Balanced Power Diagrams, Linear Programming, and Network Algorithms in Redistricting

Isaac Newell

May 31, 2018

Abstract

This paper examines the procedure of a paper by Philip Klein et al. entitled Balanced Power Diagrams for Redistricting. That paper applies a modified version of k-means clustering to districting so as to satisfy the equal population constraint. Their approach is an appealing application of computers to the incredibly complicated and easily exploitable practice of drawing district lines. It produces districts that are beautifully compact, convex, and equally populous without considering partisanship. Klein et al. formulate redistricting as an optimization problem, using linear programming and network flow algorithms. However, their paper is not easily accessible to readers unfamiliar with these topics. This paper thus aims to fill in the gaps, explaining the motivation, usage, and significance of these rich areas of computational theory.

1 Introduction

Every ten years, the U.S. government conducts a census to re-measure the demographics and distribution of the people. The year after the census is taken, the government then uses this data to draw new district lines [9]. To create the congressional boundaries, each state is allocated a certain number of seats proportional to its population. The states then create plans for themselves divide up into that number of districts [9]. There are a lot of factors that go into redistricting, and every state approaches the problem in a different way. The following are the most important metrics and constraints:

1. **Equal population**. This constraint follows the constitutional "one person, one vote" standard established in the 1960s. The Supreme Court case Wesberry v. Sanders in Georgia established that it is unconstitutional to make some districts significantly more populous than others, as such an unbalanced plan dilutes the votes of those in the more packed district [19]. This common-sense principle is necessary in modern districting plans, although perfect population equality is not really feasible. Different states have different standards for this, but all maintain roughly equal district populations [9].

2. **Compactness**. Compactness is a fairly intuitive geometric idea, yet a difficult problem to approach quantitatively. There exists no one true measure of compactness; rather we have many different measures that each behave differently, each with their own advantages and disadvantages. Methods can be split into three general categories: dispersion methods such as Reock, which measure the spread of the district, perimeter-based methods such as Polsby-Popper, which examine the contortedness of the boundary, and population-based methods which incorporate the district's demographics [9]. These measures will not explicitly be discussed in this paper. However, while they generally follow human intuition, different measures may disagree wildly on certain edge cases. Some states actually incorporate certain measures into their districting [9].

3. **Voting Rights Act**. In 1965, LBJ passed the monumental Voting Rights Act (VRA), which took comprehensive new steps to ensure robust minority representation [2]. Section 2 outlines the creation of majority-minority districts, which mandate consideration of racial polarization of voting outcomes in certain conditions [9]. Sections 4 and 5 deal with federal oversight of historically discriminatory states, which John Lewis describes as the "backbone" of the VRA [2]. However, Shelby County v. Holder (2013) ruled Section 4 unconstitutional,

rendering Section 5 useless [16]. Regardless, the VRA adds significant complication to the districting process which considers race and partisanship. Furthermore, Shaw v. Reno (1993) ruled that satisfying the VRA can't be overtly used as a guise for partisan gerrymandering [15].

4. **Competitiveness**. President Obama has stated that "no competition leads to more and more polarization in congress and it gets harder and harder to get things done" [5]. This is arguable, but it is indisputable that we are becoming more politically and geographically polarized and candidates are winning and losing by larger margins than ever. Some states emphasize this concern more than others, with Arizona being a major proponent of the standard [9].

5. **Communities of Interest**. What defines a community of interest is really quite vague. However, it makes intuitive sense that a district should represent some sort of cohesive group of people and should avoid splitting communities. California particularly cares about this concept [9].

6. **Electoral Outcomes**. The results of congressional elections are a function of both the actions of the voters and of the placement of the district lines. How does the districting plan effect the electoral result? This criterion makes intuitive sense to people; how does the fraction of votes a party gets translate to that party's share of the seats? While no such measure yet plays an explicit role in our legislature, there are several promising ones. Vieth v. Jubelirer (2004) ruled that partisan gerrymandering questions are not justiciable, but the ruling hinges on Kennedy's argument that no clear metric exists yet [17]. The efficiency gap, proposed only recently by Eric McGhee, purports to be that standard [12].

These considerations, although not directly pertinent to the mathematical focus of this paper, reveal an important core truth about redistricting: it is a complicated balance between many often-conflicting factors. Left to partisan humans looking to serve certain objectives by drawing the lines, this task can become quite problematic. Gerrymandering, made possible by clever computer algorithms, is now easy to do and very difficult to prove or prevent.

In their survey on rangevoting.org, Brian Olson and Warren Smith address the problems with representatives choosing their voters [13]. They discuss how many of our systems try to get around the problem of partisanship in counterproductive ways. For instance, nonpartisan commissions just become bipartisan commissions wherein both parties agree to keep the incumbents in power (incumbent or bipartisan gerrymandering). Requiring supermajorities to approve a plan (e.g. as Connecticut does) often have the same outcome. Furthermore, Olson and Smith point out that while we have lots of "feel-good" language in our judicial literature such as "districts should be compact," it doesn't actually mean anything quantitatively and thus accomplishes nothing.

Thus, it's easy to see why applying apartisan, objective, and optimized computer algorithms is such an appealing alternative in the districting problem. Olson and Smith recommend two different approaches: the "contest paradigm" and the "procedure-based paradigm". The former defines a clear, quantitative metric for how good a districting plan is, and people could thus propose plans from which the best would be chosen. The latter defines a clear, quantitative process for creating a districting plan, which produces (agnostic to partisan data) a good plan.

That's where Klein et al.'s paper comes in. Following the procedure-based paradigm, it provides a simple, well-defined way to create a good districting plan. See Figures 1 and 2. This paper seeks to explain how Klein's algorithm works, drawing upon the various areas of theory which it includes but doesn't itself explain. Specifically, it will examine power diagrams and Voronoi diagrams, constrained optimization and Linear programming, and how such problems are solved. Within those approaches, it will focus especially on that of Klein et al., the Goldberg-Tarjan algorithm. That explanation shall motivate some background in graph theory. Lastly, the paper will examine further research that can be done around these topics.

2 Power Diagrams

Klein et al.'s paper, Balanced Power Diagrams for Redistricting, unsurprisingly, applies balanced power diagrams to redistricting. But what is a power diagram? How is it defined mathematically, and how do we find them?

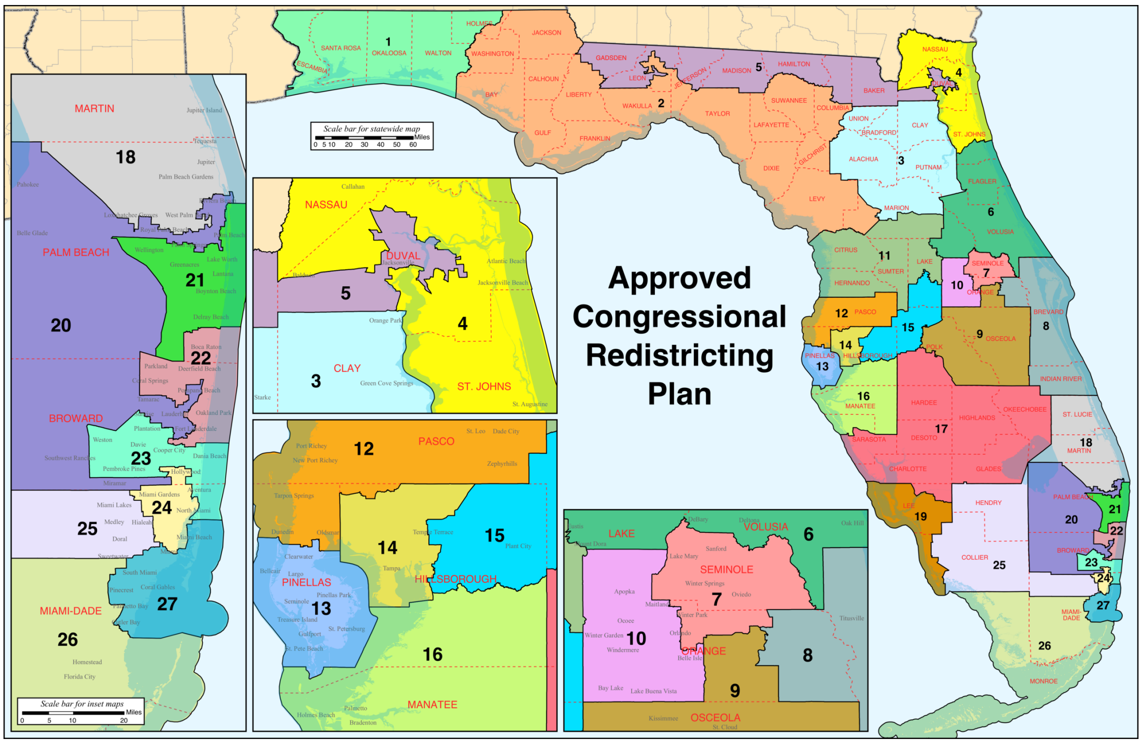


Figure 1: Florida's Congressional districts. Notice the contorted boundaries, particularly on districts 5 and 20 [11].

The reader may be familiar with Voronoi diagrams, or at least with the beautiful cells they produce. Power diagrams are basically an extension of that idea.

The objective. If we are given a set P of m points in a Euclidean space E^d and another set C of k points in the same space, we want to find a function f that maps each point in P to a point in C. This is an assignment from voters to centers, with each center defining a cell into which the some people will be partitioned [8].

Definition 2.1. Given a set of k centers C and associated weights $w_x \in \mathbb{R}\ \forall x \in C$, the **power diagram** $\mathcal{P}(C, w)$ satisfies the following condition. The weighted squared distance from a point $y \in E^d$ to a center $x \in C$ is defined as $d^2(y, x) - w_x$. The power region associated with center x is defined as follows: $C_x := \{y \in E^d \mid d^2(y,x) - w_x \leq d^2(y,z) - w_z \quad \forall z \in C \backslash \{x\}\}$. That is, no other center has a lower weighted squared distance to y than x. We define an assignment f as consistent with $\mathcal{P}(C, w)$ if it meets this condition [8]. In fact, we may view each point $x \in C$ as a sphere $s = \{y \in E^d \mid d(y,x) = \sqrt{w_x}\}$. As such, our weighted distance function is now defined as the power of a point $y \in E^d$ with respect to s, denoted $pow(y, s)$ [1]. See Figure 2.

Definition 2.2. If all the weights are zero, then we have a **Voronoi diagram**, $\mathcal{V}(C)$ where all points $y \in E^d$ are assigned to the center x with the smallest squared distance [8].

Definition 2.3. A power diagram $\mathcal{P}(C, w)$ is **balanced** if it satisfies the following property: the number of points assigned to each center differs by no more than 1. That is, the first i centers (where $i \in [0, k]$) are assigned $\lceil m/k \rceil$ points, and the remaining $(m - i)$ centers are assigned $\lfloor m/k \rfloor$ points such that $i\lceil m/k \rceil + (m - i)\lfloor m/k \rfloor = m$ [8].

Definition 2.4. A power diagram $\mathcal{P}(C, w)$is **centroidal** if for each power region C_x corresponding to center x, x is the centroid of $P \cap C_x$ [8].

Now we are in a position to formulate our full objective. We seek a balanced centroidal power diagram $\mathcal{P}(C, f, w)$ given the locations of the people P. In practice, we don't actually know the locations at that level of detail. Instead, we use census blocks, which are assigned to the centroid of their geographic area and contain a known number of individuals [8]. Furthermore, we want our assignment $f : P \to C$ to minimize the total squared distance between all residents and their assigned center.

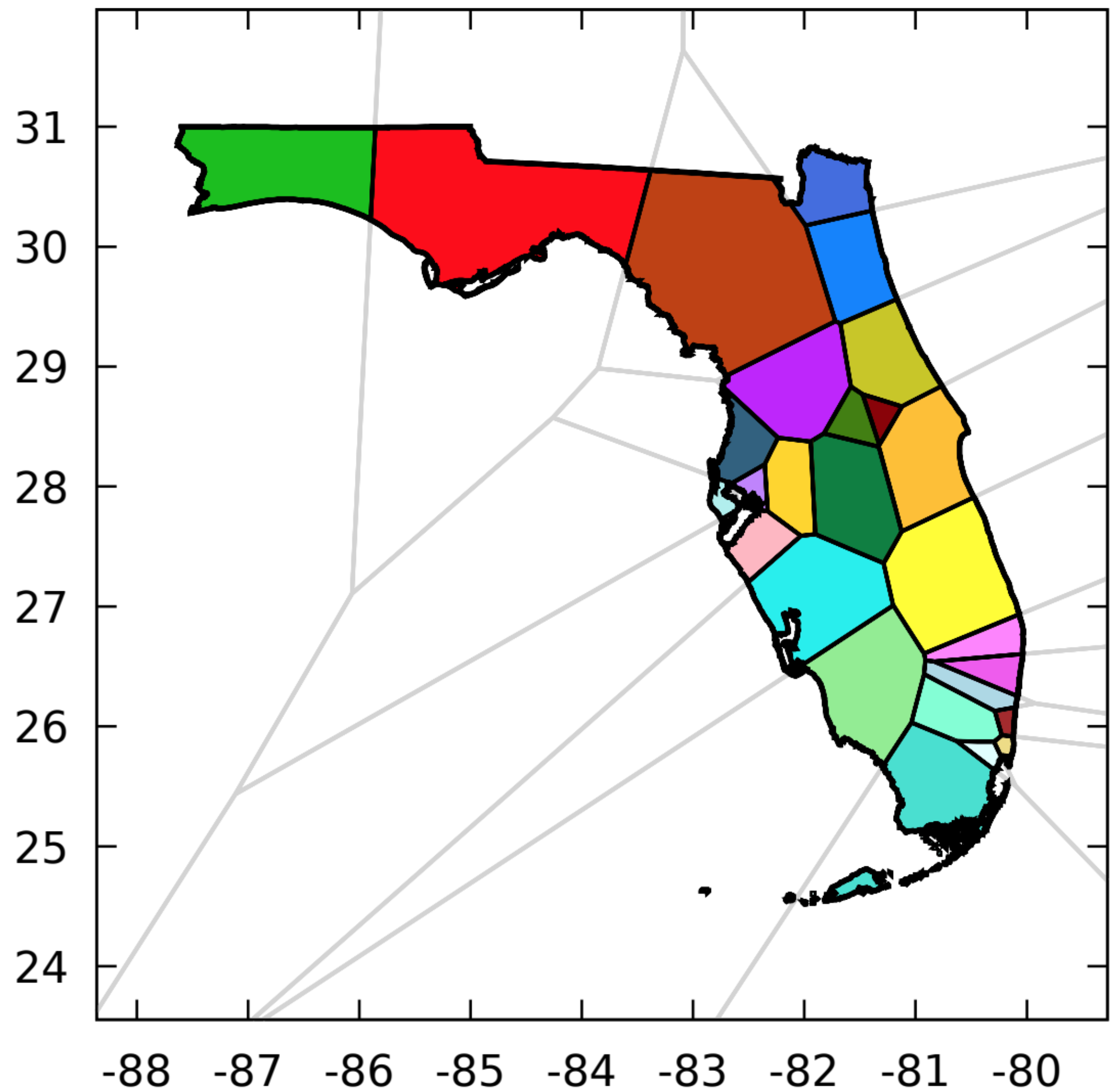


Figure 2: Klein et al.'s algorithm applied to the state of Florida (27 districts) [8].

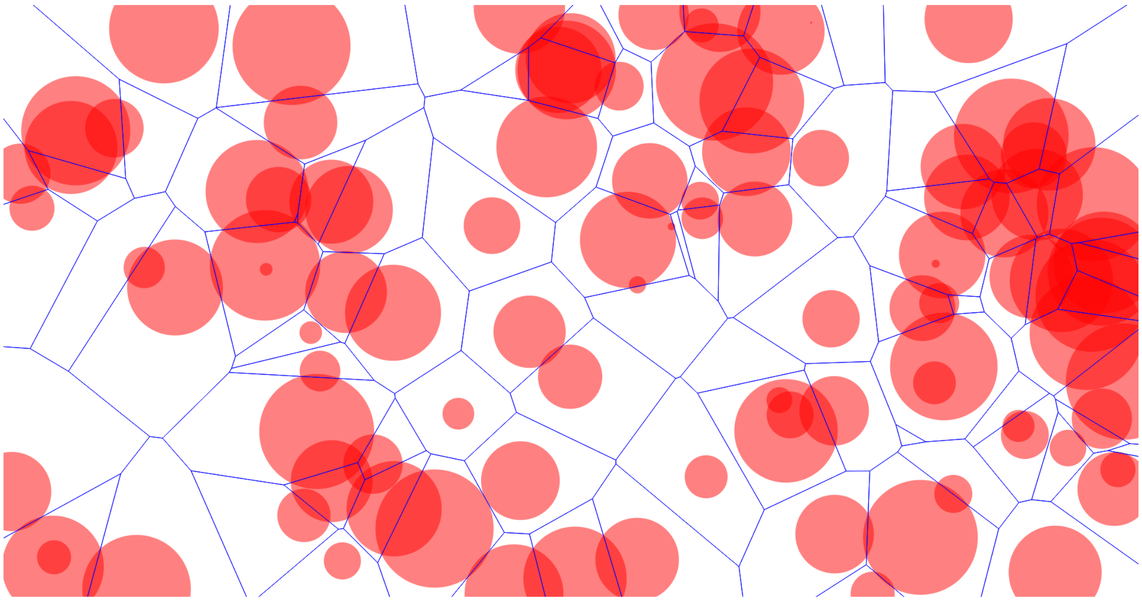


Figure 3: A weighted power diagram. This visualization shows how the weights can be represented as radii of the centers [3].

3 Lloyd's Algorithm

Lloyd's algorithm, or the k-means clustering algorithm, is a simple 2-step process to find a centroidal Voronoi diagram $\mathcal{V}(C, f)$ given the residents P and the desired number of power regions k. It operates as follows, where the cost is defined as $\sum_{y \in V} d^2(y, f(y))$:

Algorithm 1 Lloyd's algorithm [8]

```
1: procedure LLOYD($P, k$)
2:     $C \leftarrow$ set of $k$ centers randomly chosen from $P$
3:     repeat
4:         $f \leftarrow$ minimum-cost assignment from $P$ to $C$
5:         $C \leftarrow \{x \mid x =$ centroid of all points in $P$ assigned to $x\}$
6:     until $C$ and $f$ are unchanged
7:     return $C, f$
```

Basically, you alternate between fixing C and assigning all points in P to the closest center in C, and fixing f and assigning each point in C to the centroid of the points in P assigned to it. After some number of iterations, this will converge, returning a centroidal Voronoi diagram (remember, this is a power diagram with all weights equal to zero). We can also observe that the algorithm will iteratively reduce the cost, until the last iteration. The final returned (C, f) is a local minimum [8]. For an interactive visualization of k-means (coded by the author in JavaScript D3), visit this link.

We notice that Lloyd's algorithm *almost* accomplishes our objective, but for one critical flaw; the cells are not necessarily balanced. A few runs of the aforementioned visualization will make clear as much. To fix this dilemma such that Lloyd's algorithm can be applied to our goal, stipulated at the end of the previous section, we modify the algorithm as follows. Klein et al. refer to this as the *capacitated* variant of Lloyd's algorithm.

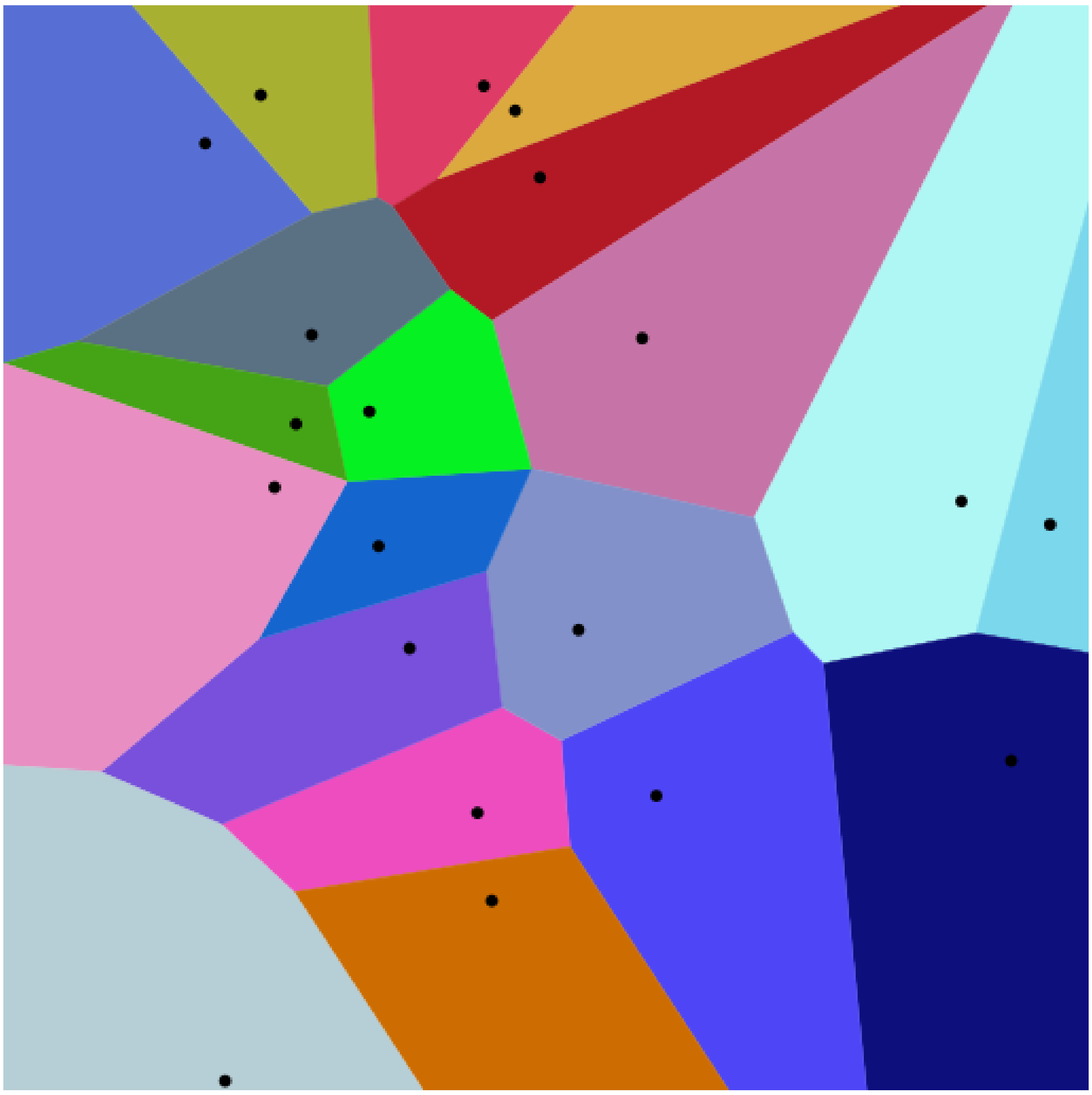


Figure 4: A Voronoi diagram. Notice how all boundary edges E between two centers c_1 and c_2 are defined by $d^2(x, c_1) = d^2(x, c_2)\ \forall x \in E$ [18].

Algorithm 2 Lloyd's algorithm, capacitated variant [8]

```
1: procedure LLOYDCAPACITATED($P, k$)
2:     $C \leftarrow$ set of $k$ centers randomly chosen from $P$
3:     repeat
4:         $f \leftarrow$ minimum-cost *balanced* assignment from $P$ to $C$
5:         $C \leftarrow \{x \mid x =$ centroid of all points in $P$ assigned to $x\}$
6:     until $C$ and $f$ are unchanged
7:     return $C, f$
```

We've only made a tiny tweak to the algorithm by requiring each definition of f to be balanced. However, this makes the problem considerably more complicated. Before, it was easy to minimize the cost $\sum_{y \in V} d^2(y, f(y))$ by just assigning every data point to the center closest to it. The assignment of each element was independent of that of other elements. That is no longer the case, as we are constrained by the requirement of balance (defined mathematically in Definition 2.3). We are now presented with a constrained optimization problem whose solution is not obvious and must be supplemented with some necessary background.

4 Constrained Optimization

Suppose we need to optimize some function $g(\mathbf{X}) = g(X_1, X_2, \ldots, X_n)$ which has first and second derivatives. If we have no constraints, then the function's extrema will be located at stationary points where

$$\frac{\partial g}{\partial X_i} = 0 \;\forall\; i \in \{1, 2, \ldots, n\}$$

If the function is constrained, however, this may not be the case, as the optima could occur at intersections with the constraint where $\nabla g \neq \vec{0}$ [4].

In general, a constrained optimization problem can be formulated as follows. We aim to optimize an **objective function** $g(\mathbf{X})$ subject to a set of m equality constraints $h_j(\mathbf{X}) = b_j$ $\forall j \in 1, 2, \ldots, m$. Furthermore, we impose the constraint that $\mathbf{X} \geq \vec{0}$ [4].

This nonnegativity condition makes practical sense and doesn't actually limit the problem. Often, the X_i we're trying to optimize represent quantities of real-world commodities, which are inherently nonnegative. If one of these variables is negative, however, we can easily redefine it by flipping the sign or adding a large amount to make it positive [4].

Taken together, we write our problem as:

$$\begin{array}{lll} \text{Optimize:} & g(\mathbf{X}) & \\ \text{Subject to} & h_j(\mathbf{X}) = b_j \quad \text{and} & \mathbf{X} \geq \vec{0} \end{array}$$

So how do we handle all of these constraint equations? Do we have to find all of the intersections and then painstakingly compare all different possible extrema? No, we don't; with a clever trick we can turn this problem into an unconstrained optimization problem. That trick is called the Lagrangian.

Definition 4.1. The **Lagrangian** is a new function to be optimized, which combines the objective function and the constraints. For objective function $g(\mathbf{X})$ and constraints $h_j(\mathbf{X}) = b_j, \mathbf{X} \geq \vec{0}$, we define the Lagrangian as follows [4]:

$$L = g(\mathbf{X}) - \sum_{j=1}^{m} \lambda_j [h_j(\mathbf{X}) - b_j]$$

Definition 4.2. The parameters λ_j associated with each constraint equation in Definition 4.1 are called the **Lagrange Multipliers**.

We can treat optimization of the Lagrangian as an unconstrained problem; we've just introduced some new variables. Whereas before we had n **decision variables** (the X_i), we have m additional variables (the Lagrange multipliers), so by setting $\nabla L = \vec{0}$ we now have $n + m$ equations [4]:

$$\begin{aligned}
\frac{\partial L}{\partial X_i} &= \frac{\partial g}{\partial X_i} - \frac{\sum_{j=1}^{m} \lambda_j \partial h}{\partial X_i} &= 0 \\
\frac{\partial L}{\partial \lambda_j} &= h_j(\mathbf{X}) - b_j &= 0
\end{aligned}$$

Note that the first constraint listed above illustrates that the gradient of g is a linear combination of the gradients of the constraint functions h_j at the optimal solution. We use the superscript $*$ to denote optimality. Our solution will be of the form (X^*, λ^*).

We can make a couple more useful conclusions. First, note that $\frac{\partial L}{\partial b_j} = \lambda_j$. That is, the value of the Lagrange multiplier tells you about how sensitive the problem is to changes in your constraint. Furthermore, at the optimal solution, $g^*(\mathbf{X}) = L$ because the constraints are all met [4].

Now let's see an example of this.

Example 4.1. Suppose we want to make an open-topped box with a square base and some height. We have a fixed amount of cardboard and we want to maximize the volume the box can hold. Our problem is thus formulated as:

$$\begin{aligned}
&\text{Optimize:} && g(\mathbf{X}) = X_1^2 X_2 \\
&\text{Subject to} && h_1(\mathbf{X}) = X_1^2 + 4X_1X_2 = S \qquad \text{and} \qquad \mathbf{X} \geq \vec{0}
\end{aligned}$$

The Lagrangian is

$$L = X_1^2 X_2 - \lambda_1(X_1^2 + 4X_1X_2 - S)$$

Thus from $\nabla L = \vec{0}$ we get:

$$\frac{\partial L}{\partial X_1} = 2X_1X_2 - 2\lambda_1 X_1 - 4\lambda_1 X_2 = 0 \tag{1}$$

$$\frac{\partial L}{\partial X_2} = X_1^2 - 4\lambda_1 X_1 = 0 \tag{2}$$

$$\frac{\partial L}{\partial \lambda_1} = X_1^2 + 4X_1X_2 - S = 0 \tag{3}$$

From (2) we have

$$X_1(X_1 - 4\lambda_1) = 0$$

Since X_1 can't be zero, we know

$$X_1 - 4\lambda_1 = 0 \implies \lambda_1 = \frac{X_1}{4}$$

We can now plug this into (1):

$$\begin{aligned}
2X_1X_2 - \frac{X_1^2}{2} - X_1X_2 &= 0 \\
-X_1^2 + 2X_1X_2 &= 0
\end{aligned}$$

Adding this result to (3) reveals

$$6X_1X_2 - S = 0$$

Now we easily express X_2 in terms of X_1:

$$X_2 = \frac{S}{6X_1}$$

We can now substitute this back int (3) to give an equation wholly in X_1:

$$\begin{aligned}
X_1^2 + 4X_1\frac{S}{6X_1} - S &= 0 \\
X_1^2 - \frac{S}{3} &= 0
\end{aligned}$$

We now have a solution for X_1, and it's easy to see how we then get X_2. The solution is thus

$$\begin{cases} X_1 = \sqrt{\frac{S}{3}} \\ X_2 = \frac{1}{2}\sqrt{\frac{S}{3}} \\ \lambda_1 = \frac{1}{4}\sqrt{\frac{S}{3}} \end{cases}$$

We can verify that this solution meets the necessary conditions. It leads to the optimal solution $g^*(\mathbf{X}) = \frac{1}{2}\left(\frac{S}{3}\right)^{2/3}$.

We can extend this concept even further to problems with inequality constraints. Since Klein et al.'s paper mentions slack variables, it is useful to mention them in this paper.

Suppose you are given the following constrained optimization problem with inequality constraints:

$$\begin{array}{lll} \text{Optimize:} & g(\mathbf{X}) & \\ \text{Subject to} & h_j(\mathbf{X}) \leq b_j \qquad \text{and} & \qquad \mathbf{X} \geq \vec{0} \end{array}$$

We can, with a simple trick, turn this into a problem with equality constraints, which we know how to handle. To do so, we introduce m new variables (recall we have m constraints) called *slack variables*. These nonnegative values fill in the gap between $h_j(\mathbf{X})$ and b_j so as to achieve equality [4]. That is:

$$h_j(\mathbf{X}) + S_j^2 = b_j$$

Note that the slack variables are squared so as to achieve nonnegativity.

We now define the *generalized Lagrangian* as

$$L = g(\mathbf{X}) - \sum_{j=1}^{m} \lambda_j (h_j(\mathbf{X}) + S_j^2 - b_j)$$

Just as before, we write out all the equations given by $\nabla L = \vec{0}$. However, this time, we have m extra equations, called the *complementary slackness* conditions [4]:

$$\frac{\partial L}{\partial S_j} = 2\lambda_j S_j = 0 \qquad \forall\, j \in \{1, 2, \ldots, m\}$$

Together with the equations $\frac{\partial L}{\partial X_i} = 0$ and $\frac{\partial L}{\partial \lambda_j} = 0$ we have $n + 2m$ equations, called the *Kuhn-Tucker conditions*. Once these conditions are established, the procedure is basically the same as with equality constraints. Refer to the previous example.

5 Linear Programming

We saw in the constrained optimization example with the cardboard box that the solution involved solving a system of equations — a tedious process even with just three variables. In practice, problems generally have many more decision variables and constraints. For instance, the redistricting problem aims to assign m residents to a center, subject to the constraint that all the centers are assigned an equal number of people and that each person is assigned to one center. This involves millions of variables. As such, it is easy to see that having some assumptions on which we can rely to simplify the problem is absolutely crucial when dealing with such problems. That is what **linear programming** (LP) does, and what makes it so powerful.

A linear program can be formulated as follows [4]:

$$\begin{array}{lll} \text{Optimize:} & Y = \mathbf{C}^T\mathbf{X} & \\ \text{Subject to} & \mathbf{AX} \leq \mathbf{B} \qquad \text{and} & \qquad \mathbf{X} \geq \vec{0} \end{array}$$

Here, we use matrix notation consistent with general practice; $\mathbf{C}$ is a vector of coefficients on the decision variables, $\mathbf{A}$ is the matrix of constraints, and $\mathbf{B}$ is the column vector of constraints.

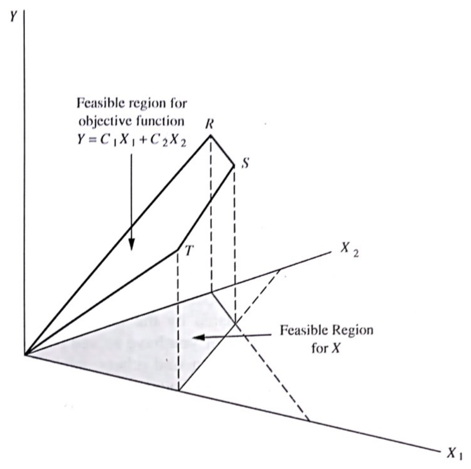
This is the familiar structure of an optimization problem with inequality constraints presented earlier. However, we now establish that both the objective function Y and the set of constraints are linear. Linearity implies the following:

Definition 5.1. A function $f(\mathbf{X}) : \mathbb{R}^n \to \mathbb{R}^m$ is **linear** if it satisfies the following two properties for all $\mathbf{U}, \mathbf{V} \in \mathbb{R}^n$ and $a \in \mathbb{R}$:

1. Additivity: $f(\mathbf{U} + \mathbf{V}) = f(\mathbf{U}) + f(\mathbf{V})$
2. Constant Returns: $f(a\mathbf{U}) = af(\mathbf{U})$

So what do these assumptions mean for where the optimum could be located?

Definition 5.2. For a linear program with objective function $Y = \mathbf{C}^T\mathbf{X}$ and constraints $\mathbf{AX} \leq \mathbf{B}$, the **feasible region** for $\mathbf{X} \in \mathbb{R}^n$ is the subset of $\mathbb{R}^n$ in the union of all the constraints [4]. As such, we also have a feasible region for the objective function Y. The following diagram illustrates the concept.



Y
Feasible region for objective function $Y = C_1X_1 + C_2X_2$
R
S
T
X_2
Feasible Region for X
X_1

Figure 5: The feasible region defined by objective function Y and two constraints on X_1, X_2 [4].

Theorem 1. *The feasible region is convex, if it exists at all.*

Proof. For a linear constraint, any two points that satisfy the constraint are on the same side of the associated hyperplane. As such, the line between those two points also entirely lies on one side of the hyperplane. This is precisely the definition of convexity: for any two points belonging to a set S, the line between them completely belongs to S. Therefore, the feasible region is convex. However, there could possibly be no feasible region, if the feasible regions defined by the constraints don't intersect [4]. $\square$

Theorem 2. *The optimum of Y must be located at a corner of the feasible region.*

Proof. The objective function Y is linear, so the directional derivative $\nabla_{\vec{v}}Y$ in any direction $\vec{v}$ is constant along $\vec{v}$. Thus, there is an edge with a greater value of the objective than anywhere in the interior. An analogous argument is used to show that the extremum of the objective along that edge must be at a corner. Furthermore, the optimum is a global optimum [4]. $\square$

5.1 Augmented Form

We can extend the concept of slack variables mentioned in the section on Lagrangians to linear programs. We can write this in matrix form [10]:

$$\begin{bmatrix} 1 & -\mathbf{C}^T & 0 \\ 0 & \mathbf{A} & \mathbf{I} \end{bmatrix} \begin{bmatrix} Y \\ \mathbf{X} \\ \mathbf{S} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{B} \end{bmatrix}$$

5.2 Duality

Duality is a crucial concept in linear programming, and features prominently in Klein et al.'s approach. It refers to the reframing of a problem into a different representation [4]. We call the original problem the **primal** and the reformulated version the **dual**. These two problems are duals of each other. For instance, the boundary of a space is dual to the space itself [4]. A more complex example is that the Voronoi boundaries are dual to the Delaunay triangulation of a set of points [10].

To find the dual, we must transpose our problem to a different form, and from there define a new objective function to be optimized. The procedure is as follows [4]:

If the primal is

$$\begin{array}{llll} \text{Maximize:} & Y = \mathbf{C}^T\mathbf{X} & & \\ \text{Subject to} & \mathbf{A}\mathbf{X} \leq \mathbf{B} & \text{and} & \mathbf{X} \geq \vec{0} \end{array}$$

then its dual is

$$\begin{array}{llll} \text{Minimize:} & Z = \mathbf{B}^T\mathbf{W} & & \\ \text{Subject to} & \mathbf{A}^T\mathbf{W} \geq \mathbf{C} & \text{and} & \mathbf{W} \geq \vec{0} \end{array}$$

Dual problems have some important properties, including the following.

- For a given primal problem p and its dual d, p is the dual of d. That is, the dual of the dual is the primal [10].
- The number of constraints becomes the number of decision variables and vice versa [4].
- Weak duality states that the optimum of the dual problem provides an bound on the optimum of the primal [10].
- Strong duality states that the solutions are equal, i.e. $Y^* = Z^*$ [10].

An example is now very much in order. The following example is from Wikipedia [10].

Example 5.1. Suppose we want to maximize the revenue for a farm with a fixed amount of land area L, fertilizer F, and pesticide P. It plants two crops (wheat and barley, say). Let X_1 denote the area of wheat to be planted, x_2 the area of barley planted, F_1, F_2 the respective amounts of fertilizer per unit area, P_1, P_2 the respective amounts of pesticide per unit area, and C_1, C_2 the respective selling prices of the goods per unit area.

Thus we have the following optimization problem:

$$\begin{array}{llll} \text{Maximize:} & Y = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} & & \\ \text{Subject to} & \begin{bmatrix} 1 & 1 \\ F_1 & F_2 \\ P_1 & P_2 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \leq \begin{bmatrix} L \\ F \\ P \end{bmatrix} & \text{and} & \mathbf{X} \geq \vec{0} \end{array}$$

The constraints at play here are restriction on land area, fertilizer used, and pesticide used. Note that we can introduce three new slack variables $\{S_1, S_2, S_3\}$ so as to achieve equality. The resulting problem is now:

Maximize Y:

$$\begin{bmatrix} 1 & -C_1 & -C_2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & F_1 & F_2 & 0 & 1 & 0 \\ 0 & P_1 & P_2 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Y \\ X_1 \\ X_2 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 0 \\ L \\ F \\ P \end{bmatrix}, \begin{bmatrix} X_1 \\ X_2 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \geq 0$$

We can also find the dual problem. According to our definition, we must define 3 new decision variables (because there were 3 constraints in the primal). We shall call these $\{W_1, W_2, W_3\}$. Now our problem can be formulated as:

$$\text{Minimize:} \quad Z = \begin{bmatrix} L & F & P \end{bmatrix} \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix}$$

$$\text{Subject to} \quad \begin{bmatrix} 1 & F_1 & P_1 \\ 1 & F_2 & F_2 \end{bmatrix} \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix} \geq \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} \qquad \text{and} \qquad \mathbf{W} \geq \vec{0}$$

Now we should ask ourselves, what does this reframing of the problem mean? Well, a bit of intuition can help here. We can see that this has gone from a problem dealing with physical quantities to a problem about economic values. In other words, the primal is a partitioning problem to maximize revenue, while the dual is a pricing problem to minimize cost.

5.3 Balanced Power Diagrams as a Linear Program

As already established, finding the balanced power diagram is a constrained optimization problem. We will now express it mathematically, from which point it will be clear we are dealing with a linear program.

Our objective is to minimize the sum of squared distances between all points in P an their assigned center in C. This is subject to the constraints that:

1. No two centers differ in population by more than one (the definition of balanced we formulated earlier).
2. Each person is only assigned to one center (districts don't overlap).

We can then write this as the following optimization problem [8]:

$$\text{Minimize:} \quad \textstyle\sum_{x\in C}\sum_{y\in P} d^2(x,y)a_{xy}$$

$$\text{Subject to:} \quad \textstyle\sum_{y\in V} a_{xy} = \mu_x \quad \forall\ x \in C$$

$$\textstyle\sum_{x\in C} a_{xy} = 1 \quad \forall\ y \in P$$

$$a_{xy} \geq 0 \quad \forall\ x \in C, y \in P$$

Klein et al.'s paper provides us with the dual problem [8]. It is:

$$\text{Maximize:} \quad \textstyle\sum_{x\in C} \mu_x w_x + \sum_{y\in P} z_y$$

$$\text{Subject to:} \quad z_y \leq d^2(x,y) - w_x \quad \forall\ x \in C, y \in P$$

This is where Klein's paper is highly deficient in background and intuition. While this paper won't write out the matrix associated with the linear program nor rigorously prove why this is the dual, it will provide some powerful intuition as to why it makes sense. Now that we have some understanding of LP and duality, we can explain what is going on here.

First, we shall list all the terms and what they represent:

Primal	
Symbol	Meaning
a_{xy}	These are the decision variables. Each a_{xy} is an entry in the matrix of assignments between people $\in P$ and centers $\in C$. That is, a_{xy} refers to the assignment of person $y \in P$ to center $x \in C$. Feasible solutions have $a_{xy} \in \{0,1\}$ $\forall\ x \in C, y \in P$.
μ_x	The number of people $\in P$ assigned to center $x \in C$. Since we have $\lvert C\rvert = k$ districts and $\lvert P\rvert = m$, we will have $\mu_x \in \{\lfloor m/k \rfloor, \lceil m/k \rceil\} \quad \forall x \in C$.

Dual	
Symbol	Meaning
w_x	These are new decision variables. Each w_x is a weight assigned to a center $x \in C$.
z_y	These are new decision variables. Each z_y is a scalar assigned to a person $y \in P$.
μ_x	Same as in the primal.

We can make a few conclusions and observations from this reformulation of the primal. They are:

- Consistent with our definition of the dual, it has flipped from a minimization problem to a maximization problem.
- All of the variables that show up as constraints in the primal are now coefficients on the decision variables in the dual (like μ_x).
- In the primal, we have $|C| + |P|$ constraints. In the dual, we now have $|C| + |P|$ decision variables: the $|C|$ w_x's and the $|P|$ z_y's.
- Conversely, we previously had $|C||P|$ decision variables (each a_{xy}) and we now have that many constraints: one for each pair of (z_y, w_x).
- The dual basically says, given that each center is assigned equal numbers of people and that each person is assigned to one center, find the weighting scheme that maximizes the power (mentioned earlier) of each point with respect to the centers.
- In the primal we were determining how much of a person (feasibly all or none) to assign to each center (similar to the partitioning problem about the farm used earlier). In the dual we are determining global quantities that determine how to make those assignments.

5.4 Solving Linear Programs

There are several ways to solve linear programs. This paper will only discuss two of them. The first is the **simplex method**. This paper will only discuss it in concept and will not rigorously delve into the technicalities of its implementation.

Basically, the simplex method relies on the fact that the feasible region for our decision variables is a convex polytope [4]. We know from Theorem 2 that the optimum of our objective function will be at a corner of this convex polytope. Naturally, a method to solve a linear program should be an organized search through the set of corners. So, we'll start at one corner and then move to adjacent ones until the optimum is found. Each iteration of the algorithm moves to the corner that, in moving towards, increases the objective the most (for minimization problems; it's vice-versa for maximization problems). Once we reach a corner for which moving to any adjacent corner worsens the objective, we know we've found the optimum [4].

Another method is **minimum-cost flow**, if it is appropriate to convert the linear program to such a problem. This approach represents a shift in how the problem is viewed; the decision variables become flows in a network, which is a graph with certain properties attached to it. This approach is the topic of the next section.

6 Minimum-cost Flow and the Goldberg-Tarjan Algorithm

Klein et al. use a minimum-cost flow approach to solving the linear program and dual, using the Goldberg-Tarjan algorithm. This section will give some high-level background on what minimum-cost flow problems are, how flow networks work, and how the Goldberg-Tarjan algorithm can be applied to such problems.

6.1 Flow Networks

When we deal with network optimization algorithms, we are dealing with flow networks. These are basically graphs with added properties and rules that govern their use.

A **flow network** is a directed graph $G(V, E)$. Each arc $a \in E$ has a real-valued capacity $u(a)$, flow $f(a)$, and cost $c(a)$. Each vertex $v \in V$ has a real-valued demand $d(v)$. The network necessarily has a few properties [7]:

- It is *symmetric*. This means that $a^R \in E \ \forall a \in E$ where a^R is the reverse arc of a.
- The cost function satisfies $c(a) = -c(a^R) \ \forall a \in E$.
- The total demand is zero. That is, $\sum_{v \in V} d(v) = 0$.

Henceforth, we will denote $n = |V|$, $m = |E|$, and C as the biggest input cost.
We can now make some definitions about flow networks.

Definition 6.1. A **pseudoflow** is a function $f : E \to \mathbb{R}$ which satisfies the following two constraints [7]:

1. Antisymmetry. That is, $\forall a \in E$, $f(a) = -f(a^R)$.
2. The flow cannot exceed the capacity in any arc. That is, $\forall a \in E$, $f(a) \leq c(a)$.

Definition 6.2. For a pseudoflow f and a vertex v, the **excess flow** into v is defined as the difference between all the flow into v and the demand of v [7]:

$$e_f(u, v) = \sum_{(u,v)\in E} f(u, v) - d(v) = 0$$

Furthermore, we say than an arc $a \in E$ is *conserving* if $e_f(a) = 0$, *deficient* if $e_f(a) < 0$, and *active* if $e_f(a) > 0$ [6].

Definition 6.3. A **pre-flow** is a pseudoflow that satisfies $e_f(a) \geq 0 \; \forall a \in E$ [6].

Definition 6.4. A **(feasible) flow** is a pseudoflow that satisfies $e_f(a) = 0 \; \forall a \in E$. This means that all nodes are conserving, and there are no active nodes [7].

Definition 6.5. For an arc $a \in E$, the **residual capacity** is defined as the unused capacity, i.e. the difference between capacity and flow [7]:

$$u_f(a) = u(a) - f(a)$$

We then define an arc $a \in E$ as *saturated* if $u_f(a) = 0$, and *residual* if $u_f(a) > 0$.

Definition 6.6. The **residual graph** is the graph of the residual arcs, $G_f(V, E_f)$ [7].

Definition 6.7. We define a **price function** as a function $p : V \to \mathbb{R}$. Then, we say that the **reduced cost** $c_p(u, v)$ of an arc is the capacity of the arc plus the change in price across it [7]:

$$c_p(v, w) = u(v, w) + p(v) - p(w)$$

Definition 6.8. An arc $a \in E$ is said to be **admissible** if it has negative reduced cost and is residual. This means that there's room for more flow [7].

Definition 6.9. For a constant $\epsilon \geq 0$ pseudoflow is said to be ϵ-optimal if $\forall a \in E$, $c_p(a) \geq -\epsilon$ [7].

Now that we have defined the general terminology of flow networks, we can explain how our linear program gets turned into a network flow problem.
The **cost** of a pseudoflow f is given by

$$cost(f) = \frac{1}{2} \sum_{a\in E} c(a)f(a)$$

Note that the $\frac{1}{2}$ is introduced because the graph is symmetrical [7].
The aim of the minimum-cost flow problem is to minimize the cost. This sort of formulation looks familiar; recall the objective function in our districting assignment LP. The assignment of a person to a center is like a flow between nodes, and the distance between them is like a cost in a flow network.
With this in mind, we shall now proceed to outline an algorithm to find a minimum-cost flow, the Goldberg-Tarjan algorithm.

6.2 The Goldberg-Tarjan Algorithm

The Goldberg-Tarjan algorithm relies on the application of two operations, *push* and *relabel*. It operates on the residual graph, applying those operations in a certain way until they can no longer be applied.
First, I shall define these operations, where they are applicable, and give examples.

6.2.1 Push

The *push* algorithm pushes as much flow as possible from an active node along an admissible arc. The algorithm is as follows:

Algorithm 3 push [14]

```
1: procedure PUSH($v, w$)
2:     assert $e_f(v) > 0$ ($v$ is active), $u_f(v, w) > 0$, $c_p(v, w) < 0$
3:     $\Delta \leftarrow min(e_f(v), u_f(v, w))$
4:     $f(v, w) \leftarrow f(v, w) + \Delta$
5:     $f(w, v) \leftarrow f(w, v) - \Delta$
6:     $e_f(v) \leftarrow e_f(v) - \Delta$
7:     $e_f(w) \leftarrow e_f(w) + \Delta$
```

The algorithm first checks that v is an active node with (v, w) and admissible arc. Then, it determines the amount of flow to send from v to w. This flow is limited by both the amount of excess at the active node v and the amount of residual capacity of the edge (v, w), hence the minimum. Once the value of Δ is determined, the flow along (v, w) is incremented by Δ and, by symmetry, the reverse arc (w, v) is decremented accordingly. The excess flows at each node are updated similarly.

6.2.2 Relabel

The relabel algorithm is applied to active nodes with no admissible out-arcs. It decreases the price function at a node v by the minimum value so as to create an admissible out-arc.

Algorithm 4 relabel [7]

```
1: procedure RELABEL($v$)
2:     assert $e_f(v) > 0$ ($v$ is active), $c_p(v, w) \geq 0 \; \forall a \in E_f$ with $u_f(a) > 0$
3:     $p(v) \leftarrow max_{(v,w) \in E_f}\{p(w) - c(v, w) - \epsilon\}$
```

Notice that for a node v the higher its price function $p(v)$, the higher the reduced cost of its out-arcs. Since only out-arcs with negative reduced cost are admissible (and we cannot change the capacities of any edge (v, w)) we must decrease $p(v)$ to create admissible out-arcs. A little bit of careful analysis reveals that $max_{(v,w) \in E_f}\{p(w) - c(v, w) - \epsilon\}$ is the largest value to which to decrease $p(v)$ such that the arc is ϵ-optimal.

We can now introduce the two higher-level algorithms which call *push* and *relabel*.

6.2.3 Min-cost

This is the algorithm that finds the minimum-cost flow. It begins by initializing ϵ to C, setting the price p to 0 at all nodes, and establishing a flow. Then, it calls the *refine* algorithm until optimality is reached.

Algorithm 5 minCost [7]

```
1: procedure MINCOST($V, E, u, c$)
2:     $\epsilon \leftarrow C$
3:     for $v \in V$ do
4:         $p(v) \leftarrow 0$
5:     if $\exists$ a flow then
6:         $f \leftarrow$ a flow
7:     else
8:         return null
9:     while $\epsilon \geq 1/n$ do
10:        $(\epsilon, f, p) \leftarrow refine(\epsilon, f, p)$
11:    return $(f)$
```

6.2.4 Refine

The refine algorithm, as you can probably infer from the above min-cost algorithm, iteratively decreases ϵ and modifies the flow network to achieve ϵ-optimality for the new value of ϵ. Each call to the method decreases ϵ by a factor of α, a hyperparameter which is set by the implementor of the algorithm (it is usually at least 2).

Algorithm 6 refine [7]

```
1: procedure REFINE($\epsilon, f, p$)
2:     $\epsilon \leftarrow \epsilon/\alpha$
3:     for $(v, w) \in E$ do
4:         if $c_p(v, w) < 0$ then
5:             $f(v, w) \leftarrow u(v, w)$
6:     while $\exists$ an applicable push or relabel operation do
7:         do that operation
8:     return $(\epsilon, f, p)$
```

6.2.5 Recap

The Goldberg-Tarjan algorithm for minimum-cost flow operates on the residual graph $G_f(V, E_f)$. Our goal is to find a flow f that minimizes our cost, $cost(f) = \frac{1}{2}\sum_{(v,w)\in E} c(v, w)f(v, w)$. We introduce the price function p so that we can formulate a reduced cost on each edge. The reduced cost doesn't depend on the flow over an edge. When we apply the push algorithm, we apply it to residual arcs with a negative reduced cost. This means that the algorithm moves towards sending more flow along low-cost edges. We begin by creating a flow, probably with the Goldberg-Tarjan maximum-flow algorithm (like the min-cost algorithm, it relies on the push and relabel procedures). From there, we manipulate it until it is $1/n$-optimal, at which point we have an optimal solution. This minimum-cost flow algorithm would be difficult to give an example for, but one can come to a good conceptual understanding of how the push-relabel methods work and why we keep track of excess flow and price by seeing an example of maximum flow problem with Goldberg-Tarjan. See Wikipedia's article, "Push-relabel maximum flow algorithm" for a good example.

7 Areas for Future Research

I have attempted various paths of research through the duration of this project. In each of these attempts I faced either conceptual, technical, or temporal difficulties that made those directions not fruitful for my project scope, level of knowledge, and time frame. This section aims to describe those directions I attempted, why I think they're worthwhile, and my process while pursuing them. I hope that by giving an overview of my research process and difficulties I faced therein, I can provide a launching point for other inquisitive people to take up these questions, perhaps more informed and less likely to fall into the same holes I did.

Initially, my intention was to fully implement Klein et al.'s approach. I planned to get a working implementation in JavaScript, using D3 to visualize my results. I figured that this would be a worthwhile direction because Klein's code is public (which I could use for reference) and because implementing anything yourself is truly the best way to understand that thing. Furthermore, implementing it my own way would allow me to tailor it to my own purposes. Also, Klein's code is in C++, a language in which I'm not familiar, and I thought it would be a good opportunity to learn a bit of a new language so as to read and understand his code.

As far as that path goes, I never actually attempted to implement any of the difficult parts of Klein's approach. Back in the fall of 2017, during a machine learning independent project at PA, I coded up K-means clustering in D3. However, somehow much of my implementation got lost and I had to re-do it. It was quite doable, so I initially thought that the balanced version would be just some small tweaks and then it would work. Basically, once I had finished the first stage of my project (doing the reading, familiarizing myself with the concept and necessary background), I decided that I didn't have time to attempt such a conceptually difficult and demanding task. With little prior experience in graph theory or convex optimization, I felt a bit over my head as I read about Goldberg-Tarjan and LP duality. Even once I could understand parts of these ideas, although absent the high-level intuition as to why they work, I struggled to see exactly how they applied directly to the redistricting problem. Why is the dual problem posed by Klein correct? Are you solving the primal, the dual, or both? How exactly does the LP problem translate to a network flow problem and what does the network look like? These are questions I struggled with, that ultimately pushed me to turn away from the conceptual content of the project, much less a full implementation of my own.

At that point, I decided to take a more technical route. Rather than coming to the complete understanding of all of the concepts necessary to implement the algorithm, I would just have enough of an understanding to get the inputs and outputs of the program. I was thinking of applying Klein et al.'s code to real world data (easily accessible from the U.S. Census Bureau) and then assessing the program's output in terms of some compactness measures. To do this, I needed to first understand how to follow all their steps listed on their GitHub repo to run the code and get a result. Initially I thought that would be an easy task; I would just run a script or two in the command line and pass it the files with the Census Bureau data. It turned out, the procedure required to run Klein's code is rather convoluted and requires running a whole sequence of files. Furthermore, the input required to run these isn't clear. The explanation given on the README is very terse, often lacking in grammar or complete sentences, and clearly missing crucial information.

I ran into the biggest roadblock when trying to run the main C++ file, `do_redistrict.cpp`. I had no prior experience with C++, so I began with figuring out how to run the file. I learned about how C++ has different file types, with .h and .hpp being header files. To run a C++ program, you need to first compile the .cpp file and then run it, similar to Java. I learned that g++ is the compiler to use, which was installed on my Mac already. However, when I tried to compile their code with g++, it threw some errors. I traced these errors down to a cause, a line with the keyword `auto` in it. I discovered that this was the source of the errors and that it must have been a g++ version issue. The `auto` keyword is a C++17 feature, it turns out. Clearly, my computer wasn't equipped to compile C++17. I learned that you can use the `-std` flag in the command line to specify the version. In fact, when I specified C++11 as opposed to nothing, some errors went away. But C++17 was not a valid value for that. Thus, I realized that I needed to either install a compiler with C++17 capabilities or try to replace all the C++17 syntax with valid, equivalent earlier-version syntax. I did that with the `auto` keyword, but it only turned up a whole slew of new errors.

At this point I realized that attempting to hack my way through many thousands of lines of someone else's code in a language with which I had low proficiency was a poor idea. I needed to install a C++17 compiler. Searching through Stackoverflow question boards about this issue, I found few clear solutions. Apparently, C++17 is not fully operational or accessible yet. At least, it's not obvious how to get it. There was one thing for me to run in my command line which looked promising; I believe it was using homebrew to install something. I followed Stackoverflow's advice and tried it. After 3 hours of installing, `-std=c++17` still didn't work. Then, I began to look into the idea of using an online compiler. Wandbox seemed promising, but I realized I was going to have to upload all of the files one by one and I wasn't sure it would work with multiple files, or even support C++17. My last plan of action, as time ticked down, was to investigate the IDE known as XCode. My friend Nhat Pham suggested that this would be a good idea, and I was eager to try it. I had to manually recreate each file, but it was my last hope so I did it. And still, no success. In fact, XCode raised far *more* errors than did g++ in my command line. After playing around with it for a while, I realized it was hopeless. Later, I noticed that in the Klein paper it says they compiled it with g++-17. I hadn't realized this was a different thing, and didn't know if that would turn up anything successful. After briefly exploring that, I was too pressed for time to go any further into the time sink that was producing no results. It was time to change course.

And so, I returned to the conceptual side of the project. I decided that I would revisit the topics I had looked at earlier (and shied away from) and try to actually acquire a deeper understanding of them. Where earlier I had just given up and decided to move in the technical direction, I read on, trying to get a solid conceptual background on the topic. I had already read the Klein paper thoroughly, but I now turned to its references for more information. I read about the Goldberg-Tarjan paper, some of a paper on power diagrams, part of another paper by Goldberg, and more. I also used a lot of Wikipedia to get an overview (the math articles are quite excellent). Furthermore, my dad found a book from his phD studies that turned out to be immensely useful on the constrained optimization topic. From there, I was able to divide the project into three main concepts: power diagrams, linear programming, and network flow. I aimed to fill in the gaps of the

Klein paper, answering some questions I had when I read it. While this paper still doesn't explain exactly how all of the concepts are applied to the districting problem, it gives some intuition on them and allows the reader to make a connection between the topics and to districting.

I realize now how the re-implement Klein's code idea was way beyond the scope for this project. The technical path was also a risky move because if the code doesn't run, you don't have anything to show for it. Writing this paper was then definitely the right path for me. Ironically, had I more strongly dedicated myself to understanding the material in the early stages, I probably could have implemented some of the code. At any rate, the other project directions would still be quite fruitful and interesting if pursued with more time and resources. Hopefully, this paper can help provide useful background for that. At the very least, I know that for me, having this compilation of power diagrams, LP, and networks all in one place would have been helpful.

References

[1] Aurenhammer, Franz. "Power Diagrams: Properties, Algorithms, and Applications." *SIAM Journal of Computing* 16, no. 1 (February 1987). https://www.cs.jhu.edu/ misha/Spring16/Aurenhammer87.pdf.

[2] Berman, Ari. *Give Us the Ballot: The Modern Struggle for Voting Rights in America.* New York: Farrar, Straus and Giroux, 2015.

[3] Davies, Jason. "Power Diagram." https://www.jasondavies.com/power-diagram/.

[4] De Neufville, Richard. Applied Systems Analysis: Engineering Planning and Technology Management. New York: McGraw-Hill, 1990.

[5] Druke, Galen. "Why Can't We Just Burn Gerrymandering to the Ground?" *The Gerrymandering Project.* Podcast audio. November 30, 2017. https://fivethirtyeight.com/features/why-cant-we-just-burn-gerrymandering-to-the-ground/.

[6] "Flow Network." Wikipedia. Accessed May 31, 2018. https://en.wikipedia.org/wiki /Flow_network

[7] Goldberg, Andrew V. *An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm.* August 1992. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.258 &rep=rep1&type=pdf.

[8] Klein, Philip N., Vincent Cohen-Addad, and Neal E. Young. *Balanced Power Diagrams for Redistricting.* January 6, 2018. https://arxiv.org/pdf/1710.03358.pdf.

[9] Levitt, Justin. *A Citizen's Guide to Redistricting.* Edited by Brennan Center for Justice, NYU. http://www.brennancenter.org/sites/default/files/legacy/Democracy /2008redistrictingGuide.pdf.

[10] "Linear Programming." Wikipedia. https://en.wikipedia.org/wiki/Linear_programming.

[11] *Map of Florida's Congressional Districts.* Illustration. Wikipedia. https://en.wikipedia.org/wiki/Florida%27s_congressional_districts#/media /File:Florida_congressional_districts.png.

[12] McGhee, Eric. Memorandum to Supreme Court of the United States, memorandum, "Brief of Eric McGhee as Amicus Curiae in Support of Neither Party," August 10, 2017. http://www.scotusblog.com/wp-content/uploads/2017/08/16-1161-ac-eric-mcghee.pdf.

[13] Olson, Brian, and Warren D. Smith. "Theoretical Issues in Political Districting." Last modified August 2011. http://rangevoting.org/TheorDistrict.html.

[14] "Push-Relabel Maximum Flow Algorithm." Wikipedia. Accessed May 31, 2018. https://en.wikipedia.org/wiki/Push%E2%80%93relabel_maximum_flow_algorithm.

[15] "Shaw v. Reno." Oyez. Oyez. https://www.oyez.org/cases/1992/92-357.

[16] "Shelby County v. Holder." Oyez. https://www.oyez.org/cases/2012/12-96.

[17] "Vieth v. Jubelirer." Oyez. https://www.oyez.org/cases/2003/02-1580.

[18] "Voronoi Diagram." Wikipedia. https://en.wikipedia.org/wiki/Voronoi_diagram.

[19] "Wesberry v. Sanders." Oyez. https://www.oyez.org/cases/1963/22.