1$  and each man points to their most-preferred partner.

The algorithm proceeds by arbitrarily having a man in LIST propose to his "current-woman"; as an example, call this man Ted and suppose his "current-woman" is Robin. Robin then chooses her preferred partner amongst Ted and her current partner, always choosing to be matched over not. The unmatched man from this scenario, call him Barney, is then added to LIST and his "current-woman" is his next most preferred woman to Robin. The algorithm terminates only when LIST is empty. As each man's "current-woman" only moves down his priority list every time he loses his partner, it is clear that eventually the algorithm will select a man whose "current-woman" is unmatched, thus reducing the number of unmatched women. Since a matched woman never becomes un-matched, LIST must eventually become empty.

Furthermore, showing that the algorithm produces a stable matching is not difficult. We will assume that some man, Ted, prefers some woman, Robin, to his current partner, Victoria; as Ted proposed to all of the women he likes as much as he likes Victoria, he must have already proposed to Robin. As a woman's matched partners can only become more desirable in this algorithm, Robin could not prefer Ted to her current partner. If Robin prefers Barney to her current partner, Sandy Rivers, then Barney must have never proposed to Robin; hence Barney prefers his current partner to Robin. Therefore, once LIST is empty, the matching must be stable.

Furthermore, as every woman can only reject a man at most once, rejections occur at most  $n - 1$  times, whereas the only time a rejection does not occur is the first time a man proposed to that particular woman, which happens once per woman. Since the algorithm performs  $n$  steps per woman, and these operations consist of all steps, the algorithm runs in  $O(n^2)$  time. As the input data is of size  $n^2$ , the "propose-and-reject algorithm" is linear in the size of the input and thus has the best possible complexity bound. Furthermore, as there are never steps that depend on the nature of any of the rankings, every pair of ranking matrices admits a stable matching.

It should be noted, however, that stable matchings are not unique. As mentioned above, this algorithm results in a matching in which every man is partnered with their best-possible stable partner; however, if we set up the algorithm so that women were proposing, then we would have that women are matched with their best-possible stable partner which could potentially be different. Suppose  $N_1 = \{Ted, Barney, Marshall\}$ ,  $N_2 = \{Robin, Victoria, Lily\}$ , and the preference lists are as follows: Ted prefers Robin, Victoria, Lily; Barney prefers Victoria, Lily, Robin; Marshall prefers Lily, Robin, Victoria; Robin prefers Barney, Marshall, Ted; Victoria prefers Marshall, Ted, Barney; and Lily prefers Ted, Barney, Marshall. If the men propose, the resulting matching is Ted-Robin, Barney-Victoria, and Marshall-Lily, whereas the women proposal will lead to Robin-Barney, Victoria-Marshall, Lily-Ted.

## Nonbipartite Cardinality Matching Problem

The **nonbipartite cardinality matching problem** (or **nonbipartite matching problem**) relaxes the condition that the underlying network is bipartite. In general, augmenting paths like those used in the proof of Hall's Marriage Theorem are used to solve the bipartite matching problem; these techniques work because the distance between nodes in a bipartite network maintain their parity always. However, the existence of odd cycles in general networks means that, depending on the order in which a search occurs, the distance between nodes can have different parity when searching along augmenting paths. In order to resolve this issue, additional terminology and more theory is required. However, the general solution technique for solving the nonbipartite matching problem mimics that of the bipartite matching problem: start with a feasible matching (potentially empty) and search for an unmatched vertex; if none exist, we terminate, but if we find one, we search for an augmenting path; if one exists, we augment our matching, and if not, we delete the vertex. This is because it is possible to show that if a node  $p$  is unmatched in a matching  $M$ , and this matching contains no augmenting path that starts at node  $p$ , then node  $p$  is unmatched in some maximum matching. As the general technique is the same regardless of whether or not there are odd cycles in the network, the additional complexity is inherent only to finding augmenting paths along odd cycles. For more information, we suggest the reader search for the "Edmond's Algorithm."

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Stable\_Marriage\_Problem&oldid=1423"

- This page was last modified on 30 April 2018, at 12:52.
- This page has been accessed 719 times.

# StarCraft II Build Order Optimization

From CU Denver Optimization Student Wiki

The StarCraft franchise developed by Blizzard Entertainment is one of the most popular real-time strategy video games. StarCraft launched in 1998 and gave rise to celebrity-status professionals and prestigious tournaments, and is still updated with new content and expansions. Essentially a war simulation, the goal of the game is to eliminate your opponent by amassing an army and making your opponent forfeit, or destroying all of your opponent's structures. A player has a choice of playing as three different races – Terran, Protoss, or Zerg – all with unique units, buildings, abilities, and mechanics. A player begins with nothing but a base and six worker units meant for collecting resources (minerals and gas) that are used to produce structures and more advanced combat and support units. The opening minutes and the timing of certain actions are crucial to the outcome of a game, as delaying unit production or failing to produce workers for a robust economy can cost a player a win. With this in mind, linear programming lends itself to finding the shortest time one can implement a rush-based build order, under 3 and a half minutes, with respect to a simplified model of unit, building, and resource economies.



A Terran base with the starting six workers. A seventh is being created in the Command Center.

## Contents

- 1 Inspiration and Method
- 2 Model Overview
  - 2.1 Definitions
  - 2.2 Basics
  - 2.3 Race Specific Differences
    - 2.3.1 Worker Dynamics
    - 2.3.2 Supply
    - 2.3.3 Additional Differences
  - 2.4 Data
  - 2.5 Objective Functions
- 3 Results
  - 3.1 Protoss Build - 2 Gate Zealot Rush
  - 3.2 Terran Build - 3 Barracks Marine Rush
  - 3.3 Zerg Build - 8-Pool
- 4 Afterthoughts
  - 4.1 Hard Coding
  - 4.2 Output
  - 4.3 Limitations and Missing Elements
- 5 Future Work
- 6 References

## Inspiration and Method

Inspiration comes from Churchill and Buro’s Build Order Optimization in StarCraft<sup>[1]</sup>, an article focused on artificial intelligence advancement with respect to StarCraft. The authors successfully created a “planner” that when incorporated with a playing agent performed comparable to professionals. Methods included a depth-first branch and bound algorithm that takes a game state and performs a recursive search on possible build orders that satisfy a particular goal. The goal of this project was to apply linear programming to their notion of optimizing build orders and assess its efficacy. Regarding a linear programming method for approaching this problem, the critical path method was the original idea.

Each race has a particular tech tree that describes the prerequisites for constructing certain buildings and which units those buildings may produce. One can think of the the game as a task scheduling problem. For example constructing a Protoss Gateway, a first tier combat-producing building, is a critical activity for producing a Protoss Zealot, a first-tier combat unit. The critical path method however does not suffice due to the added complications of resource generation and management, continuous worker production, supply tracking, and race-specific mechanics. It is important to note that these models are consistent with game data from the particular StarCraft II expansion Wing of Liberty, as starting worker values and other parameters change in later expansions.



The Terran tech tree, showing building and unit prerequisites.

## Model Overview

## Definitions

Definition	Protoss	Terran	Zerg
First Tier Building	Gateway	Barracks	Spawning Pool
First Tier Combat Unit	Zealot	Marine	Zergling
Supply Building	Pylon	Supply Depot	Overlord

Supply is defined in-game as the sum of the number of workers and first tier combat units. Each race has a particular supply limit, and once that limit is reached, a player must build a supply building before they are allowed to produce any more units. The strategy is to construct such buildings well before the supply limit is reached, so that unit production does not come to a halt.

# Basics

The models for the opening minutes of a StarCraft game are discrete in time, and variable are either declared integers or are integers by nature, other than minerals. For example, the following lines of code for the Protoss race’s model define the total minerals, workers, Zealots, Gateways (variable named Barracks for consolidation purposes), and supply at time  $t$ .

- var Minerals {t in 1..T} >= 0;
- var Workers {t in 1..T} integer >= 0;
- var Zealots {t in 1..T} integer >= 0;
- var Barracks {t in 1..T} >= 0;
- var Pylon {t in 1..T} >= 0;



A close-up of a Terran Supply Depot

There following are additional integer variables that denote the start of construction of a building or unit:

- $\text{var build } \{t \text{ in } 1..T\} \text{ integer } \geq 0$ ; Denotes the start of construction of a first tier building at time  $t$
- $\text{var create } \{t \text{ in } 1..T\} \text{ integer } \geq 0$ ; Denotes the start of construction of a first tier combat unit.
- $\text{var Create\_Worker } \{t \text{ in } 1..T\} \text{ integer } \geq 0$ ; Denotes the start of construction of a worker unit.

The variable "build" denotes the start of construction of a first tier building, "create" denotes the start of a first tier combat unit, and "Create\_Worker" denotes the start of producing a worker. There is also a variable for each race that denotes the start of constructing a supply building. Construction of buildings and units takes time that varies from race to race which will be shown. The method for tracking unit economies lies in the constraints. The following constraints track minerals and Zealots respectively for each discrete time point.

subject to Mineral\_Count  $\{t \text{ in } 2..T\}$ :

- $\text{Minerals}[t] = \text{Minerals}[t-1] + \text{Workers}[t-1] * \text{rate} - 150 * \text{build}[t-1] - 100 * \text{create}[t-1] - 100 * \text{Pylon\_Start}[t-1] - \text{Create\_Worker}[t-1] * 50$ ;

subject to Zealot\_Count  $\{t \text{ in } 28..T\}$ :

- $\text{Zealots}[t] = \text{Zealots}[t-1] + \text{create}[t-27]$ ;

The coefficients in front of the latter four terms in the "Minerals" constraint correspond to their respective mineral costs of construction. Each worker collects a certain number of minerals per second. Additionally, notice that a Gateway will take 27 seconds to produce a Zealot, so the Zealot count is updated using the "create" variable at the time point 27 seconds in that past. The following lines of code implement that tech tree that dictates a player must have, for example, a Gateway before producing a Zealot.

subject to Available\_Buildings  $\{t \text{ in } 47..T\}$ :

- $\text{Available}[t] = \text{Available}[t-1] + \text{build}[t-46] - \text{create}[t-1] + \text{create}[t-28]$ ;

subject to Building\_Existance  $\{t \text{ in } 1..T\}$ :

- $\text{create}[t] \leq \text{Available}[t]$ ;

First tier building construction takes 46 seconds to complete and the buildings are ready to produce units upon completion. Supply constraints are implemented by simply summing the number of workers and first tier combat units and ensuring it is less than the initial race's supply cap plus the number of supply buildings multiplied by 8, which is the number of supply each supply building provides. Details are discussed in the following section.

## Race Specific Differences

Each race contains slight nuances that present their own challenges with respect to model economies.



A Protoss Zealot



A Terran Barracks

## Worker Dynamics

The most apparent differences are the worker dynamics with respect to building construction. While Protoss workers simply drop an orb that constructs itself over a certain period of time (46 seconds for a Gateway and 30 seconds for a Pylon), Terran workers remain occupied for the duration of construction. A Zerg worker is in fact lost when instructed to construct a building. The following code shows an additional term at the end that reflects Zerg worker loss.

### ***Zerg Worker Economy***

subject to Worker\_Count {t in 13..T}:

- $Workers[t] = Workers[t-1] + Create\_Worker[t-12] - build[t-1];$

Most complicated of course are the Terran worker economies, as 3 additional constraints must be added to account for occupied workers.

### ***Additional Terran Constraints***

subject to Worker\_Count {t in 13..30}:

- $Avail\_Workers[t] = Workers[t-1] - Supply\_Start[t-1] - build[t-1];$

subject to Worker\_Count2 {t in 31..46}:

- $Avail\_Workers[t] = Workers[t-1] - build[t-1] - Supply\_Start[t-1] + Supply\_Start[t-30];$

subject to Worker\_Count3 { t in 47..T}:

- $Avail\_Workers[t] = Workers[t-1] - build[t-1] + build[t-46] - Supply\_Start[t-1] + Supply\_Start[t-30];$



A close up of a Terran worker

## Supply

All races begin with 6 workers. Both the Protoss and Zerg start with an initial maximum supply of 10. The Zerg begin with an Overlord that provides 8 supply, and their base provides an additional 2. The Protoss' base provides 10 supply. The Terran base provides 11 supply. Each race's supply building provides 8 supply, and the following code shows the slight differences between how the models track supply.

### ***Terran:***

subject to Unit\_Count {t in 1..T}:

- $Workers[t] + Marine[t] \leq 11 + 8 * Supply[t];$

### ***Protoss:***

subject to Unit\_Count {t in 1..T}:

- $Workers[t] + Zealots[t] \leq 10 + 8 * Pylon[t];$

### ***Zerg:***

subject to Unit\_Count {t in 1..T}:



■  $Workers[t] + Zergling[t] \leq 2 + 8 * Overlord[t];$

## Additional Differences

The Zerg race uses larvae produced at their base to produce Zerglings provided a Spawning Pool exists. Only one Spawning Pool is required as opposed to other races who can take advantage of producing Marines or Zealots from multiple Barracks or Gateways simultaneously. Additionally, Spawning Pools cost 200 minerals while Barracks and Gateways cost 150. A pair of Zerglings costs 50 minerals, a Marine costs 50, and a Zealot costs 100. Zealots and Zerglings take 27 to complete, and Marines take 18 seconds.

## Data

The following data chart is used across all races. Note that T corresponds to in-game seconds.

Parameter	Value
T (run time)	200
rate (minerals/worker/sec)	0.625
minerals0 (initial minerals)	0
workers0 (initial workers)	0
marines0 (initial tier1 units)	0
barracks0 (initial tier1 buildings)	0

## Objective Functions

The objective functions for each race are essentially identical. The goal is to maximize the following value:

■  $\sum \{t \text{ in } 47..T\} build[t-46]*150/t + \sum \{t \text{ in } 1..T\} create[t]*100/t;$

Notice that this value is maximal when the nonzero values of "build" and "create" occur at the smallest possible t. The variables are multiplied by their respective associated mineral costs. The above example is for the Protoss - a Gateway costs 150 minerals and a Zealot costs 100. It was noticed that the variable weights can chosen somewhat arbitrarily. Ideally, the weights would change so that after producing a Gateway, their is not as much priority on the second one. This was noticed most prominently in Terran's case where Barracks are prioritized over Marines.



An early-game Zerg base with a Spawning Pool as well as an Extractor for collecting gas.

# Results

## Protoss Build - 2 Gate Zealot Rush

Action	Start	Completed
Worker	14	26
Worker	27	29
Pylon	49	79
Gateway	78	124
Gateway	108	154
Zealot	128	155
Zealot	154	181
Zealot	168	195
Probe	181	193
Probe	188	200
Zealotx2	200	227

Existing build:

- 2 Workers
- Pylon
- 2 Gateways
- 2 Workers
- 4 Zealots

The desired existing build order for Protoss

The only difference between build orders is that the linear programming model prioritizes Zealots over workers. Building two workers before beginning Zealot production would delay the time of completing the first Zealot.



# Terran Build - 3 Barracks Marine Rush

Action	Start	Completed
Worker	14	26
Worker	27	39
Worker	38	50
Barracks	67	113
Worker	76	88
Worker	84	96
Barracks	108	154
Barracks	130	176
Supply Depot	144	174
Marine	156	174
Marine	159	177
Marine	176	194
Marine	184	202
Marine	187	205
Worker	188	206

Existing build:

- 3 Workers
- Supply Depot
- 2 Workers
- Barracks
- 1 Worker
- 2 Barracks
- Marines

The desired existing build order for Terran

Again, due to objective function prioritization, two Barracks finish before Marine production starts. Dynamic prioritization could possibly produce a similar build, but with earlier Marine production at the expense of slightly later Barracks timing.

## Zerg Build - 8-Pool

Action	Start	Completed
Worker	14	26
Worker	27	39
Spawning Pool	68	114
Worker	80	92
Worker	91	103
Zergling Pair	114	141
Zergling Pair	119	146
Overlord	128	158
Zergling Pair	137	164

**Zerg 8-Pool Rush**

- 2 Workers
- Spawning Pool
- 2 Workers
- Overlord
- Zerglings

The desired existing build order for Zerg

Zergling production continues at the following time points: 146, 155, 164, 172, 181, 190, 199, and 200 seconds. This build order only differs from the original in Overlord timing. This is of course in an effort to produce Zerglings as soon as possible.

## Afterthoughts

### Hard Coding

Some hard coding was necessary to produce desired results in a timely fashion. Firstly, the Pylon timing was hard coded as a constraint. subject to PylonReq:

- `Pylon_Start[49] = 1;`

The Protoss have a special condition. Pylons produce a circular power field, and building construction is limited to these power fields. A fix for this issue would be to implement a conditional variable that equals zero when there are no Pylons, and one otherwise. Multiplying this conditional variable by the "build" variable in the Protoss' Gateway constraint would ensure a Pylon exists to power buildings.

Another instance of hard coding was the timing of the Zerg Spawning Pool. This was in fact because only one Spawning Pool is required, and the timing choice is the user's. The particular pre-existing build called for a Spawning Pool at 8 supply, which led to evaluating the worker output, noting the instant the supply reached 8 and the mineral count 200, and hard coding the Spawning Pool timing accordingly. subject to 8pooltime:

- `build[68] = 1;`

Additional hard coding included designating a particular number of worker for the build as well as the number of Barracks in the Terran build. These were choices made with respect to the desired build order, and aided in simplifying consistent worker generation and the objective function. Problems arise when worker terms are included in the objective function - the function becomes unbounded.



Choosing the placement of the first Pylon with a visible potential power field.

# Output

mpg: display workers															
workers [1] 16															
1	0	21.6	41.0	61.0	81.7	101.0	121.9	141.9	161.0	181.9	201.9	221.9	241.9	261.9	281.9
2	0	22.6	42.0	62.0	82.7	102.0	122.9	142.9	162.0	182.9	202.9	222.9	242.9	262.9	282.9
3	0	23.6	43.0	63.0	83.7	103.0	123.9	143.9	163.0	183.9	203.9	223.9	243.9	263.9	283.9
4	0	24.6	44.0	64.0	84.7	104.0	124.9	144.9	164.0	184.9	204.9	224.9	244.9	264.9	284.9
5	0	25.6	45.0	65.0	85.7	105.0	125.9	145.9	165.0	185.9	205.9	225.9	245.9	265.9	285.9
6	0	26.7	46.0	66.0	86.7	106.0	126.9	146.9	166.0	186.9	206.9	226.9	246.9	266.9	286.9
7	0	27.7	47.0	67.0	87.7	107.0	127.9	147.9	167.0	187.9	207.9	227.9	247.9	267.9	287.9
8	0	28.7	48.0	68.0	88.7	108.0	128.9	148.9	168.0	188.9	208.9	228.9	248.9	268.9	288.9
9	0	29.7	49.0	69.0	89.7	109.0	129.9	149.9	169.0	189.9	209.9	229.9	249.9	269.9	289.9
10	0	30.7	50.0	70.0	90.7	110.0	130.9	150.9	170.0	190.9	210.9	230.9	250.9	270.9	290.9
11	0	31.7	51.0	71.0	91.7	111.0	131.9	151.9	171.0	191.9	211.9	231.9	251.9	271.9	291.9
12	0	32.7	52.0	72.0	92.7	112.0	132.9	152.9	172.0	192.9	212.9	232.9	252.9	272.9	292.9
13	0	33.7	53.0	73.0	93.7	113.0	133.9	153.9	173.0	193.9	213.9	233.9	253.9	273.9	293.9
14	0	34.7	54.0	74.0	94.7	114.0	134.9	154.9	174.0	194.9	214.9	234.9	254.9	274.9	294.9
15	0	35.7	55.0	75.0	95.7	115.0	135.9	155.9	175.0	195.9	215.9	235.9	255.9	275.9	295.9
16	0	36.7	56.0	76.0	96.7	116.0	136.9	156.9	176.0	196.9	216.9	236.9	256.9	276.9	296.9
17	0	37.7	57.0	77.0	97.7	117.0	137.9	157.9	177.0	197.9	217.9	237.9	257.9	277.9	297.9
18	0	38.7	58.0	78.0	98.7	118.0	138.9	158.9	178.0	198.9	218.9	238.9	258.9	278.9	298.9
19	0	39.7	59.0	79.0	99.7	119.0	139.9	159.9	179.0	199.9	219.9	239.9	259.9	279.9	299.9

The Zerg Worker count for the 8-pool build order

particular amount of energy to use the "Chrono Boost" ability which speeds up the production of a unit by a certain percentage. Terran are able to call down MULEs that mine minerals at an increased rate. Zerg units have a faster movement speed while standing on "creep" - a slimy substance that surrounds their base and spreads given the existence of certain units. Nonlinearities such as these and probabilistic actions that depend on game state are beyond the scope of what this basic linear program is suited for. The model's run time is another one of its disadvantages. The model, written and solved using AMPL and CPLEX, can take an astoundingly long time to run. Some runs had to be terminated after more than 7 hours. Hard coding became handy in this case, to reduce the amount of iterations required by the algorithm. The successful Protoss build took 49093070 simplex iterations, and other runs exceeded 220 million iterations. This is most likely due to the existence of large numbers of integer variables. Player versus player interactions and unit micro-management are other game aspects not assessed by such a linear programming model.

## Future Work

The next steps for the model is to include gas as a resource. This will open up multiple new possible units and buildings, but presents large challenges. Gas is mined by workers similarly to minerals after a building is constructed on gas geysers. There are two gas geysers per base, and gas is mined at a maximal rate with 3 workers assigned to the geyser. This means workers must be re-assigned to collect gas from minerals. The following shows initial constraints to implement a simple gas economy.

subject to Gas\_Total {t in 2..T}:

- Gas[t] = Gas[t-1] + Gas\_Workers[t-1]\*rate;

subject to Assigned\_Task { t in 1..T}:

- Gas\_Workers[t] + Mineral\_Workers[t] <= Workers[t];

However, the addition of such an economy would further reduce efficiency with respect to model solve time. Solve time is another priority, and relaxing integer constraints might aid in that respect with the challenge of producing plausible results with respect to game strategy. The objective function also requires improvement to reflect the dynamic prioritization that occurs after first tier buildings complete and unit production is available.

This linear programming model proved to be a useful tool for finding precise timings down to the second for actions rather than supply guidelines for build orders. With improved run time and a more complete model including gas and a fuller tech tree, this model could be used to develop, alter, and optimize early-game build orders.

During model manipulation and testing, the output became invaluable. Displaying worker count and mineral count provided great insight into how the economies changed given certain actions. The linear program not only successfully produces comparable rush-based build orders to existing ones, but provides the user with some useful data such as economic situations at later time points and how one more or less worker can impact the timing of building and unit creation.

## Limitations and Missing Elements

There is a large part of the game that linear programming cannot model, such as various powers and abilities different units and buildings possess. For example, the Protoss base can accumulate energy, and expend a

Minerals [1] 16															
1	0	41	9.375	81	27.5	121	87.5	161	11.875						
2	3.75	42	14.375	82	33.125	122	94.375	162	18.75						
3	7.5	43	19.375	83	38.75	123	101.25	163	25.625						
4	11.25	44	24.375	84	44.375	124	108.125	164	32.5						
5	15	45	29.375	85	0	125	115	165	39.375						
6	18.75	46	34.375	86	5.625	126	121.875	166	46.25						
7	22.5	47	39.375	87	11.25	127	128.75	167	53.125						
8	26.25	48	44.375	88	16.875	128	135.625	168	60						
9	30	49	49.375	89	22.5	129	142.5	169	66.875						
10	33.75	50	54.375	90	28.125	130	149.375	170	73.75						
11	37.5	51	59.375	91	33.75	131	156.25	171	80.625						
12	41.25	52	64.5	92	41.25	132	163.25	172	87.5						
13	45	53	69.625	93	47.5	133	170.375	173	94.375						
14	48.75	54	74.75	94	53.75	134	177.5	174	101.25						
15	52.5	55	79.875	95	60	135	184.625	175	108.125						
16	56.25	56	85	96	66.25	136	191.75	176	115.125						
17	60	57	90.125	97	72.5	137	198.875	177	122.125						
18	63.75	58	95.25	98	79.375	138	206	178	129.125						
19	67.5	59	100.375	99	86.25	139	213.125	179	136.125						
20	71.25	60	105.5	100	93.125	140	220.125	180	143.125						
21	75	61	110.625	101	100	141	227.125	181	150.125						
22	78.75	62	115.75	102	106.875	142	234.125	182	157.125						
23	82.5	63	120.875	103	113.75	143	241.125	183	164.125						
24	86.25	64	126	104	120.625	144	248.125	184	171.125						
25	90	65	131.125	105	127.5	145	255.125	185	178.125						
26	93.75	66	136.25	106	134.375	146	262.125	186	185.125						
27	97.5	67	141.375	107	141.25	147	269.125	187	192.125						
28	101.25	68	146.5	108	148.125	148	276.125	188	199.125						
29	105	69	151.625	109	155	149	283.125	189	206.125						
30	108.75	70	156.75	110	161.875	150	290.125	190	213.125						
31	112.5	71	161.875	111	168.125	151	297.125	191	220.125						
32	116.25	72	167	112	174.375	152	304.125	192	227.125						
33	120	73	172.125	113	180.625	153	311.125	193	234.125						
34	123.75	74	177.25	114	186.875	154	318.125	194	241.125						
35	127.5	75	182.375	115	193.125	155	325.125	195	248.125						
36	131.25	76	187.5	116	199.375	156	332.125	196	255.125						
37	135	77	192.625	117	205.625	157	339.125	197	262.125						
38	138.75	78	197.75	118	211.875	158	346.125	198	269.125						
39	142.5	79	202.875	119	218.125	159	353.125	199	276.125						
40	146.25	80	208	120	224.375	160	360.125	200	283.125						

The Mineral output for the Terran model

# References

1. ↑ [Churchill, D., and Buro, M. 2011. Build order optimization in starcraft. In Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=StarCraft\\_II\\_Build\\_Order\\_Optimization&oldid=886](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=StarCraft_II_Build_Order_Optimization&oldid=886)"

- 
- This page was last modified on 7 December 2017, at 03:40.
  - This page has been accessed 4,777 times.

# Steffen Borgwardt

From CU Denver Optimization Student Wiki



Steffen Borgwardt is an associate professor at the University of Colorado Denver.

My area of research is combinatorial optimization and mathematical programming. More precisely, I study high-dimensional geometric objects arising in operations research and data analytics. I enjoy an application-driven approach and see optimization as a beautiful blend of theory and working in applications.

This wiki is an outlet for all the student projects that I mentor at CU Denver.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Steffen\\_Borgwardt&oldid=4304](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Steffen_Borgwardt&oldid=4304)"  
Category: Contributors

- 
- This page was last modified on 19 April 2023, at 12:05.
  - This page has been accessed 3,611 times.

# Stetson Zirkelbach

From CU Denver Optimization Student Wiki

Stetson Zirkelbach is currently a first year master's student at the University of Colorado at Boulder. I completed my Bachelors at University of Colorado Boulder, I intend to study Optimization and Discrete math. Outside of class I love to throw down in any kind of board or video game.

I took Linear programming in Fall of 2017 and I am currently enrolled in Network Flows.

Here are my contributions to this wiki:

StarCraft II Build Order Optimization

Increasing Network Robustness

Monitoring Population to Track Low Student Retention

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Stetson\\_Zirkelbach&oldid=3507](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Stetson_Zirkelbach&oldid=3507)"

Category: Contributors

- 
- This page was last modified on 6 December 2021, at 14:26.
  - This page has been accessed 3,382 times.

# Support Vector Machines

From CU Denver Optimization Student Wiki

## Support Vector Machines

Support Vector Machines (SMVs) are a supervised learning algorithm often employed for pattern recognition. The SVM model considers observations as points in a mathematical space, mapped so that observations that possess different characteristics are divided by a distinct gap, called a margin. Part of the Support Vector Machine algorithm is to determine the widest possible margin between the two categories as defined by the best possible separating hyperplane. New observation data are then mapped into the space and are predicted to belong to one of the categories based on which side of the margin the point falls. Employing the Lagrangian Dual in the SVM model can not only solve the problem, but also allows us to identify support vectors and make predictions about the other vectors in the set. The points mapped from the original observation data is called the training set, and are divided into two distinct categories based on whether a point, represented as a vector in  $\mathbb{R}^n$ , possesses the characteristic(s) of interest. (Please note that the SVM topics covered here are restricted to the linearly separable case, and do not explore the non-linearly separable case that utilizes a Kernel-based solving method that also utilizes Lagrangian Duality.) When we separate the points among the training set, there will be at least two points among the whole set that define the margin between the two partitions. These points are called support vectors, and are defined by the fact that if we were to perturb any one of the support vectors, we change the boundaries of the margin separating the two sets.

The vectors belonging to the training set are usually assigned  $+1$  or  $-1$  to distinguish whether they fall to the "top" or "bottom" of the separating hyperplane, respectively. If a point is very far from the boundaries of the margin, it is much more predictable than a point that lies very close to the boundary. And as the support vectors define the margin, we would like to maximize the distance between them to more confidently define the categories. This is why we seek to maximize the margin. To consider whether a point falls into one category or another, we define a (binary) linear classifier with labels  $y \in \{-1, 1\}$  and parameters  $w, b$ , to be:

$$h_{w,b}(x) = g(w^T x + b) \text{ [1]. So, } g(w^T x + b) = \begin{cases} +1 & \text{if } w^T x + b \geq 0 \\ -1 & \text{otherwise} \end{cases}.$$

In general, the SVM model seeks a classifier (a linear separator) with as big a margin as possible. Recall that the distance from a point  $(x_0, y_0)$  to the line  $Ax + By + c = 0$  is  $\frac{Ax_0 + By_0 + c}{\sqrt{A^2 + B^2}}$ . This means that the distance between the separating hyperplane,  $H$ , and the positive boundary,  $H_1$ , is

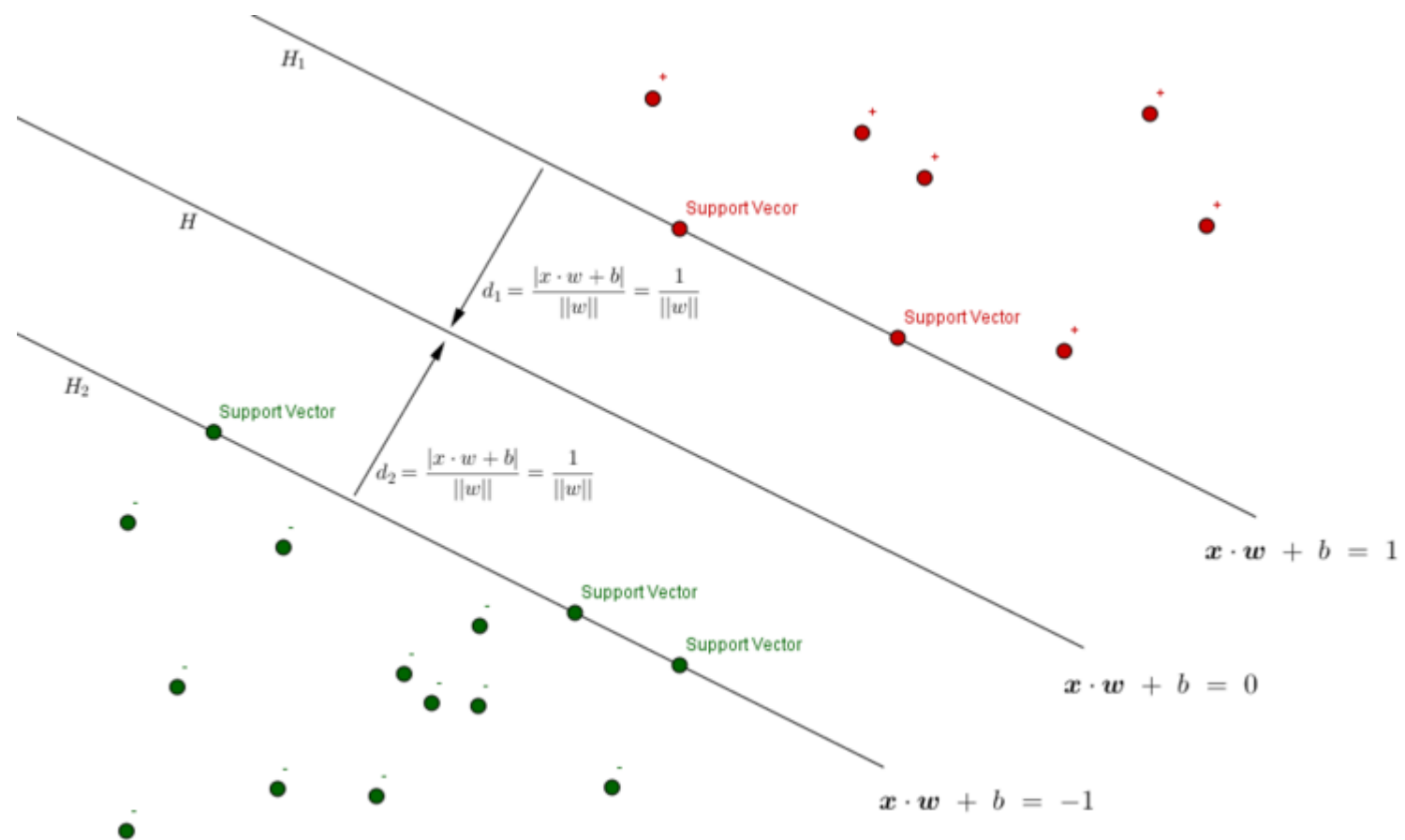
$$\frac{|w \cdot x + b|}{\|w\|} = \frac{1}{\|w\|}. \text{ This is also the distance between } H \text{ and } H_2. \text{ This implies that the total distance between } H_1 \text{ and } H_2 \text{ is } \frac{2}{\|w\|}.$$

Therefore to maximize the margin with a width of  $\frac{2}{\|w\|}$ , we seek to minimize  $\|w\|$  with the constraint that no data points fall between  $H_1$  and  $H_2$ .

And these conditions can be summarized as  $\begin{cases} \mathbf{x} \cdot \mathbf{w} + b \geq +1 & \text{when } y_i = +1 \\ \mathbf{x} \cdot \mathbf{w} + b \leq -1 & \text{when } y_i = -1 \end{cases}$  and can be combined into the generalized condition:  $y_i(\mathbf{x} \cdot \mathbf{w} + b) \geq 1$  [2].

Thus we end up with the following quadratic program:

$$\min ||w||$$



$$s.t. y_i(\mathbf{x} \cdot \mathbf{w} + b) = 1 \quad i = 1, 2, \dots, m$$

This optimization problem can be rewritten as:

$$\min \frac{1}{2} ||w||^2$$

$$s.t. y_i(\mathbf{x} \cdot \mathbf{w} + b) = 1 \quad i = 1, 2, \dots, m$$

Now that we have defined the program, we can formulate the Lagrangian Function. For the SVM optimization problem, the Lagrangian is:

$L(w, b, \alpha) = \frac{1}{2} ||w||^2 - \sum_{i=1}^m \alpha_i [y_i(\mathbf{x} \cdot \mathbf{w} + b) - 1]$  [3]. And to find the Lagrangian Dual, we first minimize  $L(w, b, \alpha)$  with respect to  $w$  and  $b$ , fix the

Lagrangian Multipliers  $\alpha_i$  with the constraint  $\alpha_i \geq 0$  then find the maximum minimum value.

STEPS OMITTED

Therefore the Lagrangian Dual formulation of the SVM program is:

$$\max_{\alpha} \theta(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

$$s.t. \alpha_i \geq 0, i = 1, \dots, m$$

$$\sum_{i=1}^m \alpha_i y_i = 0 \text{ [1].}$$



And it has been shown that the Lagrangian Multipliers, the  $\alpha_i$ 's in this case, will all be zero except for at the active constraints, which in this case are the support vectors. Therefore, by considering the Lagrangian Dual formulation of the SVM problem, we gain insight into the structure of the original problem. This not only helps us to solve for the optimal separating marginal hyperplane, but the Lagrangian Multipliers also allow us to also "position" the vectors that do not support the margin.

## References

- ↑ <sup>1.0</sup> <sup>1.1</sup> Ng, Andrew. "Support Vector Machines." Stanford. CS229 Lecture Notes.
- ↑ Weston, Jason. "Support Vector Machine (and Statistical Learning Theory) Tutorial. NEC Labs. Princeton, USA.
- ↑ Cristianin, Nello, and Shawe Taylor. An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge: Cambridge UP, 2000. Print. Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Support\\_Vector\\_Machines&oldid=618](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Support_Vector_Machines&oldid=618)"

- 
- This page was last modified on 30 November 2017, at 18:08.
  - This page has been accessed 918 times.

# Teacher Scheduling

From CU Denver Optimization Student Wiki

## Contents

- 1 Class Scheduling through Linear Programming
- 2 Abstract
- 3 Model
- 4 AMPL Code for Model
- 5 Extensions

## Class Scheduling through Linear Programming

by Amber Rosacker

I worked on trying to create a very basic model for teacher scheduling as my final project for Linear Programming, Fall 2019.

## Abstract

## Model

## AMPL Code for Model

## Extensions

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Teacher\\_Scheduling&oldid=2271](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Teacher_Scheduling&oldid=2271)"

- This page was last modified on 27 November 2019, at 19:17.
- This page has been accessed 485 times.

# TeacherScheduling

From CU Denver Optimization Student Wiki

Teacher Scheduling by Amber Rosacker

I have worked on trying to create a very basic model for teacher scheduling as my final project for Linear Programming, Fall 2019.

## Contents

- 1 Abstract
- 2 Model
- 3 AMPL Code for Model
- 4 Extensions

**Abstract**

**Model**

**AMPL Code for Model**

**Extensions**

Retrieved from "<https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=TeacherScheduling&oldid=2191>"

- 
- This page was last modified on 13 November 2019, at 19:53.
  - This page has been accessed 381 times.

# The Circuit Less Travelled: A Path of Gentrification Through Denver Neighborhoods

From CU Denver Optimization Student Wiki

Hey! Welcome to Angela Morrison and Weston Grewe's page on gentrification.

## Abstract

Many factors shape us into the people we are today. Among these factors are the places and people we grew up with or still live with today. The neighborhood block parties, kids playing together, and the businesses and schools that were within walking distance played a crucial role in your growth and development. All of these helped create a unique community and culture within your neighborhood that arose naturally as people grew up, moved out, and new families moved in. A common question that arises with this is “What does natural growth and development look like?” A disruption of organic growth is what many refer to as gentrification. By introducing new, market-rate housing and luxurious amenities such as high-end restaurants and new parks, the price of rent can increase and inevitably force local businesses and longtime residents who can no longer afford space in the neighborhood to leave.

We build a model to understand how the neighborhoods of Denver are changing over time. This allows us to predict where gentrification is happening in Denver and where people move when they are displaced. Using data collected in the American Community Surveys over the last 15 years to derive a clustering of neighborhoods based on factors that are associated with gentrification, such as average income, demographic breakdown, and level of education. We construct a so-called circuit walk between these clusters which yields the changes in clusters in smaller time steps than what is possible by looking at just the data sets alone. Using these smaller steps, we can identify when a neighborhood might be susceptible to outside forces making changes to the community. This allows us to make target policy recommendations such as a tax on vacant properties, targeted redistribution of community funds for neighborhood improvements, or an increase in affordable housing construction, to help preserve the natural growth and development of neighborhoods.

## Helpful Project Links

GitHub (<https://github.com/wgrewe/D2P-Optimization-Fall-2021>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=The\\_Circuit\\_Less\\_Travelled:\\_A\\_Path\\_of\\_Gentrification\\_Through\\_Denver\\_Neighborhoods&oldid=3528](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=The_Circuit_Less_Travelled:_A_Path_of_Gentrification_Through_Denver_Neighborhoods&oldid=3528)"

- 
- This page was last modified on 6 December 2021, at 22:43.
  - This page has been accessed 514 times.

# The amazing Fourier, progenitor of linear systems

From CU Denver Optimization Student Wiki

Most often the subject of linear programming is presented by citing a number of apparently disconnected examples with little or nothing in common except that their mathematical form embodies a system of linear inequalities that optimizes (minimizes or maximizes) the objective function. Conceptually linear programming is concerned with building a model for describing the interrelations of the components of a system, which researchers have been doing since as early as the 1700s. Its approach to model building has applications to a broad class of decision problems. Next we look at how linear programming made its debut.

Joseph Fourier (1768-1830) was one of the earliest pioneers of LP. The extent of his conceptual understanding is well illustrated by the following passage, where he described the general method of solving a problem in three variables:

". . . the system of all these planes forms a vessel which serves them as the limit of an envelope. The figure of this extreme vessel is that of a poly-hedron on which the convexity is turned towards the horizontal plane. The lower point of the vessel or polyhedron has for ordinates the values X, Y, Z which are the object of the question: that is to say, that Z is the smallest possible value of the greater variation, and that X and Y are the values of x and y proper to give this minimum . . ." [italics inserted].

Founrer's method. In order to demonstrate Fourier's method we will consider an LP model in a standard form as a maximization subject to  $\leq$  constraints. Clearly any model can be converted into this standard form. When we try to solve an LP one of three possibilities results:

1. The model is infeasible, i.e., there are no values for the variables which satisfy all constraints simultaneously. 2. The model is unbounded, i.e., the value of the objective function can be increased without limit by choosing values for the variables. 3. The model is solvable, i.e., there exists a set of values for the variables giving a finite optimal value to the objective function.

Although case (3) applies to our illustrative numerical example, it will be in the method how cases (1) and (2) manifest themselves. In order to demonstrate the method we will use the model P above. Since we wish to maximize  $-4x_1 + 5x_2 + 3x_3$  as well as solve the inequalities we will consider the model in the form: Maximise z subject to:

4x1 - 5x2 - 3x3 + z	≤	C0
- x1 + x2 -x3	≤	C1
x1+ x2 + 2x3	≤	C2
-x1	≤	C3
-x2	≤	C4
-x3	≤	C5

Constraint C0 is really a way of saying we wish to maximize  $z \leq -4x_1 + 5x_2 + 3x_3$  which nothing more than maximizing the value of the objective function.

Fourier gives a method of eliminating variables from inequalities which results in the transformed model P2:

Maximise z subject to:

-x2 -7x3 +z < 8	C0 + 4C1
-5x2 -3x3 +z < 0	C0 + 4C3
2x2 + x3 < 5	C1 + C2

x2 +2x3	< 3	C2 + C3
- x2	< 0	C4
- x3	< 0	C5

Where the origins of the combined constraints are indicated. We also note that there can be variations on this method.

It was around 1760 that economists first began to describe economic systems in mathematical terms. For the most part, mathematical economists made use of general functions the parameters of which were as a rule unspecified. It is only fair that we should out in their defense that very little factual information was available on income, quantities of production, productive capacity, consumption, business investments, savings, distribution, etc., which reduced the purpose of the mathematical equations to an attempt at describing and analyzing in a qualitative rather than a quantitative manner the perceived interrelations within a system while the manipulation of equations corresponded to logical deductions from the assumptions. High-scale practical mathematical models to describe an economy came about in the 1930's as a result of the great depression; it was also the debut of the "New Deal," which was a serious attempt on the part of the government to support certain activities with the intention of speeding up recovery.

Wasslly Leontief, professor at Harvard, brought out his book “The Structure of the American Economy” during this period. It was largely different from earlier model building approaches because a large number of industries (or sectors) were interrelated and actually interdependent, which constituted a single model that allowed realistic estimation of the parameters in a quantitative way. Each sector was identified as representing a number of related functions that were “summed up together and called an industry, such as the food, steel, energy industry and so on. The following yeas saw a generalization of this approach, which allowed it to accommodate dynamic and complicated Air Force applications.

“About the same time that Leontief was producing his model, John von Neumann, world—famous mathematician, published his "Model of Economic Equilibrium," (1937). His model (like the Leontief model) was a linear programming model. It was more general, as it allowed for alternative activities”.

There appeared to have been an insignificant amount of interest in the model itself among economists for, the reason for which was explained by T. C. Koopmans in the following way:"To many economists, the term linearity is associated with narrowness, restrictiveness and inflexibility of hypothesis. However, it was the broad applicability of the linear programming approach to military logistics and planning that stimulated economists during the post war period to recognize the potentialities of this type of model in studying economic problems.

It was once possible for a supreme commander to plan operations personally. As the planning problem expanded in general complexity, the inherent limitations in the capacity of any one individual were becoming increasingly a barrier. Military histories are filled with instances of commanders who failed because they did not take into account some detail, not because they were essentially incompetent and could not have eventually have considered all the relevant details, but because they could not master all the details fast enough in a humanly possible way.

In the US Air Force it became unavoidably clear to members of this organization that coordinating quickly and well the vast energies and complex resources of whole nations in a total war required no less than effective scientific programming techniques. Undoubtedly this need had risen many times in the past but this time the means of accomplishment were at hand and those means were maturing quickly, that is linear programming.

SCOOP" (Scientific Computation of Optimum Programs) was an intensive-work project that had begun in 1947. At first, the idea was to rely heavily on the use a linear programming model to develop robust Air Force programs, but it had not been long before it was recognized that even the most optimistic estimates of the efficiency of future computing procedures and equipment would not be adequate to prepare effectively detailed Air Force programs. Therefore ,in the spring of 1948 this issue led to the commencing of a second SCOOP proposal which aimed to develop special linear programming models, referred to as triangular models, that would parallel the stepwise staff procedure.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=The\\_amazing\\_Fourier,\\_progenitor\\_of\\_linear\\_systems&oldid=978](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=The_amazing_Fourier,_progenitor_of_linear_systems&oldid=978)"

- This page was last modified on 7 December 2017, at 14:43.
- This page has been accessed 1,570 times.

# Using AMPL and Python to solve the cutting stock integer program

From CU Denver Optimization Student Wiki

AMPL is a modeling language that gives human readable code when solving optimization problems. An issue with AMPL is that there is no debugger in the compiler and while AMPL handles simple programs well, more complicated solving techniques can start to becoming challenging to code. Many of our usual coding languages such as Python, C, etc... have built in packages that allow you to code in the host language while implementing AMPL's solvers. In this project we explore 3 different ways to model the cutting stock problem: AMPL, Python with AMPL background, and Python. Below you will find the link to my code as well as slides used to present the material.

Github link and Presentation slides:Cutting Stock Code (<https://github.com/Crawfoni/Cutting-Stock>)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Using\\_AMPL\\_and\\_Python\\_to\\_solve\\_the\\_cutting\\_stock\\_integer\\_program&oldid=4298](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Using_AMPL_and_Python_to_solve_the_cutting_stock_integer_program&oldid=4298)"

- 
- This page was last modified on 13 April 2023, at 10:54.
  - This page has been accessed 49 times.



# Using After School Activities To Reduce Crime

From CU Denver Optimization Student Wiki

The purpose of this project is to develop a linear program that will identify potential sites to maximize effectiveness of after school programs for at risk teens in the Denver area. More specifically, we will examine the data identifying the location of crimes that are most often committed by teenagers for 2014 in Denver, as well as the locations of after school programs for that year. From this information, we will determine the optimal locations for additional after school programs for various integer amounts.

The data that will be used is provided on the Data to Policy Website. Using this information, feasible areas will be identified to suggest possible sites for after school programs. Using libraries associated with AMPL, we will investigate each feasible area to determine optimal locations for after school programs. The results will be compiled to provide policy suggestions at the Data to Policy Symposium held in Denver Colorado, in November of 2018.

## Contents

- 1 Motivation
- 2 The Linear Program
- 3 Implementation
- 4 Results
- 5 Poster
- 6 Possible Future Improvements
- 7 References

## Motivation

There have been studies done that show that after school programs reduce teenage crime <sup>[1]</sup> <sup>[2]</sup>. We were therefore interested in what would be the proper placement of after school programs to help reduce these crimes. In particular, we wanted to focus most closely on those crimes committed by teenagers, and so we focused just on theft/larceny, vandalism and alcohol related offenses. Furthermore, we wanted to place these afterschool programs so that they were far away from other after school programs as to help reach the largest number of teenagers as possible.

## The Linear Program

We use the linear program

$$\min \sum_{y \in Y} f(y) \left( \frac{\sum_{s \in S_y} d(y, s)^{-2} + \sum_{y' \in Y} f(y') d(y', y)^{-2}}{|S_y| + k - 1} - \frac{\sum_{c \in C_y} d(c, y)^{-2}}{|C_y|} \right)$$

$$\text{s.t } \sum_{y \in Y} f(y) = k, f(y) \in \{0, 1\},$$

where  $k$  is the number of after school programs we wish to select,  $S_y$  is the set of all locations of existing after school programs in the Denver area within 2 miles of point  $y$ ,  $C_y$  is the set of locations of crimes most often committed by teenagers in the Denver area within 2 miles of point  $y$ ,  $Y$  is the set of locations of possible after school programs,  $d(\cdot, \cdot)$  is a distance functions, and  $f$  is an indicator function to decide whether or not we pick a specific location. We use a special distance function for this linear program, rather than a strict Euclidean distance. Specifically, we if we have two points  $a = (x_1, y_1), b = (x_2, y_2)$  where the Euclidean distance is less than about 350 feet, about the size of a smaller city block, we fix  $d(a, b) \approx 350$ . We do this, because when picking an after school program to help prevent crime, it does not really matter whether that crime occurred 10 feet from the building or 100 feet. Furthermore, this will also avoid some very large values, which could skew our data, as we are dealing with reciprocals. We also only consider points within 2 miles of the possible after school programs. This is because after school program will likely not have an effect on teenagers who live far away, as they probably wouldn't even consider attending the program.

The first term in the series is the average of the reciprocals squared of how far the after school program is from other ones. This term includes both the after school programs already created as well as the ones we plan to implement, which is included in the  $f(y')$  term. This is to make sure that when adding all of the additional after school programs, we will not add two that are too close to each other. The larger this number is, the closer it is to other after school programs on average. The second part of this minimize function is the same thing, but with distances from crimes rather than other after school programs. Similar to before the larger this number is, the closer it is to crimes on average. Therefore we use a minimize, as we try to maximize distance from other after school programs, which corresponds to making the first term small, and minimize the distance to crimes, which corresponds to make the second term large.

The AMPL code is included below, first the model file and then the data file. The model file can be individually on github here: <https://github.com/toadchkjl/SchoolPoster/blob/master/SchoolProgramAMPL> The data file is here: <https://github.com/toadchkjl/SchoolPoster/blob/master/AfterSchoolAMPLData>

```
param N > 0; #number of possible programs
param K > 0; #number of after school programs we want

param avgDisC{1..N} > 0; #AVERAGE recipiricols of distance from crime
param disS{1..N} > 0; #SUM of recipricols of distance from after school programs
param numDisS{1..N} > 0; #number of after school programs considered using distance function
param innerDis{1..N, 1..N} >= 0; #distance between asp we're picking

var y{1..N} binary; #whether or not we pick school i

#Constraint function
minimize distances:
    sum{i in 1..N} (y[i] * ( (disS[i] + sum{j in 1..N} (y[j] * innerDis[i,j]) )/(numDisS[i] + K - 1)
    - avgDisC[i]) );
```

#makes sure we add the correct number of after school programs

```
subject to correctNumber:
    sum{i in 1..N} y[i] = K;

param N := 11;
param K := 3;

#All parameters pulled from excel spreadsheets

param: avgDisC disS numDisS :=
1 1.2852E-07 4.6709E-6 226
2 5.2598E-8 6.3672E-6 219
3 1.1447E-7 4.0132E-6 219
4 1.1868E-7 4.6461E-6 232
5 1.2810E-7 6.1081E-6 230
6 6.52293E-08 4.1059E-06 197
7 3.96687E-08 7.40798E-06 209
8 1.4143E-08 3.66676E-06 169
9 6.29126E-08 4.07824E-06 198
10 2.14668E-08 2.40147E-06 129
11 9.28353E-08 9.0258E-06 213;

param innerDis: 1 2 3 4 5 6 7 8 9 10 11 :=
1 0 8.38391E-08 7.83163E-07 8.98376E-06 7.67529E-08 1.37545E-08 2.42142E-08 4.0668E-09 1.39433E-08 1.67089E-09 1.62554E-07
2 8.38391E-08 0 1.79911E-07 9.18722E-08 4.13457E-08 8.83414E-09 1.02527E-08 2.7362E-09 9.11994E-09 1.29285E-09 2.917E-08
3 7.83163E-07 1.79911E-07 0 8.70121E-07 6.37104E-08 1.1558E-08 1.76669E-08 3.56013E-09 1.17962E-08 1.52686E-09 8.15912E-08
4 8.98376E-06 9.18722E-08 8.70121E-07 0 9.07506E-08 1.43081E-08 2.28441E-08 3.96488E-09 1.45545E-08 1.65897E-09 1.33112E-07
5 7.67529E-08 4.13457E-08 6.37104E-08 9.07506E-08 0 3.04406E-08 1.70859E-08 3.63768E-09 3.24152E-08 1.75282E-09 3.91868E-08
6 1.37545E-08 8.83414E-09 1.1558E-08 1.43081E-08 3.04406E-08 0 1.49998E-08 4.63979E-09 6.47392E-06 2.6296E-09 1.39916E-08
7 2.42142E-08 1.02527E-08 1.76669E-08 2.28441E-08 1.70859E-08 1.49998E-08 0 1.16401E-08 1.41756E-08 2.92557E-09 5.97507E-08
8 4.0668E-09 2.7362E-09 3.56013E-09 3.96488E-09 3.63768E-09 4.63979E-09 1.16401E-08 0 4.41387E-09 7.65004E-09 5.67739E-09
9 1.39433E-08 9.11994E-09 1.17962E-08 1.45545E-08 3.24152E-08 6.47392E-06 1.41756E-08 4.41387E-09 0 2.5306E-09 1.38052E-08
10 1.67089E-09 1.29285E-09 1.52686E-09 1.65897E-09 1.75282E-09 2.6296E-09 2.92557E-09 7.65004E-09 2.5306E-09 0 1.98329E-09
11 1.62554E-07 2.917E-08 8.15912E-08 1.33112E-07 3.91868E-08 1.39916E-08 5.97507E-08 5.67739E-09 1.38052E-08 1.98329E-09 0;
```

# Implementation

The data for this project was from the Data2policy website. Specifically, we pulled from two data sets, Afterschool Programs from the Denver Open Data Catalog<sup>[3]</sup> and Crime from the Denver Open Data Catalog<sup>[4]</sup>.

Initially the data was too large for our linear program. In order to reduce the size and focus on pertinent data points, we decided to use OpenRefine software. This allowed us to find the locations of crimes that were likely to be committed by teenagers. It also allowed us to focus the timeline of crimes to match the information we had about afterschool programs. This greatly reduced the size of the data from a size that was well into the hundred thousands to 8,665 points.

Our linear program examined specific sites that we selected based on high crime density and low after school program density. These sites can be seen in Figure 1 and Figure 2.

The location on the crime data was both in latitude and longitude coordinates as well as x,y coordinates for EPSG:102654 NAD 1983 StatePlane Colorado Central FIPS 0502 Feet, while the afterschool program data only had these x and y coordinates. Therefore we decided to use the x,y coordinate system for the project rather than latitude and longitude coordinates. This one done for a few reasons. The first of these being that we only had to change 11 coordinates, the feasible locations of the afterschool programs, to

these x and y coordinates, rather than changing the 629 afterschool program coordinates to latitude and longitude. This change was done using epsg.io, which allowed latitude and longitude to be plugged in and returned the x,y coordinates. The second reason the coordinate system was preferable was that its units were in feet, which made interpretation of the results, as well as the restrictions on the distance function, much easier to implement. Furthermore, as it was a plane that points were projected onto, Euclidean distances could be used.

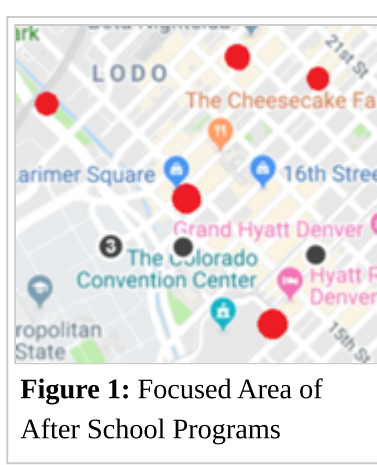
Although the linear program involves many sums, there are only a few parameters needed for the actual program. There is  $\frac{\sum_{c \in C_y} d(c,y)^{-2}}{|C_y|}$ ,  $|S_y|$ , and  $\sum_{s \in S} d(y,s)^{-2}$  for each  $y \in Y$ , and  $d(y,y')$  for each pair of  $y,y' \in Y$ . These parameters were all calculated on an excel spreadsheet and the manually entered into the AMPL. This was partially done because there are a relatively small number of points, which means that it is relatively easy to enter all of the parameters. This is especially true when compared to using Excel in connection with AMPL, which from earlier attempts, did not seem easy. However. it was also done this way as Excel is particularly good at handling conditional statements. This was of use when summing just over distances within two miles of the afterschool program in question, which was relatively easy with Excel. Since this is a binary linear program, we don't necessarily expect to have any data sizes consisting of many more than the 11 points we did consider. Therefore with the problem size, manually entering data that Excel calculated was the best option.

This is a quadratic binary linear program. We know however that any binary quadratic equation can be converted into a linear binary equation, with the introduction of a few additional variables. This is done automatically in AMPL. To solve the program we use the cplex solver, which works for binary linear programs.

## Results

On the first iteration, the results of Figure 3 were found for our focused region. Specifically, the locations are 1) 1947 Lawrence Street, Denver CO 80202 and 2) 700 14th Street, Denver CO 80202.

On the final iteration of the program, the one who's code is included above, we ran the program with 11 possible after school programs, and picked three of them. These 11 coordinates are shown in the table below with respect to the coordinate system EPSG:102654 NAD 1983 StatePlane Colorado Central FIPS 0502 Feet.



**Figure 1:** Focused Area of After School Programs

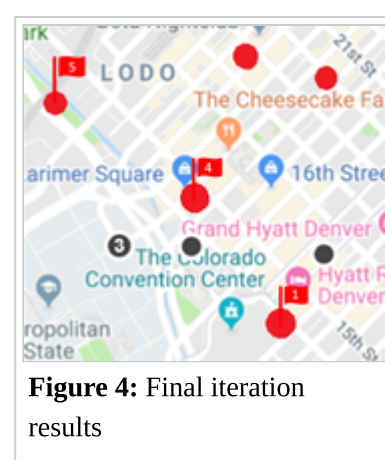


**Figure 2:** Focused Area of After School Programs



**Figure 3:** First iteration results

X coordinate	Y Coordinate
3142785	1699290
3139362	1699749
3141718	1699662
3142581	1699026
3142195	1695729
3145893	1691350
3149194	1698818
3158462	1698936
3145508	1691271
3165332	1689797
3145217	1699777



Our linear program found the locations 1, 4, and 5 were the optimal placements of the after school programs (See Figure 4).

With 11 points the implementation with AMPL and CPLEX took slightly less than one second to run. However, due to the exponential nature of binary linear programs, not many more additional points could be included before the program would take too long to run. However, not many groups looking to build after school programs would be examining more than a dozen or so locations, meaning that this binary linear program would still be feasible for many practical applications.

## Poster

The pdf of our poster is located on github and can be found here: <https://github.com/toadhkjl/SchoolPoster/blob/master/AfterSchoolPoster.pdf>

## Possible Future Improvements

On a macro scale, our project could be expanded to larger areas, like statewide programs. This would require an adjustment in both code and data files as it's assumptions and generalizations we made would become less valid. Using further research, we could also expand (or condense) the amount of crimes examined. As mentioned with the original project, if we were given additional specifications, we could scout the suggested location to find an actual property in which to offer the after school programs from.

It would also be possible to make the program more automatic, so that individual parameters would not need to be copied over from Excel. As we discussed in the section on implementation this made sense for the size of the program we were working on. If however, the program was much larger, it may be helpful to automate some things. The amount of variables that need to be entered is quadratic, which means that any increase in the number of possible afterschool programs, greatly increases the amount of manual entering done. Furthermore if the program was used for different cities, used multiple times, or if the data was updated frequently enough, some automation may be beneficial to reduce the amount of work.

In terms of the actual linear program, it would be possible to implement population data, if there exists a sufficiently high resolution population map for the areas of interest. This would make it possible to consider how many teenagers would possibly be affected by the building of the afterschool program, by examining how many live close to the program's possible location. This would help make the program even more beneficial for potential participants. It would also be possible to use budgetary constraints rather than constraining by the number of afterschool programs we wish to build. This seems like a good approach, as renting space in buildings downtown would be more expensive, but it also seemed as if this is where the linear program tended to concentrate the new afterschool programs. It is also much more realistic, as budgetary constraints would more often be the limiting factor in real applications.

## References

- ↑ Gottfredson, D. C., Gerstenblith, S. A., Soulé, D. A., Womer, S. C., & Lu, S. (2004). Do After School Programs Reduce Delinquency? *Prevention Science*, 5(4), 253-266. doi:10.1023/b:prev.0000045359.41696.02
- ↑ Taheri, S. A., & Welsh, B. C. (2015). After-School Programs for Delinquency Prevention. *Youth Violence and Juvenile Justice*, 14(3), 272-290. doi:10.1177/1541204014567542
- ↑ <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-afterschool-programs>
- ↑ <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-crime>

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Using\_After\_School\_Activities\_To\_Reduce\_Crime&oldid=1938"

- 
- This page was last modified on 6 December 2018, at 12:55.
  - This page has been accessed 11,648 times.

# Using Total Unimodularity to Solve a Warehouse Problem

From CU Denver Optimization Student Wiki

## Abstract

With a daily increase in the usage of online retailers like Amazon, the operations of warehouses are becoming more and more valuable. One way to create an efficient warehouse is to effectively place items that have yet to be shipped. The primary goal of this project is to develop a mathematical model that could create a scheme of inventory layout in a warehouse. From a mathematical standpoint, the original model I propose is an integer programming model. In order to solve any integer program of a reasonable size, one would need a significant amount of memory and a significant amount of time. A natural next step then is to relax the developed integer program for a more efficient solution. We do so by using a mathematical concept called "total unimodularity" to prove that the optimal vertex solutions of a linear programming relaxation are integral.

## Sources

[1] Cunningham, William H., and Geelen, James F.. "Integral Solutions of Linear Complementarity Problems." Mathematics of Operations Research, vol. 23, no. 1, 1998, pp. 61{68. 8 [2] Chandrasekaran, R. "Total Unimodularity of Matrices." SIAM Journal on Applied Mathematics, vol. 17, no. 6, 1969, pp. 1032{1034.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Using\\_Total\\_Unimodularity\\_to\\_Solve\\_a\\_Warehouse\\_Problem&oldid=2414](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Using_Total_Unimodularity_to_Solve_a_Warehouse_Problem&oldid=2414)"

- 
- This page was last modified on 2 December 2019, at 14:38.
  - This page has been accessed 776 times.

# Using Trees to Get Into College

From CU Denver Optimization Student Wiki

Hello! You have found the page for an integer programming project by Angela Morrison and Weston Grewe, to find other projects we have worked on just click on our names.

## Contents

- 1 Overview
- 2 Abstract
- 3 Mathematics
  - 3.1 Data Preparation
  - 3.2 Variables
  - 3.3 Constraints
  - 3.4 Objective Function
  - 3.5 Formulation
- 4 Implementation
  - 4.1 Example of Tree Topology
- 5 Methods
  - 5.1 Linear Relaxation and Cutting Planes
  - 5.2 Branch and Bound
- 6 Results
  - 6.1 Tree 1
  - 6.2 Tree 2
  - 6.3 Tree 3
  - 6.4 Tree 4
  - 6.5 Tree 5
- 7 Policy Recommendation
- 8 Resources
- 9 References



# Overview

College education has proved to be one of the safest bets to ensure a middle class future. There has been no shortage of ideas of how to increase a school's college enrollment rates. Ideas have ranged from increasing teacher pay, boosting the number of AP classes, shrinking class sizes, and the list goes on. The sheer number of solutions presented can confound knowledge of which solution works best. For this project, we develop an interpretable, optimal decision tree to understand the most important features for high college enrollment rates.

We draw data from Massachusetts's Public Schools <sup>[1]</sup>, a dataset which can be found on Kaggle and linked below. For this project, we substantially process the data to have it be readable for our AMPL program. The data processing can be found in a linked Jupyter Notebook.

We solve for an optimal decision tree using an integer program. We solve this integer program using AMPL along with implementing branch and bound and our own cutting planes and valid relaxations of variables.

## Abstract

With so much information out there about what students need to do in order to get into college, it can be hard to know which things a particular applicant should focus on. This is true for schools as well, but they are restricted due to budgets and other outside factors such as a global pandemic. One way to narrow down the important college application features is through a decision tree. Our project creates an interpretable, optimal decision tree based on features that have an impact on college enrollment after high school. This optimal decision produces a focused list of features which schools can work on adjusting or improving in order to make the biggest impact on their students' college enrollment. These features are determined via an integer program which maximizes the correct classification of schools which are both “successful” and “unsuccessful” in getting students to enroll in college directly after high school. For the purposes of this project, the benchmark for success is the average percentage of students that enroll in college after high school for a state in particular. One notable aspect is that these features can pertain to the specific dataset they originate from instead of a one-size fits all solution. This allows the model to take into account school community specific features as well as events such as COVID-19 which can impact the required college application materials.

## Mathematics

The model we implement is found in <sup>[2]</sup>. In short, the authors develop an integer programming based model to compute an optimal decision tree. For this model we need the following input:

- A tree structure specified through leaf nodes, branch nodes, and their connections to each other
- A labeled categorical data set

## Data Preparation

Our program requires each category of data to be broken into binary variables. Thus, we have a collection of feature groups  $G$  where every binary feature  $j$  belongs to a unique group.

For example, if we have categories Hair\_Color = {black, brown, blonde, white} and Shoe\_Type = {sandal, sneaker, dress} then this can be broken into a set of 7 binary features: Hair\_Color\_Black = {0, 1}, Hair\_Color\_Brown = {0, 1}, Hair\_Color\_Blonde = {0, 1}, Hair\_Color\_White = {0, 1}, Shoe\_Type\_Sandal = {0, 1}, Shoe\_Type\_Sneaker = {0, 1}, Shoe\_Type\_Dress = {0, 1}. An entry of 1 indicates membership of that group and 0 indicates nonmembership. So a person with blonde hair and sandals can be represented with the vector  $(0, 0, 1, 0, 1, 0, 0)$ . The first four features belong to the first feature group and the last 3 features belong to the second feature group.

## Variables

We will define three classes of variables. First, for each leaf node  $b$  and each sample  $i$ , we define the binary variable  $c_b^i$  where  $c_b^i = 1$  if sample  $i$  is assigned to leaf node  $b$  and 0 otherwise. The second group of variables we define is  $v_g^k$  for each feature group  $g$  and branch node  $k$ . The variable  $v_g^k = 1$  if feature group  $g$  is selected for branching at node  $k$  and 0 otherwise. Finally, we define the variables  $z_j^k$  where  $z_j^k = 1$  if feature  $j$  is selected to branch left at node  $k$  and zero otherwise.

## Constraints

To ensure that our result yields a decision tree, we will need to enforce constraints. First, at every branch node, exactly 1 feature group must be selected for branching. Therefore, we need the constraint  $\sum_{g \in G} v_g^k = 1$  for all  $k \in K$ . Next, for each branch node, only features from the selected feature group may be selected for branching left, therefore, we also have  $z_j^k \leq v_{g(j)}^k$  for each  $k \in K$  and where  $g(j)$  indicates the feature group which  $j$  belongs to.

Next, we will introduce constraints for the structure of the tree. Let  $a_j^i$  be the value of sample  $i$  at feature  $j$ . Define the function  $L(i, k) = \sum_{j \in J} a_j^i z_j^k$ . The function  $L(i, k)$  is binary, to see this, note that at node  $k$  exactly 1 feature group is selected, thus  $z_j^k = 0$  for all  $j$  not in the chosen feature group. By the structure of the binary data, there is only a single  $j$  in the feature group such that  $a_j^i = 1$ . Therefore, for each  $(i, k)$ ,  $L(i, k) \in \{0, 1\}$ . Also, we can see that  $L(i, k) = 1$  if and only if sample  $i$  branches (by convention) left. Consider a leaf node  $b$ , we can define the set  $K^L(b) \subseteq K$  of branch nodes that must branch left to reach leaf node  $b$ . We can make the conclusion that  $c_b^i \leq L(i, k)$  for all  $i \in I, k \in K^L(b)$ . This constraint accounts for the fact that if  $c_b^i = 1$  then it must be that it branched left at the correct branch nodes. We can define similar sets and constraints  $c_b^i \leq R(i, k)$  for all  $i \in I, k \in K^R(b)$ .

For our final constraint, we want to ensure that each sample is assigned to exactly 1 node. To do this we only need the constraint  $\sum_{b \in B} c_b^i = 1$  for all  $i \in I$ .

## Objective Function

For our objective function, we will maximize a weighted sum of correct classifications. Consider the following function  $\sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i$ . In this formulation,  $I_+$  ( $I_-$ ) correspond to the set of positive (negative) samples and  $B_+$  ( $B_-$ ) correspond to the set of positive (negative) leaf nodes. The constant  $C > 0$  allows for control between precision and recall. A large  $C$  implies a high precision and a smaller  $C$  allows for a greater recall.

## Formulation

constraints and the objective function to give the formulation of the problem as

$$\begin{aligned}
& \max \sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i \\
& \text{such that } \sum_{g \in G} v_g^k = 1 \text{ for all } k \in K \\
& z_j^k \leq v_{g(j)}^k \text{ for all } j \in J, k \in K \\
& c_b^i \leq L(i, k) \text{ for all } i \in I, b \in B, k \in K^L(b) \\
& c_b^i \leq R(i, k) \text{ for all } i \in I, b \in B, k \in K^R(b) \\
& \sum_{b \in B} c_b^i = 1 \text{ for all } i \in I \\
& z_j^k \in \{0, 1\} \text{ for all } j \in J, k \in K \\
& v_g^k \in \{0, 1\} \text{ for all } g \in G, k \in K \\
& c_b^i \in \{0, 1\} \text{ for all } b \in B, i \in I
\end{aligned}$$

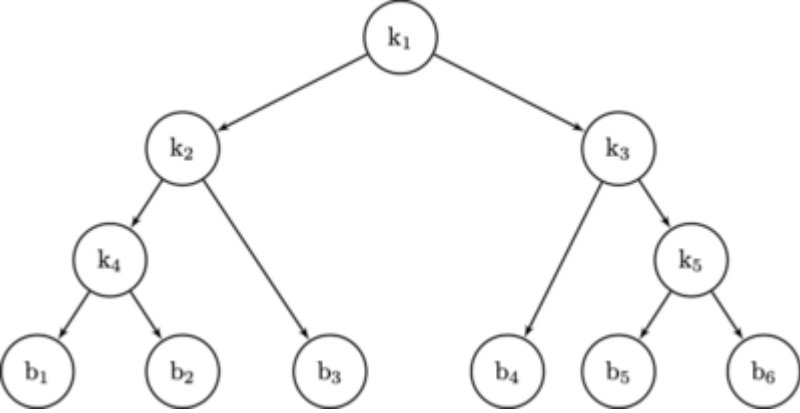
## Implementation

A detailed notebook and AMPL model can be found on our GitHub Repository. (<https://github.com/DillWithIt77/D2P-Spring-2021>) We began with a dataset from Kaggle containing data from Massachusetts public schools in 2017. This dataset contains data from all public schools in Massachusetts and contains numerical and categorical features. We used Python with Jupyter Notebooks to prep the dataset for the model we used. We first selected only data for only schools serving 12th graders. Next, we removed columns that contained features that we are not interested in studying. We then converted all data into a collection of binary features. This conversion is a two step process, we first converted numerical data to categorical data through bucketing, second, we converted all categorical features to groups of binary features where a value 1 indicates membership in a category and 0 otherwise.

The other half of the project was creating the AMPL model. Some constraints are straightforward to implement, others take a little more creativity. In this section, we will only discuss those constraints that are nontrivial to implement. First, in our parameters of the model, we had to define a set of leaf nodes and a set of branch nodes. In the model, we need access to the sets  $K_+(b)$ ,  $K_-(b)$  for each leaf node  $b$ . To compute these sets, we created parameters both dependent on the leaf nodes  $b_i$  and the branch nodes  $k_j$ . To account for the sets  $K_+(b)$  we had the  $(i, j)$  value as 1 if branch node  $j$  branches left to meet leaf node  $i$  and 0 otherwise. Parameters for the set  $K_-(b)$  is defined similarly. These parameters define the topology of the tree. Altering these parameters will alter the topology of the tree, this allows us to easily account for other topologies.

## Example of Tree Topology

For the decision tree in the image below



we have the branch nodes  $K = \{k_1, k_2, k_3, k_4, k_5\}$  and leaf nodes  $B = \{b_1, b_2, b_3, b_4, b_5, b_6\}$ . By convention, we assume that the nodes for the final branch are positive instances, that is  $B_+ = \{b_1, b_4, b_5\}$ , likewise  $B_- = \{b_2, b_3, b_6\}$ . We can also compute sets  $K^L(b)$ , for example,  $K^L(b_2) = \{k_1, k_2\}$  where as  $K^L(b_5) = \{k_5\}$  and  $K^L(b_6) = \emptyset$ .

In AMPL we can organize these sets as a table.

```

param leaf_nodes_left: k1 k2 k3 k4 k5 :=
b1 1 1 0 1 0
b2 1 1 0 0 0
b3 1 0 0 0 0
b4 0 0 1 0 0
b5 0 0 0 0 1
b6 0 0 0 0 0
  
```

When designing the AMPL program, this table can be used to check if a path for a leaf node must branch left at a branch node to reach the leaf node. In practice, it will be easier to have this table index over  $\{1, 2, 3, 4, 5\}$  and  $\{1, 2, 3, 4, 5, 6\}$ . For the program to run as desired, it will also be critical to decompose this table into two tables, one table for the positive leaf nodes and one table for the negative leaf nodes.

## Methods

To solve the integer program in an efficient manner, we will implement 3 methods: a linear relaxation of some variables, cutting planes, and a branch and bound strategy.

### Linear Relaxation and Cutting Planes

We do not need to require that the variables  $v_g^k$  be integral. The variables  $v_g^k$  are bounded below by 0. From our constraints, for each branch node  $k$ ,  $\sum_g v_g^k = 1$ . This bounds  $v_g^k$  from above by 1. Moreover, for each branch node  $k$ ,  $z_j^k \leq v_g^k, z_j^k \in \{0, 1\}$  and for each node at least 1  $z_j^k = 1$ . This forces one  $v_g^k = 1$  and by the sum constraint, the other  $v_g^k = 0$ . Therefore,  $v_g^k$  is integral, even when we relax it to a continuous variable.

Next, we will introduce a couple cutting planes. We will do so by working with the constraints  $c_b^i \leq L(i, k)$  for all  $i \in I, b \in B, k \in K^L(b)$ . For a given  $i \in I, k \in K$  we enforce that there is exactly one  $b \in B$  such that  $c_b^i = 1$ . Therefore, we have  $\sum_{\{b \in B: k \in K^L(b)\}} c_b^i \leq L(i, k)$ . This constraint is valid for all integer solutions, but cuts off some non-integral solutions. We can also build the similar constraint  $\sum_{\{b \in B: k \in K^R(b)\}} c_b^i \leq R(i, k)$ . This allows us to drop the constraints  $c_b^i \leq L(i, k), c_b^i \leq R(i, k)$  and replace them with these new constraints. Moreover, when  $k$  is the root node the constraint  $\sum_{b \in B} c_b^i \leq 1$  is implied, thus we may drop that constraint as well. With this new formulation, we are also allowed a second relaxation of our variables.

We also do not need to require that the variables  $c_b^i$  be integral. Since the only bounds on  $c_b^i$  are  $c_b^i \leq L(i, k)$  and  $c_b^i \leq R(i, k)$ , both  $L(i, k), R(i, k)$  are integral and we are maximizing a weighted sum, so it follows that for the maximum,  $c_b^i$  will be integral.

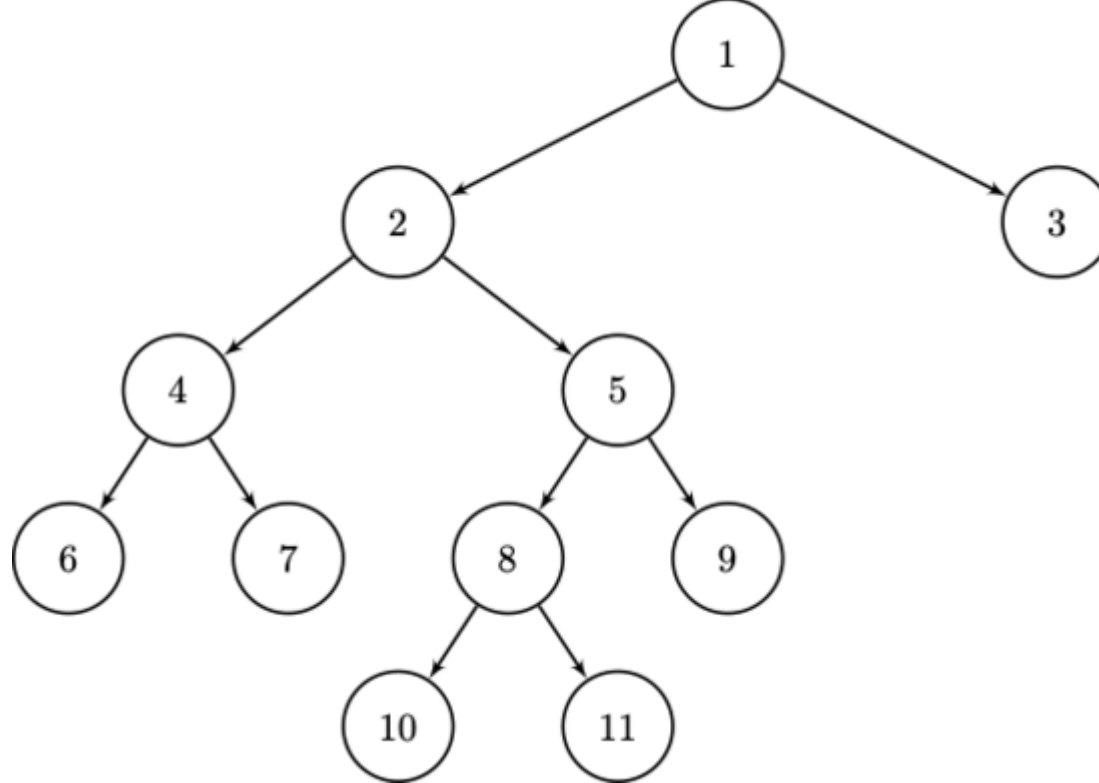
With these cutting planes, we can rewrite our program as follows:

$$\begin{aligned}
& \max \sum_{i \in I_+} \sum_{b \in B_+} c_b^i + C \sum_{i \in I_-} \sum_{b \in B_-} c_b^i \\
& \text{such that } \sum_{g \in G} v_g^k = 1 \text{ for all } k \in K \\
& z_j^k \leq v_{g(j)}^k \text{ for all } j \in J, k \in K \\
& \sum_{\{b \in B: k \in K^L(b)\}} c_b^i \leq L(i, k) \text{ for all } i \in I, k \in K \\
& \sum_{\{b \in B: k \in K^R(b)\}} c_b^i \leq R(i, k) \text{ for all } i \in I, k \in K \\
& z_j^k \in \{0, 1\} \text{ for all } j \in J, k \in K
\end{aligned}$$

## Branch and Bound

Finally, we implemented a branch and bound strategy. It is important to note that Tree 1 did not need any branch and bound, so none was done on this tree structure. Tree 2 was used as a small demonstration of implementing our own branch and bound method. Finally Tree 3, Tree 4, and Tree 5 were done using the options within the CPLEX solver. This is due to their sheer size and computation time necessary to solve these.

The branch and bound strategy for tree 2 begins with the tree below. Ultimately, more branching is required, this is just a subtree of the full branch and bound tree.



The following table indicates the branching decision at each node. By convention, if Feature  $n$  is selected for branch at node  $k$  then we branch left for  $n = 1$  and right otherwise.

**Branching Strategy (Tree 2)**

Branch Node	Feature	Subproblem Evaluation	Integral
1	Feature 5	423	No
2	Feature 19	161	No
3	Feature 6	278.375	No
4	Feature 53	15.5	No
5	Feature 20	147.499	No
6	N/A	7	Yes
7	N/A	2	Yes
8	Feature 35	103	No
9	Feature 16	52.75	No
10	N/A	6	Yes
11	N/A	94	Yes

The following pseudocode was used to construct this branch and bound tree:

```
Solve original problem:
  if integral:
    end
  else:
    using branching method from below.
Pull subproblem from search set.
Solve the subproblem via an LP relaxation.
Check solution:
  if integral:
    updated global lower bound with cost function of optimal solution
  if non-integral:
    if feature has been used for branching on earlier branch or all samples have same feature value:
      branch on next feature in vector
    else:
      branch on feature earliest in the vector
add new subproblems to search set.
if search set empty:
  end
```

Only a small portion of the branch and bounding was done the long way for this tree structure because, according to the CPLEX solver, the final tree should have around 47 nodes. This was a bit too much to complete in the time span of this project, but future work could be done to finish this strategy and compare to the CPLEX solver output.

We now take a look at the branch and bounding done for Tree 3, Tree 4, and Tree 5. This strategy was built off of options that already exist within CPLEX. For our project we tested a few different option combinations in order to gain a better understanding of what CPLEX is doing when using these parameters and how that can affect computation time of our program.

The parameters that were changed for the CPLEX solver for this project were as follows:

- branch : determines the direction one branches on the selected fractional variable
- cutpass : determines how many cutting planes are used while solving the source node in branch and bound
- display : allows information to be displayed while solving the problem
- heurfreq : directive to specify the frequency with which the solver applies a heuristic at the nodes
- mipcuts : determines if cuts are used for solving the main problem and subproblems
- mipdisplay : decides what gets reported to the screen while solving the problem
- mipinterval : controls the frequency of node logging
- presolve : determines if the solver applies presolve during preprocessing
- varsel : determines how the solver chooses a fractional-valued variable to branch on

These parameters are what allow us to determine the branching and bounding scheme to a certain extent that CPLEX uses while trying to solve our problem. In this section we will report the number of nodes needed to solve the problem for certain variable settings as well as explain what the output displayed by AMPL means in terms of the problem.

It is important to note that many of the parameters listed remain the same for all cases due to the nature of what they mean. these are the parameters and their values that do not change throughout these different runs of the CPLEX solver (see <sup>[3]</sup> for more in-depth meanings of these parameters):

- cutpass = -1
- display = 1
- heurfreq = -1
- mipcuts = -1
- mipdisplay = 3
- mipinterval = 1
- presolve = 0

The parameters that we adjust for testing are "branch" and "varsel". Branching has values that indicate it is either branching up which means we go down the path such that  $x_i \geq \lceil x_i \rceil$  or down which means we follow the path such that  $x_i \leq \lfloor x_i \rfloor$ . For "varsel" there are three choices: selecting a variable with the smallest integer feasibility, selecting the variable with the largest integer feasibility, or strong branching. The variable with the largest integer feasibility means that the variable we are branching on is closest to 0.5 since we are using binary variables. Similarly, the variable with the largest integer feasibility is one that is farthest from 0.5. Strong branching, as you may be familiar with, means that the solver will select the variable to branch on based on which gives the best improvement to the objective function before branching. With these things in mind, here are the tables for Tree 3, Tree 4, and Tree 5.

Branching Outcomes (Tree 3)			
branch	varsel	Number of Nodes (without including cutting plane constraints)	Number of Nodes (including cutting plane constraints)
Up	Variable with Smallest Integer Feasibility	38,468	39,101
Up	Variable with Largest Integer Feasibility	7,128	9,356
Up	Strong Branching	3,194	1,681
Down	Variable with Smallest Integer Feasibility	115,641	166,909
Down	Variable with Largest Integer Feasibility	12,407	15,403
Down	Strong Branching	778	845

Branching Outcomes (Tree 4)			
branch	varsel	Number of Nodes (without including cutting plane constraints)	Number of Nodes (including cutting plane constraints)
Up	Variable with Smallest Integer Feasibility	unable to run	unable to run
Up	Variable with Largest Integer Feasibility	unable to run	unable to run
Up	Strong Branching	13,608	11,405
Down	Variable with Smallest Integer Feasibility	unable to run	unable to run
Down	Variable with Largest Integer Feasibility	unable to run	unable to run
Typesetting math: 100% hinged		26,381	4,624



Branching Outcomes (Tree 5)			
branch	varsel	Number of Nodes (without including cutting plane constraints)	Number of Nodes (including cutting plane constraints)
Up	Variable with Smallest Integer Feasibility	unable to run	unable to run
Up	Variable with Largest Integer Feasibility	unable to run	unable to run
Up	Strong Branching	84,619	75,627
Down	Variable with Smallest Integer Feasibility	unable to run	unable to run
Down	Variable with Largest Integer Feasibility	unable to run	unable to run
Down	Strong Branching	146,758	17,4981

When getting these results, the AMPL should output something that looks similar to the following image. The important pieces of information (see <sup>[4]</sup> for more details) in this are as follows:

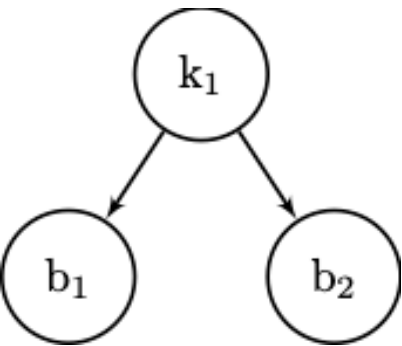
- Node Column: indicates which subproblem node is being solved at the moment
- Nodes Left Column: indicates how many nodes remain in the search set
- IInf Column: gives the number of non-integer values in the solution vector
- Cuts/Best Bound Column: explains what the "best" possible objective function is based on the current global minimum.
- Asterisk next to Node Column: indicates that this node had a feasible integer solution

Nodes		Objective	IInf	Best Integer	Cuts/ Best Bound	ItCnt	Gap
Node	Left						
0	0	510.0000	8		510.0000	1884	
Detecting symmetries...							
0	2	510.0000	8		510.0000	1884	
Elapsed time = 0.13 sec. (107.48 ticks, tree = 0.02 MB)							
1	3	510.0000	7		510.0000	1969	
2	3	381.5227	25		510.0000	3182	
3	4	510.0000	8		510.0000	2023	
4	5	375.4627	25		510.0000	3360	
*	5	integral	0	368.3750	510.0000	3510	38.45%
Found incumbent of value 368.375000 after 0.22 sec. (242.59 ticks)							

## Results

We ran our program on 5 different tree structures. By convention, at each branch node, a positive answer will indicate branching left. For each of the following trees we will present a diagram of the tree and the question asked at each branch node.

## Tree 1

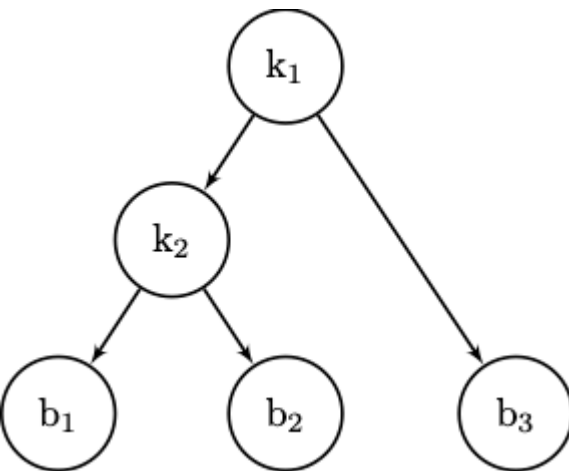


Branching Decisions

Branch Node	Branching Decision
$k_1$	2 AP Test 33-66% and 2 AP Test 66-100%

This feature means that of all the AP Test Takers at the schools, if 33-100% of those test takers took exactly 2 AP tests, it had an impact on if they enrolled in college after high school or not.

## Tree 2

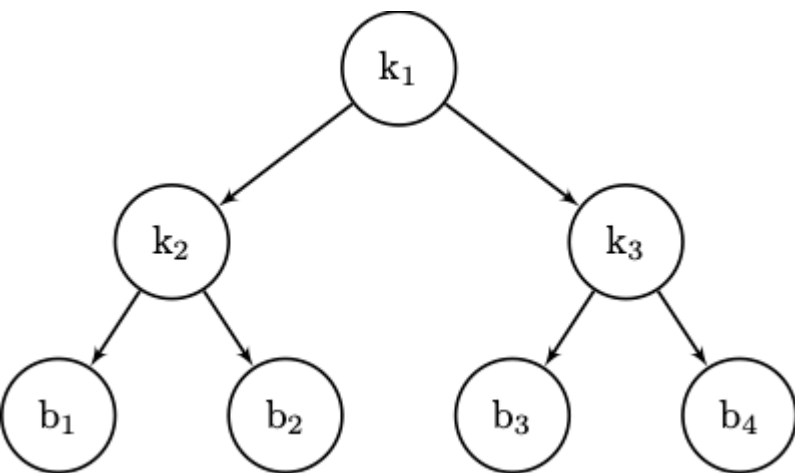


Branching Decisions

Branch Node	Branching Decision
$k_1$	First Language Not English 0-33%
$k_2$	2 AP Test 33-66% and 2 AP Test 66-100%

The first node states that if you are part of the 0-33% of the student population where English was not your first language, that had an impact on if you enrolled in college after high school. The second node is similar to that of Tree 1.

### Tree 3

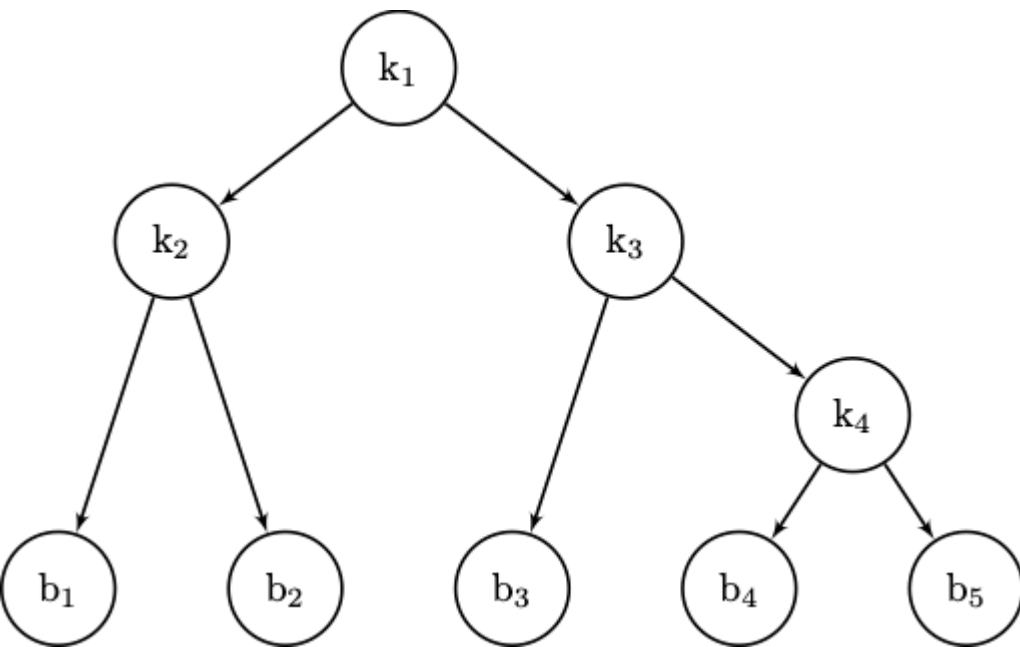


**Branching Decisions**

Branch Node	Branching Decision
$k_1$	1 AP Test Percent 0-33% and 1 AP Test Percent 66-100%
$k_2$	Total Enrollment 500-1000 and Total Enrollment 1500-2000
$k_3$	2 AP Test 33-66% and 2 AP Test 66-100%

For this tree the AP Test features are similar to the descriptions mentioned for Tree 1 and Tree 2. The enrollment feature is stating that schools with a total enrollment of between 500-1000 and 1500-2000 have an impact on if students enroll in college after high school.

# Tree 4

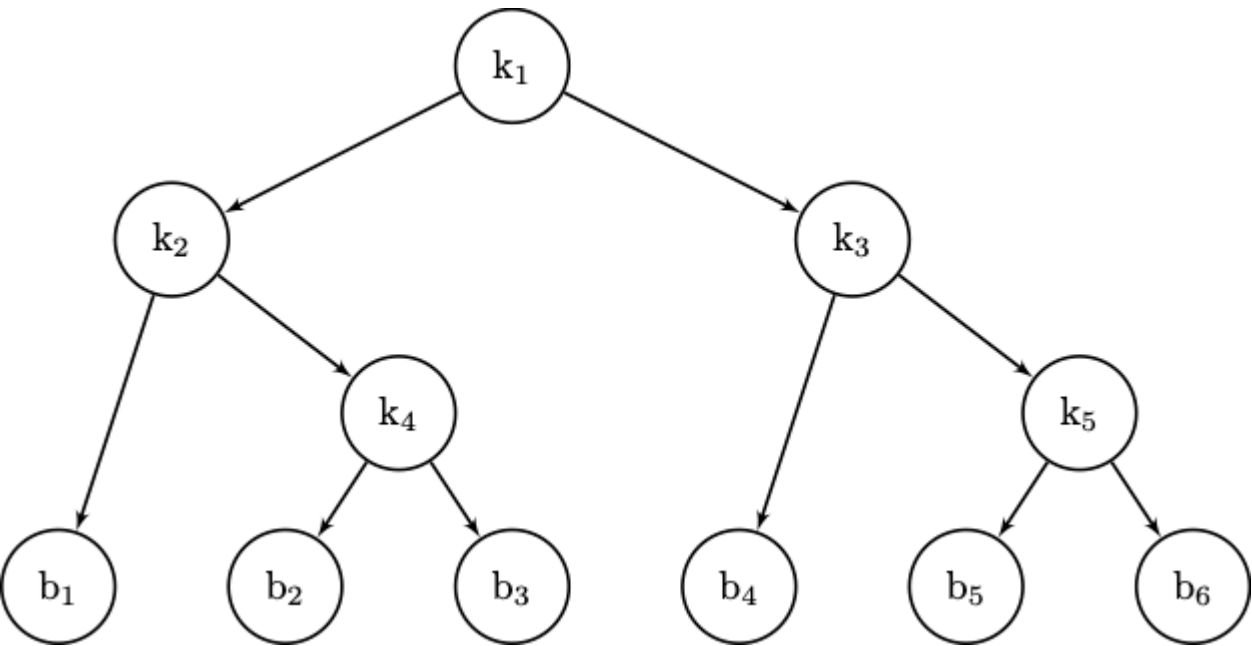


Branching Decisions

Branch Node	Branching Decision
$k_1$	Total Enrollment 500-1000
$k_2$	1 AP Test 33-66% and 1 AP Test 66-100%
$k_3$	Avg Class Size 20-25, Avg Class Size 25-30, and Avg Class Size 30-35
$k_4$	2 AP Test 66-100%

For this tree we see that Average Class Size now plays a part in whether or not students will enroll in college after high school. In particular if the average class size is between 20 and 35 (which is the maximum average class size for this sample of data).

Tree 5



Branching Decisions

Branch Node	Branching Decision
$k_1$	Total Enrollment 500-1000 and Total Enrollment 2500-3000
$k_2$	1 AP Test Percent 33-66% and 1 AP Test Percent 66-100%
$k_3$	Avg Class Size 20-25, Avg Class Size 25-30, and Avg Class Size 30-35
$k_4$	2 AP Test 33-66% and 2 AP Test 66-100%
$k_5$	2 AP Test 66-100%

This is just a culmination of the previous feature described in the other tree structures.

## Policy Recommendation

Our results show that schools that have many students who take 2 AP tests are likely to have high college enrollment rates. Even schools with many students taking just 1 AP test have a high college enrollment rate. Therefore, a multipronged strategy for increasing the number of students taking AP tests would be warranted. At the individual school level, schools can implement strategies to improve the number of students who take AP classes. This can come through special perks or just teacher encouragement. At the district level, districts can hire more teachers who are qualified to teach AP courses, this could also come with incentives. Districts can also direct funding towards waivers to pay for AP tests so that students who come from socioeconomically disadvantaged backgrounds are not put off by the cost of a test. At the state and federal level, officials should be making sure there is money available for schools and districts to implement this policy.

What is also interesting is what our results did not show. The number of students taking the SAT test was not used as a predictor. Recently, due to Covid-19, universities have been dropping their requirement of an SAT test. There has been sizable backlash for these decisions. However, our findings point to the fact that SAT test scores may not matter that much. It would be more important to focus on the AP test. For this reason, state and federal governments should not worry as much about the importance of the SAT test and instead focus more on the AP tests. In particular, for Covid-19 social distancing procedures, if only one test can be selected, it should be the AP test.

Other important features we see are class size and school enrollment. The fact that class size between 20-35 is selected lends itself to the idea that class size may not matter. Of course, this should be studied in more depth. There are large bodies of research arguing that class size matters. In our data it could be that the schools with smaller class sizes are special programs for students who do not find traditional school to be the right path. These students would also be unlikely to attend college. Due to this, expert knowledge of Massachusetts Public Schools would be useful.

## Resources

- GitHub Repository (<https://github.com/DillWithIt77/D2P-Spring-2021>)
- Kaggle Data (<https://www.kaggle.com/ndalziel/massachusetts-public-schools-data>)
- Model Paper ([http://www.optimization-online.org/DB\\_FILE/2018/01/6404.pdf](http://www.optimization-online.org/DB_FILE/2018/01/6404.pdf))

## References

1. ↑ [1] (<https://www.kaggle.com/ndalziel/massachusetts-public-schools-data>)<https://www.kaggle.com/ndalziel/massachusetts-public-schools-data>
2. ↑ [2] ([http://www.optimization-online.org/DB\\_FILE/2018/01/6404.pdf](http://www.optimization-online.org/DB_FILE/2018/01/6404.pdf))[http://www.optimization-online.org/DB\\_FILE/2018/01/6404.pdf](http://www.optimization-online.org/DB_FILE/2018/01/6404.pdf)
3. ↑ [3] (<https://www.ampl.com/BOOKLETS/amplcplex100userguide.pdf>)<https://www.ampl.com/BOOKLETS/amplcplex100userguide.pdf>
4. ↑ [4] (<https://www.ibm.com/docs/en/icos/12.8.0.0?topic=mip-progress-reports-interpreting-node-log>)<https://www.ibm.com/docs/en/icos/12.8.0.0?topic=mip-progress-reports-interpreting-node-log>

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Using\\_Trees\\_to\\_Get\\_Into\\_College&oldid=3338](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Using_Trees_to_Get_Into_College&oldid=3338)"

- This page was last modified on 8 August 2021, at 17:14.
- This page has been accessed 2,655 times.

# Vaccine Distribution

From CU Denver Optimization Student Wiki

## Contents

- 1 Abstract
- 2 Project Creators
- 3 The Project
  - 3.1 Motivation & Overview
  - 3.2 Data
    - 3.2.1 Non-Equity Data
    - 3.2.2 Equity Data
- 4 Integer Program
  - 4.1 Parameters
  - 4.2 Variables
  - 4.3 Objective Function
  - 4.4 Constraints
    - 4.4.1 Herd Immunity
    - 4.4.2 Non-Negativity
    - 4.4.3 Equity
    - 4.4.4 Supply
    - 4.4.5 Facility Cost
  - 4.5 Code
- 5 Results
- 6 Future Work
  - 6.1 Equity Constraints
  - 6.2 Non-Linear Programming
  - 6.3 Different Times
- 7 Links & Sources

## Abstract

Optimizing COVID-19 Vaccine Allocations in Denver

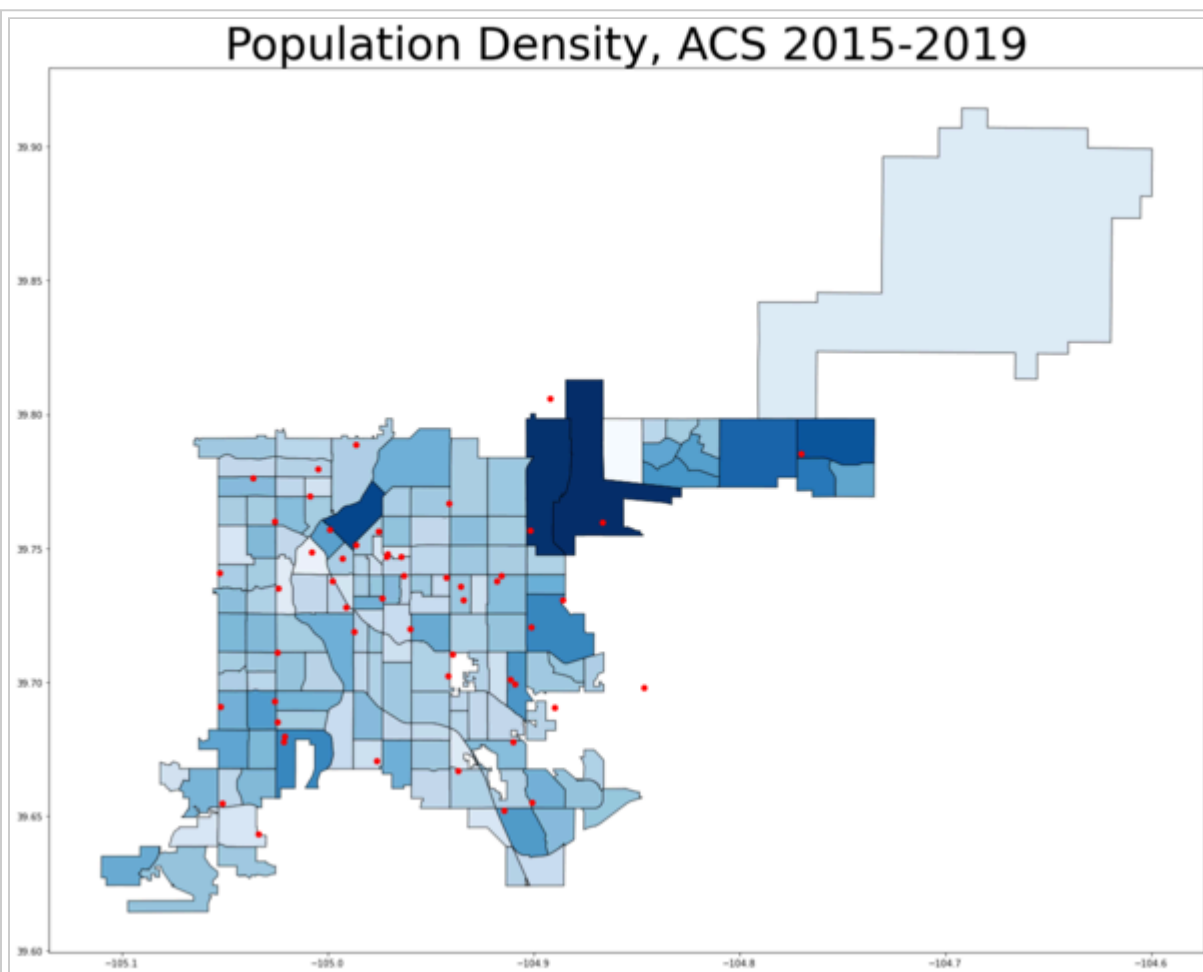
The world has been under the grip of the COVID-19 global pandemic for more than one year. With the advent of Sars-Cov-2 vaccines, the US needs to vaccinate as many people as possible to reduce risk of severe illness to immunocompromised populations, as well as to start the path forward to normalcy. In this project we investigate convex optimization as a means to improve vaccine allocation to the county of Denver in such a way that distribution is optimized for demand for the vaccine. We attempt to optimize

this allocation based on availability of service providers, their capacity, as well as by trying to ensure equitable distribution of vaccines amongst Denver’s disadvantaged and minority populations. Our hope is that by explicitly accounting for the needs of a given census tract, whether by considering its minority population or its proportion of unvaccinated, we can achieve herd immunity quickly and fairly.

## Project Creators

Michael Burgher, Collin Powell, and Sandra Robles.

## The Project



American Community Survey, 2015-2019. Vaccine Service Providers overlaid in red

## Motivation & Overview

Stating the importance of any efforts to fight a global pandemic are beyond obvious, and therefore unnecessary. We (nationally and globally) agree that it's a matter of utmost public importance to get as many people vaccinated against Sars-Cov-2 as fast as possible. Our interest, therefore, lies more in seeing if we provide an optimized vaccine allocation strategy in such a way that we achieve herd immunity as quickly as possible, while taking into account vulnerable communities.

Current vaccine allocation trickles downwards starting from the Federal government purchasing vaccines directly from vaccine providers. These doses get allocated to states based on the States' population size. In the case of Colorado, since the state makes up 1.69% of the country's population, it receives 1.69% of the available doses at any given time. Once the State has their allotted doses, it distributes them to Vaccine Service Providers within a given county based on a myriad set of rules that include county population size, size of community served by Service Provider, and committee recommendations.

We believe an Integer Programming approach to the allocation problem will yield interesting results that will likely differ from those advocated by the committee recommendations. This is especially true because any time decisions are made based on human recommendations, they will inherently be sub-optimal. So our motivation is to show we can provide an optimized strategy that still takes into account disadvantaged communities.



At this current point in time (mid Spring 2021), the allocation of vaccines has been streamlined significantly, so results from an IP problem would be less 'interesting' conceptually now. As such, we sought to optimize allocation at the beginning of the distribution, back in November/December 2020. At this point in time ("time zero"), vaccine supply was limited *and* demand was high. Not only that, but there was a need to distribute vaccines in such a way that overall risk of severe illness stemming from COVID was minimized as much as possible.

Our goal was to approach the problem from a social equity perspective. Meaning, not simply take into account the cost of vaccine distribution or of opening a service provider, but rather also the cost of **not vaccinating** disadvantaged subsets of the population that may have been missed by standard vaccine allocation methods. We will discuss this more in depth in the next few paragraphs. We believe it is important to have this equity mindset to ensure that under-privileged communities have access to a vaccine, since there is some proof that their wealthier non-minority counterparts are getting vaccinated at higher rates (<https://denverite.com/2021/01/22/wealthier-and-whiter-neighborhoods-in-denver-have-higher-vaccinations-rates/>).

## Data

We wanted to focus our vaccine allocation efforts to a manageable, yet still impactful, area. Hence, we chose the 144 census tracts in the county of Denver, Colorado. A census tract is a way to divide counties into sub-regions, with the boundary of a census tract meant to encompass around 4,000 inhabitants as defined on the census ([https://en.wikipedia.org/wiki/Census\\_tract](https://en.wikipedia.org/wiki/Census_tract)). Choosing census tracts over another geographic boundary method allows for easy visualizations and implementations due to its manageable size.

Next, we needed to define the possible service providers within county boundaries. These ran the gamut of hospitals, clinics, pharmacies, public mass vaccination events, and even a private company. The list of Service Providers were picked directly from the County of Denver's directory of available providers, with the assumption that these would have been available at time zero. They are displayed on the plot to the left, alongside population density. As can be seen, there's an interesting juxtaposition between population density and number of available providers, with some high density areas having very few nearby providers. There are also two providers that fall outside the boundaries for the County, but since they were marked as in-county providers by County officials, we are accepting them as part of the list of available providers.

## Non-Equity Data

The biggest data pieces were defining the census tracts & service providers (above) as well as deriving equity data (below). However, there were still other pieces of data to research.

The first one is, how many vaccine doses are available to the county of Denver for our purposes? The county has an approximate population of 700,000 individuals (<https://en.wikipedia.org/wiki/Denver>), so we decided that there were going to be 70,000 vaccine doses available, accounting for 10% of the population. We wanted to have enough doses to make the problem interesting (i.e. if Denver only got 1,000 doses, then the problem of allocating them becomes mostly null), while keeping the number low enough for the vaccines to still be considered scarce.

Next, we needed to know the general capacity at every service provider, as well as general cost to "open" a service provider. These were roughly estimated based on news articles and our own best guesses, and can be seen defined in the 'Data\_Denver\_Vaccination\_Sites.xlsx' file found in our github repository. We believe the most important aspect of these variables was that they were proportional to the service provider type. Meaning, a pharmacy needed to have a fraction of the capacity of a hospital, etc.

## Equity Data

In order to address the large number of factors which go into determining the equity of various areas, we are using the CDC's Social Vulnerability Index, or SVI. This is an index that the CDC has constructed and used for many years which takes into account several different variables regarding the population demographics of an area. The index itself is then given by a number which ranges from 0 to 1 with a higher number representing that a particular area is the most vulnerable to disaster, including a viral pandemic outbreak. As of 2018, the variables which went into the construction of this index include:

### ■ Socioeconomic Status

Below Poverty  
Unemployed  
Income  
No High School Diploma

### ■ Household Composition & Disability

Aged 65 or Older  
Aged 17 or Younger  
Civilian with Disability  
Single-Parent Households

### ■ Minority Status & Language

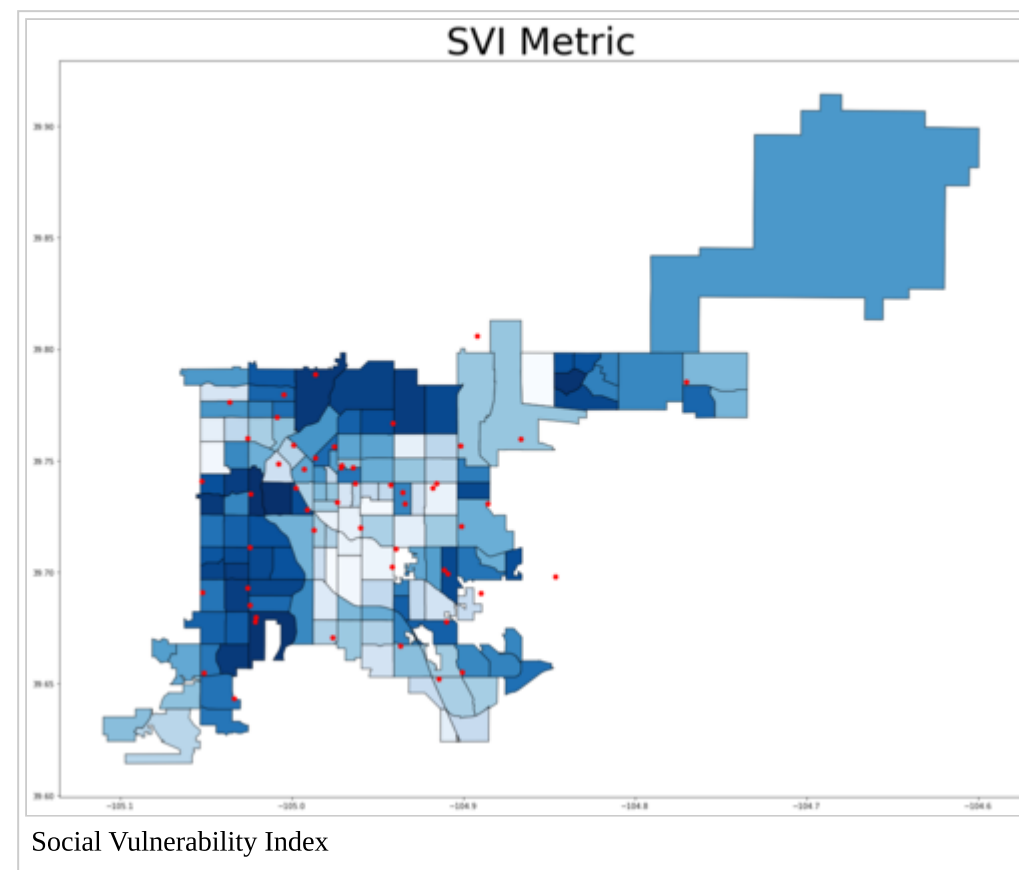
Minority  
Speaks English "Less than Well"

### ■ Housing Type & Transportation

Multi-Unit Structures  
Mobile Homes  
Crowding  
No Vehicle  
Group Quarters

## Newly added Variables in 2018

- ACS Estimates for persons without Health Insurance
- Estimate of Daytime Population



# Integer Program

## Parameters

List of  $j$  facilities and  $i$  census tracts

$p_i$  = population of tract  $i$

$v_i$  = number of vaccinated individuals in tract  $i$

$c_i$  = cost of vaccinating no one in tract  $i$ . Value equal to  $p_i(1 + SVI)(0.7 - \frac{v_i}{p_i})$

$d_{ij}$  = cost of delivering a vaccine to tract  $i$  from facility  $j$ . Value based on latitude/longitude distance from center of census tract to facility.

$f_j$  = minimum percent of max capacity of facility  $j$  for it to open

$m_j$  = maximum vaccines that facility  $j$  can distribute

## Variables

$y_j$  = 1 if facility  $j$  is open, 0 otherwise (binary)

$x_{ij}$  = number of vaccines delivered to tract  $i$  from facility  $j$

## Objective Function

$$\min \sum_i \sum_j (d_{ij} - \frac{c_i}{2p_i}) x_{ij}$$

## Constraints

### Herd Immunity

Once an area has reached 70% vaccinated, we do not get any more value by continuing to vaccinate that area while others are lower than 70%

$$\sum_j x_{ij} \leq 0.7p_i - v_i \text{ for all } i$$

## Non-Negativity

0 is the minimum amount of vaccines that any facility can distribute to any tract

$$x_{ij} \geq 0 \text{ for all } i, j$$

## Equity

This can be used to give a tract a minimum amount of vaccines in case certain areas are unintentionally being ignored or to ensure elderly, health care workers and other high priority individuals can get vaccinated right away.

$$\sum_j x_{ij} \geq b_i \text{ for all } i$$

## Supply

The total amount distributed must be less than the supply available

$$\sum_i \sum_j x_{ij} \leq S$$

## Facility Cost

The left inequality ensures that all open facilities will have at least  $f_j \in [0, 1]$  of its maximum capacity to open. The right inequality ensures a facility will not distribute more than its maximum capacity.

$$f_j m_j y_j \leq \sum_i x_{ij} \leq m_j y_j \text{ for all } j$$

## Code

```
##### Data Stes #####
set BLOCKS;
set FACILITIES;

##### Parameters #####
param pop {i in BLOCKS};
#population of block i

param val {i in BLOCKS};
#vale of the cost of vaccinating no one in block i
```

```

#min vaccines to be distributed to block i

param vac {i in BLOCKS};
#number of vaccinated individuals in block i

param mincap {j in FACILITIES};
#between 0 and 1 - min percent of max capacity for a facility to open

param maxcap {j in FACILITIES};
#maximum capacity of facility j

param dist {BLOCKS,FACILITIES};
#cost of distributing a vaccine to tract i from facility j

param supply;
#number of vaccines available

##### Variables #####

var X {i in BLOCKS, j in FACILITIES} integer;
#number of vaccines distributed from facility j to tract i

var Y {j in FACILITIES} binary;
#1 if facility j is open, 0 otherwise

##### Objective Function #####
minimize Cost: sum {i in BLOCKS} (sum {j in FACILITIES} ((dist[i,j]-val[i]/(2*pop[i]))*X[i,j]));
#minimizes the cost of the non-vaccinated individuals

##### Constraints #####

subject to Positivity {i in BLOCKS, j in FACILITIES}: X[i,j] >= 0;
#0 is the minimum vaccines a facility can distribute to a tract

subject to Supply: sum {i in BLOCKS} (sum {j in FACILITIES} X[i,j]) <= supply;
#Total vaccines distributed must be less than the available supply

subject to Population {i in BLOCKS}: sum {j in FACILITIES} (X[i,j]) <= 0.7*pop[i] - vac[i];
#Once Herd Immunity is reached, there is no additional value in vaccinating an area

subject to Equity {i in BLOCKS}: sum {j in FACILITIES} (X[i,j]) >= eq[i];
#Minimum amount of vaccines to be shipped to tract i

subject to MinCapacity {j in FACILITIES}: mincap[j]*maxcap[j]*Y[j] <= sum {i in BLOCKS} (X[i,j]);
#ratio of max capacity necessary to ensure a facility is open

subject to MaxCapacity {j in FACILITIES}: sum {i in BLOCKS} (X[i,j]) <= maxcap[j]*Y[j];
#no facility can distribute more than its max capacity

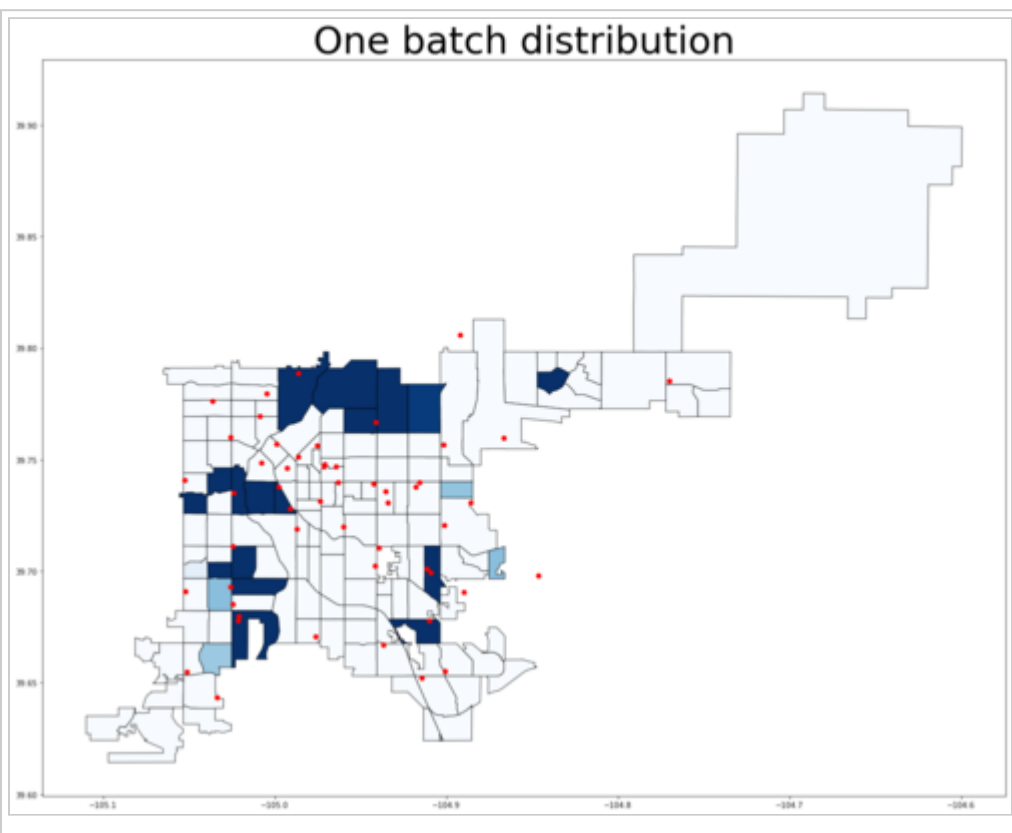
#reset; model IP_Project.mod; data IP_Project_Big.dat; option solver cplex; solve; display Y;
#display {i in BLOCKS} sum {j in FACILITIES} X[i,j]; display {i in BLOCKS} sum {j in FACILITIES} X[i,j]/pop[i]; display X;

```

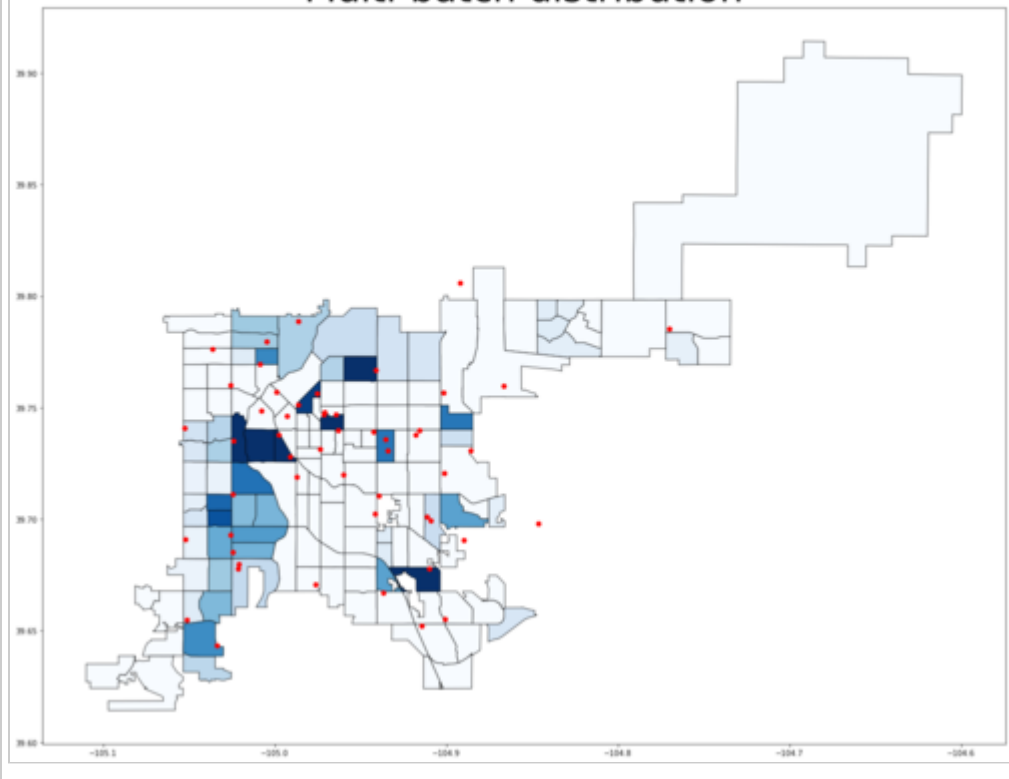
# Results

We will speak about the results using graphics, though the actual IP output can be found in the file 'Final Problem Solutions.xlsx' found within our github repository. As mentioned earlier, we ran the Integer Program in two different manners: assuming the 70,000 were available in one batch, and distributing that same quantity over 10 batches each having 1/10th of that (so, 7,000).

Our programs are clearly targeting the census tracts with the highest SVI first. This is most clearly seen with the single-batch program, whereby only a select handful of census tracts were allotted a majority of the vaccines, and these happened to be tracts with fairly high SVI. The multi-batch process still targeted vulnerable communities (as desired) but did so more spread out.



Multi-batch distribution



## Future Work

### Equity Constraints

The equity constraints (labeled  $b$  in the meta code and  $eq$  in the code) were not used for our initial runs. For future runs, these ideas could especially be important given the program seems to be suggesting a concentration strategy. These could also be used to ensure health care workers, elderly and other first priorities get vaccines.

### Non-Linear Programming

Currently, due to the linearity of our objective function, the program sees the same value for vaccinating the first and last individual in a certain area. We wish to design a program where as soon as the first individual is vaccinated, there is less value in vaccinating the second individual. Initial thoughts are that this could be solved with quadratic programming but there are likely unseen obstacles.

We ran our program iteratively in order to overcome this. We would distribute 10% of the supply through an iteration, then copy the results into excel and use the updated data for the next iteration. This was a complicated process so a better system (which seems doable though involves improving computer programming skills) would be nice going forward. This iterative process spread the vaccines out a bit more but would still return to partially vaccinated areas before hitting all unvaccinated areas. This has led us to conclude the program seems to be suggesting the concentration strategy.

## Different Times

For this project, the program was run at the beginning. This means there were 0 vaccines distributed anywhere before we started. It may be beneficial to run the program with certain areas already partially vaccinated.

## Links & Sources

**Github Repository:** <https://github.com/srobles09/COVIDvaccineAllocationIP2021>

Census Tract Data: <https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-american-community-survey-tracts-2015-2019>

Service Providers: <https://www.denvergov.org/Government/COVID-19-Information/Vaccination>

CDC Social Vulnerability Index: [https://www.atsdr.cdc.gov/placeandhealth/svi/documentation/SVI\\_documentation\\_2018.html](https://www.atsdr.cdc.gov/placeandhealth/svi/documentation/SVI_documentation_2018.html)

Current Allocation Methods: <https://www.cdc.gov/coronavirus/2019-ncov/vaccines/distributing.html> <https://covid19.colorado.gov/vaccine-providers>

Vaccine capacity by facility type: <https://www.cnn.com/2021/02/16/biden-administration-increases-weekly-covid-vaccine-shipments-to-states-and-pharmacies-.html>

<https://www.9news.com/article/news/local/next/covid-vaccine-doses-in-colorado-gone-to-waste-throw-away/73-0619d83e-b38c-474f-8c0d-b1254b588946>

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Vaccine\\_Distribution&oldid=3324](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Vaccine_Distribution&oldid=3324)"

- 
- This page was last modified on 4 May 2021, at 13:31.
  - This page has been accessed 2,640 times.



# Valentinas Sungaila

From CU Denver Optimization Student Wiki

I am a Masters Student in Statistics and Economics at CU Denver and plan to graduate Spring 2020. I got my bachelors degree in Economics from Western Washington University. I enjoy the outdoors and love trying new things.

Follow this link to see my Network Flows class project: [Separable Convex Cost Network Flow Problems](#)

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Valentinas\\_Sungaila&oldid=2749](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Valentinas_Sungaila&oldid=2749)"

Category: Contributors

- 
- This page was last modified on 4 May 2020, at 16:30.
  - This page has been accessed 907 times.

# Van der Waerden's Theorem

From CU Denver Optimization Student Wiki

Welcome to Amanda Ward's project page on van der Waerden's Theorem, created for the MATH 6406 final project.

## Abstract

This project explores van der Waerden's Theorem through a combination of classical combinatorics and computational experimentation. By combining visual aids, randomized and exhaustive search strategies, and walkthroughs of classical proofs, this project aims to offer an accessible and self-contained exploration of van der Waerden's Theorem, focusing on minimal cases to build intuition about the structure and complexity of problems in Ramsey theory, while emphasizing the power of computation in modern combinatorics.

Much of the project centers on the specific case  $W(2, 3)$ . The theorem asserts that for any positive integers  $r$  and  $k$ , there exists a number  $W(r, k)$  such that any  $r$ -coloring of the integers  $\{1, 2, \dots, W(r, k)\}$  will contain a monochromatic arithmetic progression (AP) of length  $k$ . This is a foundational result in Ramsey theory, and the goal of this notebook is to illustrate its implications, develop structural intuition, and validate key results through code-driven demonstrations.

We begin by examining a classical upper bound argument showing that  $W(2, 3) < 325$ . This proof divides the interval  $\{1, \dots, 325\}$  into 65 blocks of 5 elements each and applies the pigeonhole principle to show that repeated block colorings inevitably lead to a monochromatic 3-term arithmetic progression. The notebook mirrors this argument in code and highlights how overly generous the bound is in practice.

Next, we demonstrate that  $W(2, 3) > 8$  by constructing an explicitly defined 2-coloring of the set  $\{1, \dots, 8\}$ . By checking all arithmetic progressions within this coloring and visualizing the result using color-coded terminal output, we identify a valid counterexample that avoids 3-APs and thus establishes the lower bound.

To support the claim that  $W(2, 3) = 9$ , we then analyze randomly generated 2-colorings of the set  $\{1, \dots, 9\}$ . While this is not an exhaustive proof across all possible colorings, it offers strong empirical evidence that a monochromatic 3-AP almost always occurs, lending confidence to the minimality of the bound.

Although we do not attempt to confirm  $W(3, 3)$  (as the classical bound is far beyond feasible computational limits), we include demonstrations of lower bounds and random colorings for the 3-color case to show how these ideas naturally extend beyond the simplest case.

## Python Implementation Summary

Several reusable Python functions were developed to support this exploration, designed to be general enough for a variety of visualization and analysis tasks related to integer colorings:

- `analyze_repeated_block_AP`: simulates the pigeonhole-style argument by identifying repeated block colorings and constructing candidate APs from them
- `colordict`: formats a string with an ANSI color code to match a number's assigned color
- `colorname_dict`: returns the color name (e.g., "red", "blue") for a given key, useful in generating readable string output
- `display_coloring_with_AP`: prints the coloring in formatted blocks, optionally highlighting a chosen AP and showing block alignment

- `format_coloring` (used internally): arranges colored output in a row/column grid with block structure
- `get_monochromatic_APs`: returns a coloring (random or provided) and lists all monochromatic APs of a given length under that coloring
- `make_AP`: generates all arithmetic progressions of a given length within a specified interval

## External links

GitHub repository for this project (<https://github.com/amandaward1/van-der-Waerdens-Theorem>), contains the project notebooks and slide deack  
Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Van\\_der\\_Waerden%27s\\_Theorem&oldid=5036](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Van_der_Waerden%27s_Theorem&oldid=5036)"

- 
- This page was last modified on 14 May 2025, at 19:27.
  - This page has been accessed 29 times.

# Weston Grewe

From CU Denver Optimization Student Wiki

Hi! I am a second year PhD student at CU Denver. Currently, I research with Dr. Steffen Borgwardt where we study diameters of polyhedra, especially those arising from networks and totally-unimodular matrices. I also have a B.S. and M.S. in mathematics from Cal Poly, San Luis Obispo. In my undergraduate I worked on a research project studying the numerical range of matrices which provides a wonderful connection between linear algebra and classical algebraic geometry. During my master's I had a small project working on image registration and segmentation. Additionally, I wrote a thesis on homogenizing a model for cardiac tissue. Outside of school I enjoy most physical activities, in particular, I spend my free time running, biking, practicing yoga, and throwing frisbees. I also love to cook!

Other fun facts:

- My favorite board game (card game??) is Moose in the House.
- While I do not have a favorite bird, my least favorite bird is the Canadian Goose
- I believe in pineapple on pizza
- So far the best pizza I have found in Colorado is a tie between Humberto's Pizza Pub in Winter Park and High Mountain Pies in Leadville (please correct me if you think there is a better pizza)

I'm not a big fan of social media, but if you want sporadic updates on trivial parts of my life and would like to know how fast (or slow) I ran (or biked or hiked) on any particular day please follow me on Strava (<https://www.strava.com/athletes/66532954>).

## Projects

In the fall of 2020 for Linear Programming, I worked with Angela Morrison on Creating Fair Voting Districts. In this project, we use a clustering algorithm to design voting districts that are easy to understand and minimize the distance a voter has to travel to a polling location.

In the spring of 2021 for Integer Programming, I worked with Angela Morrison on Using Trees to Get Into College. For this project, we pulled used data from Massachusetts public schools to develop a decision tree to understand the most important features that lead to a student enrolling in college directly out of high school.

In the fall of 2021, Angela Morrison and I studied gentrification in the city of Denver using circuit walks in our project titled The Circuit Less Travelled: A Path of Gentrification Through Denver Neighborhoods.

In the spring of 2022, Angela Morrison and I (4/4!!!) analyzed bike paths in Denver using shortest path algorithms in A Wheelie Good Time: Safe Biking in Denver.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Weston\\_Grewe&oldid=3801](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Weston_Grewe&oldid=3801)"

Category: Contributors

- This page was last modified on 3 May 2022, at 08:35.
- This page has been accessed 1,157 times.

# Weston White

From CU Denver Optimization Student Wiki

After growing up in San Diego, CA, Weston went to CSUSM to earn a bachelor's in Economics. He continued his study of economics by attending CSU and earned a Master's there. After working for a few years, he decided to go back to school to study Applied Math.

## Projects

In the fall of 2023 for Linear Programming, I worked with Courtney Franzen on Housing Assistance Program Allocation.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Weston\\_White&oldid=4427](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Weston_White&oldid=4427)"

Category: Contributors

- 
- This page was last modified on 7 November 2023, at 12:35.
  - This page has been accessed 17 times.

# What are multicommodity flows

From CU Denver Optimization Student Wiki

There is currently no text in this page. You can search for this page title in other pages, or search the related logs ([https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Special:Log&page=What\\_are\\_multicommodity\\_flows](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Special:Log&page=What_are_multicommodity_flows)), but you do not have permission to create this page.

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php/What\\_are\\_multicommodity\\_flows](https://math.ucdenver.edu/~sborgwardt/wiki/index.php/What_are_multicommodity_flows)"

---

# Zachary Sorenson

From CU Denver Optimization Student Wiki

I started CU Denver in the Fall of 2018. I completed my undergraduate at Colorado School of Mines in Applied Math and I have my Master's in Math education. Outside of being a student and teaching, I enjoy playing soccer and watching movies.

For my first major project I investigated the how to optimally place after school programs to maximize effectiveness with Connor Mattes: Using After School Activities To Reduce Crime

For my second project, I investigated the quickest routes to access, and clean, parks in Denver with Connor Mattes: Cleaning Parks for a Safer Future

For my third project, I investigated the min-mean cancelling cycle and its applications with Connor Mattes: Min-mean Cycle Cancelling Algorithm and It's Applications

For my first readings course, I used Total Unimodularity to problem involving Warehouse Layouts: Using Total Unimodularity to Solve a Warehouse Problem

For my fourth project, I investigated the relationship between circuit walks and minimum weight matchings with Nicholas Crawford: Circuits and Bloom's Algorithm

Retrieved from "[https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Zachary\\_Sorenson&oldid=3453](https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Zachary_Sorenson&oldid=3453)"

Category: Contributors

- 
- This page was last modified on 28 November 2021, at 19:18.
  - This page has been accessed 2,291 times.



# Zane Showalter-Castorena

From CU Denver Optimization Student Wiki

I have a B.S in Applied Mathematics from University of Colorado Denver. In my free time I enjoy craft beer, reading, and programming. Personal projects include a 3D rendering of my brain, a 3D Ulam Spiral, and a 2D Ulam Spiral. Currently working on [http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Optimizing\\_Highschool\\_Graduation\\_Rates](http://math.ucdenver.edu/~sborgwardt/wiki/index.php/Optimizing_Highschool_Graduation_Rates)

Retrieved from "https://math.ucdenver.edu/~sborgwardt/wiki/index.php?title=Zane\_Showalter-Castorena&oldid=3086"

Category: Contributors

- 
- This page was last modified on 7 December 2020, at 14:00.
  - This page has been accessed 840 times.