Linear Optimization of Gerrymandering

Yue Huang[1], Shuanger Huang[1], Brian Kang[1], and Stephen Hung[2]

[1]Department of Applied Mathematics, University of Washington

[2]Department of Mathematics, University of Washington

February 7 2019

1 Introduction

Our project's goal is to model the "redrawing" of district lines in a US State in order to maximize the winning chance of a one out of two parties. Since it would be trivial to change from one party to another, we chose the red party. In this project our model does not physically redraw the lines however, given the results from our model it is possible for someone to redraw district lines.

In order to model the data, we use a graph interpretation of Arizona State where the counties (or precincts as they will be referred to in this paper) became the vertices and the edges indicated whether a county/precinct were adjacent to each other. This can be seen in the graph in Appendix A.6

While we were unable to make these methods work for the final model, we believe that these methods are integral to the project and should be included. When dealing with the contiguity problem we attempted methods based on network flow optimization and planar graphs in order to force a contiguity constraint.

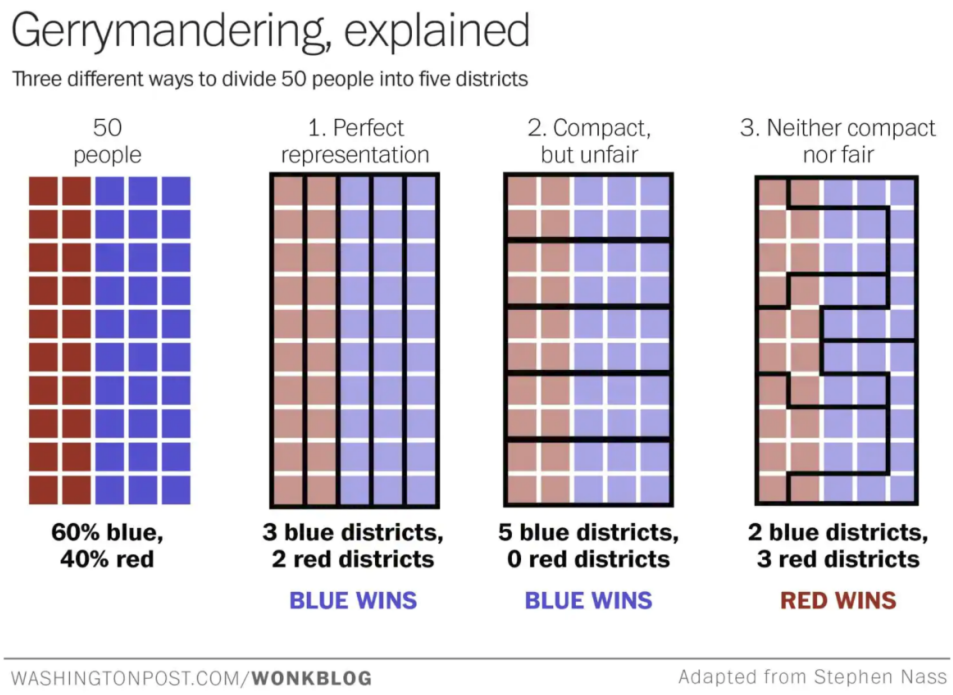

Figure 1: Illustration of Gerrymandering

2 Background

Gerrymandering is the manipulation of the boundaries of electoral districts in order to favor one party. When this happens, there are situations where the majority population vote for one party, but the majority district vote for the other. To draw the boundaries of political districts, there are several criteria that need to be satisfied:

1. Integrity: Each precinct must belong to exactly one district, that is, a precinct cannot be split into two or more parts.

2. Contiguity: All precincts in the same district must be connected somehow.

3. Absence of enclaves: No district can be fully surrounded by another district.

4. Compactness: A district must have a regular geometric shape and should not be unnecessarily spread out or odd shaped.

5. Population equality: District population must be as balanced as possible.

Before determining this topic, we researched problems that we could potentially optimize and created a list of feasible topics such as generating music sheets, optimizing package

delivery, and gerrymandering. Most of these topics were found by separate group members who thought it would be interesting to see optimized. We discussed what problems seemed the most interesting and viable before talking with Dr. Conroy where we finalized the topic of gerrymandering.

Similar research on applying Integer Programming to gerrymandering is abundant. The very first paper researching applications of Integer Programming to gerrymandering was in Hess' 1965 paper.[1] Hess' paper talks about minimizing variation between population count in districts within New Castle County. While they achieved 12% variation from the average population count, they had to take into account several constraints such as preserving Wilmington boundaries and using large population counts. Hess' 1965 paper provides a foundation for which many other researchers would continue to research applications of optimization and Integer Programming to gerrymandering. These papers are similar to our work in that they take into account contiguity to redraw voting districts. However, Hess, and the majority of political districting researchers, wants to ensure fairness by minimizing population variance between different districts in contrast to our objective to maximize unfairness by making a single party win.

Our project will adhere to the assumptions of political district boundaries that must be satisfied just like historically previous research already done. The project will also aim to apply some real world data to clearly illustrate how our research can influence the existing system, just like how many previous and current research show the stakes of their studies. Thus, we will carry on the implied standards of ongoing academic studies, but through a contradictory objective, our project will display a different perspective on how the current system can be easily gerrymandered while satisfying all the criteria required for political districting.

[1] S. W. Hess et al. "Nonpartisan Political Redistricting by Computer". In: *Operations Research* 13.6 (1965), pp. 998–1006.

3 Model

3.1 Model Implementation

For this project, we use linear programming to consider the problem of maximizing the sum of districts that vote for a party in a state so that the party wins in a two-party competition. We have districts encompassing precincts that either vote for Republican (red) or Democratic (blue). The districts are also either Republican (red) or Democratic (blue) depending on which party has the majority number of precincts in a district. In this project, we want to make red win. Although it is trivial to make blue win if desired.

We want to draw the district boundaries to satisfy integrity. Since we are interested in maximizing the unfairness in the election to make a single party win, contiguity, absence of enclaves, compactness, and population equality are ignored for now. In our model, we have n precincts to be assigned into k districts ($n \geqslant k$). Each district can have a different size, which is the number of precincts assigned in the district, but the difference cannot be more than two precincts in number. Before proposing our linear programming model, we introduce four variables: y_i denotes the party (red or blue) that the district i, as a whole, voted for; x_{ij} denotes the precinct j in district i; d_i is the number of precincts in each district i; b_i represents the number of precincts that voted for red in each district i, where $i, j \in \mathbb{N}$.

$$y_i = \begin{cases} 1 & \text{iff more than half of the precincts in district } i \text{ voted for red, } \forall i \in \{1, ..., k\} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if precinct j is assigned to district } i, \forall i \in \{1, ..., k\}, j \in \{1, ..., n\} \\ 0 & \text{otherwise} \end{cases}$$

$d_i =$ the number of precincts in each district $i, \forall i \in \{1, ..., k\}$

$b_i =$ the number of precincts that votes red in each district $i, \forall i \in \{1, ..., k\}$

To help formulate the LP model, we also need to introduce two more parameters: c_j

denotes the party (red or blue) that the precinct j votes for. It is a binary variable with values randomly assigned. M is an integer with some large value to attain the inequality constraints.

$$c_j = \begin{cases} 1 & \text{if precinct } j \text{ votes for red, } \forall j \in \{1, ..., n\} \\ 0 & \text{otherwise} \end{cases}$$

M: a large number, $M \in \mathbb{R}^+$

Then our linear program's objective function that satisfies integrity is:

$$\max \sum_{i=1}^{k} y_i$$

subject to,

$$\sum_{i=1}^{k} x_{ij} = 1, \qquad \forall j \in \{1, ..., n\} \tag{1}$$

$$\sum_{j=1}^{n} x_{ij} \geqslant 1, \qquad \forall i \in \{1, ..., k\} \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} \leqslant \frac{n}{k} + 1, \qquad \forall i \in \{1, ..., k\} \tag{3}$$

$$\sum_{j=1}^{n} x_{ij} \geqslant \frac{n}{k} - 1, \qquad \forall i \in \{1, ..., k\} \tag{4}$$

$$d_i = \sum_{j=1}^{n} x_{ij}, \qquad \forall i \in \{1, ..., k\} \tag{5}$$

$$b_i = \sum_{j=1}^{n} c_j x_{ij}, \qquad \forall i \in \{1, ..., k\} \tag{6}$$

$$My_i \geqslant (2b_i - d_i) - \frac{1}{2}, \qquad \forall i \in \{1, ..., k\} \tag{7}$$

$$M(y_i - 1) < (2b_i - d_i) - \frac{1}{2}, \qquad \forall i \in \{1, ..., k\} \tag{8}$$

$$y_i, x_{ij} \in \{0, 1\}, \qquad \forall i \in \{1, ..., k\}, j \in \{1, ..., n\} \tag{9}$$

Since we want to draw district boundaries to favor Republican (red), the objective func-

tion is to maximize the number of districts that vote for red.

To satisfy integrity, i.e., a precinct cannot be split into two or more parts and then assigned to different districts, we build constraint (1) to ensure that every precinct can only be assigned to exactly one district.

We predetermine a reasonable number of districts in our model. Constraint (2) ensures that all districts are assigned with at least one precinct.

Though our goal is to maximize the unfairness of gerrymandering, we try to make sense of the model by assigning each district a similar number of precincts. Constraint (3) gives the lower bound of the number of precincts assigned in each district and constraint (4) gives the upper bound of that. If size is the same for each district, it equals to total number of precincts divided by total number of districts, $\frac{n}{k}$, rounded to the nearest integer. In our model, district size can differ by most two precincts numerically, thus we define the range of the size from $(\frac{n}{k} - 1)$ to $(\frac{n}{k} + 1)$ inclusive.

After assigning precincts to each district, we store the number of precincts in each district because we need that to check if red wins that district. We define d_i as the number of precincts in each district, and constraint (5) helps record the value.

As we have a list of binary number of c_j, which indicates if the precinct votes for red, we want to know how many precincts vote for red in each district in order to check whether or not red wins the district. b_i is the number of precincts that vote for red in each district and it is equal to constraint (6), which is the sum of the products of precincts that vote red and color assigned to each one.

Since we want to maximize the the sum of districts who vote for red, y_i, in our model, we need constraints to force y_i to be either one or zero to indicate if more than half of the district vote for red or blue, respectively. We set $y_i = 1$ if and only if more than half of the district i vote for red. Otherwise, $y_i = 0$. Thus, if more than half of the district i vote for blue or there is a "tie" between precincts in a district, we assume red does not win the district.

Constraint (7) and (8) ensure y_i to be either 1 or 0 if and only if more than half of

a district is red or not, respectively speaking. We want to use the difference between the number of precincts vote for red in district i, b_i, and the half of the number of precincts in district i, $\frac{d_i}{2}$. Thus, what we want to check is whether or not $b_i - \frac{d_i}{2} > 0$. Notice that in reality, the difference between $b_i - \frac{b_i}{2}$ is always greater than or equal to $\frac{1}{2}$ since both b_i and d_i are integers because third and minority parties may participate in political elections. Because we want integers in our model, we modified the condition $y_i = 1$ be $2b_i - d_i > 1$. That is, $y_i = 1$ if and only if $2b_i - d_i > 1$. Otherwise, $y_i = 0$. In order to give lpsolve a more precise range for y_i, we do minus $\frac{1}{2}$ on the right-hand side in both constraints. It will not change the results since $(2b_i - d_i)$ is guaranteed to be an integer and y_i is binary.

If $y_i = 0$, we know $2b_i - d_i \leqslant \frac{1}{2}$ from constraint (7) and (8), it follows that the majority vote of the district is not red. If $y_i = 1$, then we get $2b_i - d_i > \frac{1}{2}$ from those two constraints. Since we know the difference between $2b_i - d_i$ is always an integer, the solution shows us $2b_i - d_i \geqslant 1$,which follows that the majority vote of the district is red.

If $(2b_i - d_i) - \frac{1}{2} < 0$, then we know y_i is greater then some negative numbers and $y_i - 1$ less than or equal to some negative number range from -1 to 0, exclusively, because M>> $(2b_i - d_i) - \frac{1}{2}$. Thus, we know $y_i = 0$. If $(2b_i - d_i) - \frac{1}{2} > 0$, we know $0 < y_i \leqslant 1$ from those constraints. Thus, $y_i = 1$. We do not care about the equal sign in those inequality equations (constraint (7) and (8)) because $2b_i - d_i$ is guaranteed to be an integer, so $(2b_i - d_i) - \frac{1}{2}$ can never be zero.

Therefore, $y_i = 1$ if and only if more than half of district i votes for red; otherwise $y_i = 0$.

Constraint (9) defines y_i and x_{ij} to be binary.

3.2 Real World Data

The data we are using for this Gerrymandering problem is Arizona's 2018 US Senate voting history. We pulled this data from Politico's 2018 US Senate coverage[2].

This data has 15 counties or "precincts" and 9 districts.

[2]Politico. *Arizona Election Results 2018: Live Midterm Map by County & Analysis.* URL: `www.politico.com/election-results/2018/arizona/.` (visited on 02/09/2019).

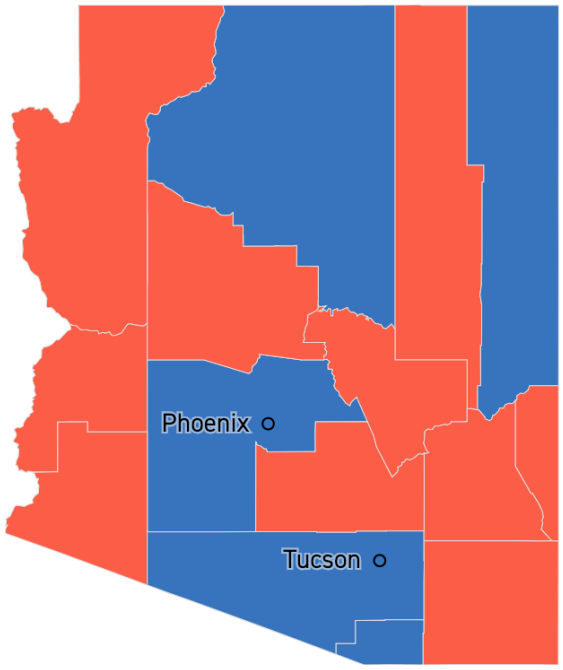

Figure 2: Visualization of Arizona 2018 Voting History

3.3 Coding Methods and Challenges

In order to generate the LP we used Python 3 as seen in Appendix A.1.

There were three main challenges that we faced when creating a model for Gerrymandering.

1. Creating an objective function to maximize the number of districts that vote red.
2. Generating the contiguity constraint to be more similar to the real world.
3. Dealing with a "tie" between precincts in a district.

3.3.1 Challenge 1: Creating an Objective Function

Due to the nature of Gerrymandering, much of the research about Gerrymandering is about maximizing fairness of the district lines instead of maximizing unfairness. Because of this, there were no research papers we could use as a foundation for our objective function.

We wanted our objective function to maximize unfairness by maximizing the number of districts that would vote red. We defined a district as voting red if and only if a majority of the precincts in the district were voting red. However we had an issue with translating an if and only if statement into a linear constraint. We initially came up with a linear constraint that would check if every district had the majority: $\forall i \in [k]$

$$\frac{(\sum_{j=1}^{n} c_j x_{ij})}{(\sum_{j=1}^{n} x_{ij})} \geqslant \frac{1}{2}$$
$$2\sum_{j=1}^{n} c_j x_{ij} \geqslant \sum_{j=1}^{n} x_{ij}$$

After talking with Dr. Conroy we used a previous example in class about if and only if statements and inequality logic statements in order to formulate the final objective function:

We first added a new variable

$$y_i = \begin{cases} 1 & \text{if more than half of the precincts in district } i \text{ voted for red, } \forall i \in \{1, ..., k\} \\ 0 & \text{otherwise} \end{cases}$$ and

an arbitrarily large number $M \in \mathbb{R}^n$. Then defined constraints to limit y_i to 1 and 0 to show when district i voted red or now.

$$My_i \geqslant 2\sum_{j=1}^{n} c_j x_{ij} - \sum_{j=1}^{n} x_{ij}$$

which forces y_i to be 1 if there is a majority of the precincts voting red.

$$M(y_i - 1) < 2\sum_{j=1}^{n} c_j x_{ij} - \sum_{j=1}^{n} x_{ij}$$

This constraint forces y_i to be 0 if there is not a majority of the precincts voting red.

3.3.2 Challenge 2: Contiguity

Contiguity is the capability for all points in a district to be directly adjacent to any one point in the same district. This constraint is important to simulate real world conditions

because voting districts in the real world cannot be separated and must satisfy contiguity.

During the course of our research we discovered that there were methods to create a contiguity constraint. For instance, Justin William's paper on a programming model for contiguous land acquisition uses several concepts from graph theory and network optimization in order to construct the contiguity constraint.[3]

However, the problem with utilizing William's constraints were that our understanding of planar graphs and primal/dual graphs in a network were not enough to use his work as a foundation for our contiguity constraint.

Another method we found was Nemoto's paper that used a network flow approach. He found that the classic flow balance constraint guarantees contiguity.[4] However, the issue with using Nemoto's approach was that network flow is used on direct graphs of which ours wasn't. This can be seen in the graph visualization of our Arizona data in Appendix A.6 Furthermore, the network flow balance constraint depends on having weighted edges which we did not have.

Our attempt on formulating constraints for this contiguity condition varied from using a different kind of data structure to creating additional conditions that may have been able to prevent sub-cycles, although it may have not proven adjacency. Our initial attempt on tackling this issue started with formulating our model based off 2-D arrays. We thought that given the 2-D dimensions of every district and their respective locations in a larger encompassing matrix, we would be able to check that for all precincts there is at least one precinct located within a distance of 1, i.e, they are adjacent. Another method we attempted to use is searching all precincts' top, bottom, left, and right for an existing precinct. If all precincts passed this test, we can confirm that they are contiguous. However, the problem with this approach is that it is not only extremely cost heavy and low performance, but

[3]Justin C. Williams. "A Zero-One Programming Model for Contiguous Land Acquisition". In: *Geographical Analysis* 34.4 (), pp. 330–349. DOI: 10.1111/j.1538-4632.2002.tb01093.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1538-4632.2002.tb01093.x.

[4]Hotta K Nemoto T. "Modelling and solution of the problem of optimal electoral districting". In: *CommunOR Soc Jpn* 48 (), pp. 300–306. URL: http://www.orsj.or.jp/~archive/pdf/bul/Vol.4804300.pdf.

also the underlying assumption is that we already know about the locations and shapes of districts, which we do not have because we do not have the constraints.

Our final attempt was to figure out a constraint that will force precincts to be located within the same district by eliminating the possibility of sub-cycles, these prove adjacency. The problems with this approach started from formulating equations itself to the fact that not only each precinct, but also each district can be sub-cycles and we again contradicted the assumption that we already knew the shape and location within a matrix. Therefore, we decided to model our LP using vertices. Ultimately, we were unable to generate a contiguity constraint that is critical to simulate real world conditions.

3.3.3 Challenge 3: Tie-Breaking

When we first built our model, the original constraint (7) and (8) were

$$My_i \geqslant 2b_i - d_i, \qquad \forall i \in \{1, ..., k\}$$

$$M(y_i - 1) < 2b_i - d_i, \qquad \forall i \in \{1, ..., k\}$$

However, after running several tests on lpsolve using the input file generated by these constraints, we found a "tie" issue. When the number of precincts that vote for red is the same as that of precincts who vote for blue in a single district. lpsolve considered this situation as $y_i = 1$ (red wins) which is not what we want we necessarily.

After talking with Dr.Conroy, he suggested us to subtract $\frac{1}{2}$ only on the right-hand side of the constraint (8). However, the problem didn't solve after we tried that, so we subtracted $\frac{1}{2}$ in both constraints. This time, it turns out that the "tie" issue was resolved.

We think the reason why we needed to subtract $\frac{1}{2}$ in those constraints is because it will give lpsolve a more precise range to get the value of y_i.

4 Solution

4.1 Lpsolve Input File

To solve the LP using lpsolve, we use Python to create an input file defining the LP model.

Before generating the input file, we created and assigned appropriate values to n, total number of precincts; k, number of districts; and M, a big value variable in Python. Also, we needed to randomly generate a binary variable c_j for each precinct j where $c_j = 1$ if the precinct votes for red and $c_j = 0$ otherwise.

The general version of input file is:

```
max: y_1 + y_2 + ... + y_k;
(n lines of the following type:
ensure that each block can only be assigned to exactly one district)
x_1_1 + x_2_1 + ... + x_k_1 = 1;
.
.
x_1_n + x_2_n + ... + x_k_n = 1;
(k lines of the following type:
ensure that all districts are assigned)
x_1_1 + x_1_2 + x_1_3 + ... + x_1_n >= 1;
x_2_1 + x_2_2 + x_2_3 + ... + x_2_n >= 1;
.
.
x_k_1 + x_k_2 + x_k_3 + ... + x_k_n >= 1;
(k lines of the following type:
ensure upper bound of the number of precincts in each district)
x_1_1 + x_1_2 + x_1_3 + ... + x_1_n <= n/k + 1;
x_2_1 + x_2_2 + x_2_3 + ... + x_2_n <= n/k + 1;
.
.
x_k_1 + x_k_2 + x_k_3 + ... + x_k_n <= n/k + 1;
(k lines of the following type:
ensure lower bound of the number of precincts in each district)
x_1_1 + x_1_2 + x_1_3 + ... + x_1_n >= n/k - 1;
x_2_1 + x_2_2 + x_2_3 + ... + x_2_n >= n/k - 1;
.
.
x_k_1 + x_k_2 + x_k_3 + ... + x_k_n >= n/k - 1;
(k lines of the following type:
generate a variable 'd_i' for the number of precincts in each district i)
d_1 = x_1_1 + x_1_2 + x_1_3 + ... + x_1_n;
d_2 = x_2_1 + x_2_2 + x_2_3 + ... + x_2_n;
.
.
d_k = x_k_1 + x_k_2 + x_k_3 + ... + x_k_n;
(k lines of the following type:
generate a variable 'b_i' for the number of precincts in each district i that is red
```

```
c[j] here denotes the random bianry variable
where c_j = 1 if precinct j votes for red and c_j = 0 otherwise)
b_1 = c[1] * x_1_1 + c[2] * x_1_2 + c[3] * x_1_3 + ... + c[n] * x_1_n;
b_2 = c[1] * x_2_1 + c[2] * x_2_2 + c[3] * x_2_3 + ... + c[n] * x_2_n;
.
.
b_k = c[1] * x_k_1 + c[2] * x_k_2 + c[3] * x_k_3 + ... + c[n] * x_k_n;
(k lines of the following type:
ensure that y_i = 0 if red does not win in a district i)
M * y_1 >= 2 * b_1 - d_1 - 0.5;
M * y_2 >= 2 * b_2 - d_2 - 0.5;
.
.
M * y_k >= 2 * b_k - d_k - 0.5;
(k lines of the following type:
ensure that y_i = 1 if red wins in a district i)
M * y_1 - M <= 2 * b_1 - d_1 - 0.5;
M * y_2 - M <= 2 * b_2 - d_2 - 0.5;
.
.
M * y_k - M <= 2 * b_k - d_k - 0.5;
bin x_1_1, x_1_2, ..., x_1_n, x_2_1, x_2_2, ..., x_2_n, ...,
x_k_1, x_k_2, ..., x_k_n, y_1, y_2, ..., y_k;
```

For the input file we used with the Arizona Data it can be seen in Appendix A.2

4.2 Lpsolve Solution Output

Rather than assigning values to the variables, we used the closest real world data we can use to test the model. Arizona has 15 precincts and 9 districts (data from Politico's 2018 US Senate coverage[5]). So then we assigned $n = 15$, $k = 9$, $M = 1000000$, $c = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1]$ in our python code to generate the input file for lpsolve.

After that, we ran lpsolve on this file (p1Input.txt) with the command: ./lp_solve -ia p1Input.txt, which suppressed output of zero variables, making things easier to read.
It takes a while to run, and we get the output (all other variables equal zero) showing in Appendix A.3.

On a 2015 Macbook Pro Retina running macOS High Sierra (10.13.6) it took 8 minutes and 17 seconds to run LPSolve.

[5]Politico, *Arizona Election Results 2018: Live Midterm Map by County & Analysis.*

5 Commentary on Solution

5.1 Solution Meaning

The value of the objective function tells us that the optimal solution for our model has 6 districts out of 9 total voting red.

From the value of y_i, we know that red wins in district 1, 2, 4, 5, 6 and 7. The value of x_{ij} shows us which precinct is assigned to which district. Recall that d_i and b_i indicate the number of precincts and the number of precincts that vote for red in each district, respectively.

District	Precinct		Result
1	1(red)	12(red)	red wins
2	11(red)	15(red)	red wins
3	4	7	blue wins
4	13(red)		red wins
5	9(red)		red wins
6	8(red)		red wins
7	3(red)	5(red)	red wins
8	2(red)	6	tie
9	10	14	blue wins

Figure 3: Precincts and assigned Districts in Arizona Data

These results fit the value of $d_i, i \in \{1, \cdots, 9\}$.

5.2 Binding Constraints

In the case of Arizona where n = 15 and k = 9, the district size is bounded between 0.67 and 2.67. Since the size can only be an integer, the possible values for the size of the district is 1 or 2. After checking $d_i, i \in \{1, \cdots, 9\}$, the number of precincts in each district, we know

that when $i = 1$, 2, 7 and 8, constraint (3) is binding. When i = 3, 4, 5, 6 and 9, constraint (4) is binding.

5.3 Feasibility of Solution in Real World

In order for our solution to be realistic, it must be able to be applied to real world voting districts. Therefore, our solution is not realistic as we are missing several key constraints in order to apply it to Arizona.

The first constraint is that real world voting districts are contiguous while our solution is not. An example is in district 1 where precincts 1 and 12 were added together. However, according to the graph representation in A.6 we can see that 1 and 12 are in completely different locations and are not connected in any manner.

The second constraint is that the US Constitution requires that each district must have about the same population. While, constraints (3) and (4) in our model ensure that the number of precincts are as close as possible we did not take into account the population.[6]

The third constraint is with the specific state we chose. Arizona requires that that the districts are compact, follow political boundaries, and preserve communities of interest. We have taken none of these constraints into account thus our redrawing of the district lines are unrealistic.[7]

6 Variations

We can create related problems by modifying our original problem. The intent is to create variations that allow us to reiterate, reconfirm, and realize both quantitative and qualitative components of our model that we may or may not have anticipated. Through this process, we can analyze how our model may have to be updated.

[6]Loyola Law School. *Where the Lines Are Drawn - Congressional Districts.* URL: `redistricting.lls.edu/where-tablefed.php.` (visited on 02/09/2019).

[7]School, *Where the Lines Are Drawn - Congressional Districts.*

6.1 Variation 1: $n = 23$, $k = 4$

This variations considers the possibility of getting an even number of Republican districts and an even number of Democratic districts, ultimately getting a tie. For this specific variation, we were lucky enough to not get a tie. The result we get is the following:

District	Precinct						Result
1	2(red)	7(red)	17(red)	22(red)	19	20	red wins
2	5(red)	10(red)	16(red)	18(red)	1	3	red wins
3	13(red)	14(red)	23(red)	9	12		red wins
4	4	6	8	11	15	21	blue wins

Figure 4: Precincts and assigned Districts in Variation One

On a ThinkPad T470s Signature the runtime was 9.275 seconds. We get 3 red districts and 1 blue district with all blues. So we see that for this example we do not get even numbers of party districts because the lpsolve searches for the optimal result given the constraints. The lpsolve output can be seen in Appendix A.4

Also notice that six precincts go to district 1,2,4 while five go to district 3. Our constraint that sets a lower and upper bound to the difference in each district size comes into action. Something we realize that can be improved is providing additional constraints that can potentially eliminate the chance that we may get a tie in terms of district party allocation.

6.2 Variation 2: $n = 100$, $k = 3$

This variation is to discover if a large difference in the number of precincts and number of districts will provide interesting insight into our model. The result is the following:

District	Red Precincts	Blue Precincts	Result
1	17	16	red wins
2	17	16	red wins
3	13	21	blue wins
Total	47	53	red wins

Figure 5: Number of Precincts Assigned to Districts in Variation Two

On a ThinkPad T470s Signature the runtime was 0.065 seconds. The runtime is very short because the number of districts limited to is very small, then, provided the randomly assigned colors of each precinct, the constraints are satisfied swiftly because that are also binary variables. The variables with an effective coefficient of zero will simply be ignored by the lpsolve to optimize performance. Looking at the result, District 1 and 2 has 33 precincts and 17 red votes each, ultimately winning the district red. District 3 has 34 precincts and 13 red votes, winning blue. This problem follows the same procedures of how other variations are solved. And in this case, total votes for red is 47 out of 100, but red wins as a result even with fewer votes. That is our goal to maximize the 'unfairness'. The size of each district is similar and given the votes of each precinct the maximum reds are allocated to some districts to win red. However, the interesting insight is that, given the abundant precincts, we can clearly see the lpsolve's behavior that after allocating the red precincts, it "dumps" all the remaining blue votes to the leftover districts because we want to simply optimizing the number of red districts.

6.3 Variation 3: No Size Constraints

We can also make variations by changing or removing constraints from our model. For example, if we remove the size constraints (3) and (4), and generate a new input file, then apply the model to the Arizona voting data using lpsolve, the result is the following:

District	Precinct			Result
1	4	10	14	blue wins
2	13(red)			red wins
3	12(red)			red wins
4	11(red)			red wins
5	5(red)	8(red)	7	red wins
6	1(red)	9(red)	6	red wins
7	3(red)			red wins
8	2(red)			red wins
9	15(red)			red wins

Figure 6: Precincts and Assigned Districts with Arizona Data in Variation Three

The output indicates that the maximum number of districts that vote for red is now 8 instead of 6 as it is in the original model. From the value of $y_i's$, we know that red wins in all of the districts except district 1. The lpsolve assigns only blue votes into district 1 with size of 3, and there are 6 districts assigned with only 1 red precinct thus red wins that district. Compared to the original model with same data, the runtime for this variation has been greatly reduced. On a 2015 Macbook Pro Retina running macOS High Sierra (10.13.6) it took around 4 seconds to get an output on lpsolve. The lpsolve output can be seen in Appendix A.5

Therefore, removing size constrains for districts makes it much easier to favor one party. However, it may lead to great difference in district sizes when the amount of districts is small. If there are k districts, the lpsolve might generate as many districts as possible with only one red precincts and assign the other precincts into the remaining district(s).

7 Conclusion

From our project we can conclude that Gerrymandering is a complicated problem to model and solve. From the criteria of Gerrymandering, we know that it is an optimization problem and we can model it using Linear Programming. However, the criteria of many Gerrymandering research papers is to create fair and legal political districts, so it is hard for us to maximize the districts who vote for red from scratch. Furthermore, we found that in order to truly model Gerrymandering, a deeper understanding of graph theory is required so that constraints such as contiguity can be created.

Our model is able to successfully generate districts which red wins under some reasonable initial conditions. However, the run-time required for LPSolve increases if the initial conditions are too complicated. There are also situations where the number of precincts voting for red is too few while the number of districts is large, our model fail to have an optimized solution. In that case, the value of our objective function is 0. In addition, our model does not incorporate contiguity and compactness. From this we can conclude that our Linear Program implementation is not the most optimal and requires careful fine-tuning in order to hold up to real world standards. However, being able to create model to represent a different perspective on a abstract problem was eyeopening to us on how important district lines are for voting.

References

Hess, S. W. et al. "Nonpartisan Political Redistricting by Computer". In: *Operations Research* 13.6 (1965), pp. 998–1006.

Nemoto T, Hotta K. "Modelling and solution of the problem of optimal electoral districting". In: *CommunOR Soc Jpn* 48 (), pp. 300–306. URL: `http://www.orsj.or.jp/~archive/pdf/bul/Vol.4804300.pdf`.

Politico. *Arizona Election Results 2018: Live Midterm Map by County & Analysis.* URL: www.politico.com/election-results/2018/arizona/. (visited on 02/09/2019).

School, Loyola Law. *Where the Lines Are Drawn - Congressional Districts.* URL: redistricting.lls.edu/where-tablefed.php. (visited on 02/09/2019).

Williams, Justin C. "A Zero-One Programming Model for Contiguous Land Acquisition". In: *Geographical Analysis* 34.4 (), pp. 330–349. DOI: 10.1111/j.1538-4632.2002.tb01093.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1538-4632.2002.tb01093.x.

A Appendix

A.1 Python Code

Python Code to generate a Linear Program Model and print lpsolve input file

```
# project 1
# linear optimization of gerrymandering

import random

# Assign number of precincts n, number of districts k, and a big value M.
n = 15
k = 9
M = 1000000

c = [1,1,1,0,1,0,0,1,1,0,1,1,1,0,1]

# This block randomly generates a binary variable c_j where c_j = 1 if the
    precinct votes for red.
#c = [0] * n
#for j in range(n):
#    c[j] = random.randint(0,1)
#print(c)

text_file = open("p1Input.txt", "w")

lp = ""

# This block generates the objective function.

lp += "max: "
for i in range(1, k+1):
    if i != k:
        lp += "y_{} + ".format(i)
lp += "y_{};\n".format(k)

# This block generates the first constraint (Ensuring that each block can only
    be assigned to one district).
for j in range(1,n+1):
    for i in range(1, k+1):
        if i != k:
            lp+= "x_{}_{} + ".format(i,j)
    lp+="x_{}_{} = 1; \n".format(k,j)

# This block generates the second constraint (All districts are assigned).
for i in range(1,k+1):
    for j in range(1,n+1):
        if j != n:
            lp += "x_{}_{} + ".format(i,j)
    lp += "x_{}_{} >= 1; \n".format(i,n)

# This block generates the third constraint (upper bound of the number of
    precincts in each district).
```

```
for i in range(1,k+1):
    for j in range(1,n+1):
        if j != n:
            lp += "x_{}_{} + ".format(i,j)
    lp += "x_{}_{} <= {}; \n".format(i,n,n/k+1)

# This block generates the fourth constraint (lower bound of the number of precincts in each district).
for i in range(1,k+1):
    for j in range(1,n+1):
        if j != n:
            lp += "x_{}_{} + ".format(i,j)
    lp += "x_{}_{} >= {}; \n".format(i,n,n/k-1)

# This block generates a variable 'd_i' for the number of precincts in each district 'i'.
for i in range(1, k+1):
    lp += "d_{} = ".format(i)
    for j in range(1, n+1):
        if j != n:
            lp += "x_{}_{} + ".format(i,j)
    lp+= "x_{}_{}; \n".format(i,n)

# This block generates a variable 'b_i' for the number of precincts in each district 'i' that is red.
for i in range(1,k+1):
    lp += "b_{} = ".format(i)
    for j in range(1,n+1):
        if j != n:
            lp += "{} * x_{}_{} + ".format(c[j-1],i,j)
    lp += "{} * x_{}_{}; \n".format(c[n-1],i,n)


# This block generates the constraint to ensure that y_i is 1/0 if a district is majority red (more than half) or not.
for i in range(1, k+1):
    lp += "{} * y_{} >= 2 * b_{} - d_{} - 0.5; \n".format(M,i,i,i)

for i in range(1, k+1):
    lp += "{} * y_{} - {} < 2 * b_{} - d_{} - 0.5; \n".format(M,i,M,i,i)


# This block generates the binary constraint (All variables are binary).
lp += "bin "
for i in range(1,k+1):
    for j in range(1,n+1):
        if i != k or j != n:
            lp += "x_{}_{}, ".format(i,j)
lp += "x_{}_{}, ".format(k,n)

for i in range(1, k+1):
    if i != k:
        lp += "y_{}, ".format(i)
lp += "y_{};".format(k)

```

```
97
98 text_file.write(lp)
99 text_file.close()
```

A.2 Input File of Lpsolve for Arizona Data

```
max: y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 + y_8 + y_9;
x_1_1 + x_2_1 + x_3_1 + x_4_1 + x_5_1 + x_6_1 + x_7_1 + x_8_1 + x_9_1 = 1;
x_1_2 + x_2_2 + x_3_2 + x_4_2 + x_5_2 + x_6_2 + x_7_2 + x_8_2 + x_9_2 = 1;
x_1_3 + x_2_3 + x_3_3 + x_4_3 + x_5_3 + x_6_3 + x_7_3 + x_8_3 + x_9_3 = 1;
x_1_4 + x_2_4 + x_3_4 + x_4_4 + x_5_4 + x_6_4 + x_7_4 + x_8_4 + x_9_4 = 1;
x_1_5 + x_2_5 + x_3_5 + x_4_5 + x_5_5 + x_6_5 + x_7_5 + x_8_5 + x_9_5 = 1;
x_1_6 + x_2_6 + x_3_6 + x_4_6 + x_5_6 + x_6_6 + x_7_6 + x_8_6 + x_9_6 = 1;
x_1_7 + x_2_7 + x_3_7 + x_4_7 + x_5_7 + x_6_7 + x_7_7 + x_8_7 + x_9_7 = 1;
x_1_8 + x_2_8 + x_3_8 + x_4_8 + x_5_8 + x_6_8 + x_7_8 + x_8_8 + x_9_8 = 1;
x_1_9 + x_2_9 + x_3_9 + x_4_9 + x_5_9 + x_6_9 + x_7_9 + x_8_9 + x_9_9 = 1;
x_1_10 + x_2_10 + x_3_10 + x_4_10 + x_5_10 + x_6_10 + x_7_10 + x_8_10 + x_9_10 = 1;
x_1_11 + x_2_11 + x_3_11 + x_4_11 + x_5_11 + x_6_11 + x_7_11 + x_8_11 + x_9_11 = 1;
x_1_12 + x_2_12 + x_3_12 + x_4_12 + x_5_12 + x_6_12 + x_7_12 + x_8_12 + x_9_12 = 1;
x_1_13 + x_2_13 + x_3_13 + x_4_13 + x_5_13 + x_6_13 + x_7_13 + x_8_13 + x_9_13 = 1;
x_1_14 + x_2_14 + x_3_14 + x_4_14 + x_5_14 + x_6_14 + x_7_14 + x_8_14 + x_9_14 = 1;
x_1_15 + x_2_15 + x_3_15 + x_4_15 + x_5_15 + x_6_15 + x_7_15 + x_8_15 + x_9_15 = 1;
x_1_1 + x_1_2 + x_1_3 + x_1_4 + x_1_5 + x_1_6 + x_1_7 + x_1_8 + x_1_9 + x_1_10
+ x_1_11 + x_1_12 + x_1_13 + x_1_14 + x_1_15 >= 1;
x_2_1 + x_2_2 + x_2_3 + x_2_4 + x_2_5 + x_2_6 + x_2_7 + x_2_8 + x_2_9 + x_2_10
+ x_2_11 + x_2_12 + x_2_13 + x_2_14 + x_2_15 >= 1;
x_3_1 + x_3_2 + x_3_3 + x_3_4 + x_3_5 + x_3_6 + x_3_7 + x_3_8 + x_3_9 + x_3_10
+ x_3_11 + x_3_12 + x_3_13 + x_3_14 + x_3_15 >= 1;
x_4_1 + x_4_2 + x_4_3 + x_4_4 + x_4_5 + x_4_6 + x_4_7 + x_4_8 + x_4_9 + x_4_10
+ x_4_11 + x_4_12 + x_4_13 + x_4_14 + x_4_15 >= 1;
x_5_1 + x_5_2 + x_5_3 + x_5_4 + x_5_5 + x_5_6 + x_5_7 + x_5_8 + x_5_9 + x_5_10
+ x_5_11 + x_5_12 + x_5_13 + x_5_14 + x_5_15 >= 1;
x_6_1 + x_6_2 + x_6_3 + x_6_4 + x_6_5 + x_6_6 + x_6_7 + x_6_8 + x_6_9 + x_6_10
+ x_6_11 + x_6_12 + x_6_13 + x_6_14 + x_6_15 >= 1;
x_7_1 + x_7_2 + x_7_3 + x_7_4 + x_7_5 + x_7_6 + x_7_7 + x_7_8 + x_7_9 + x_7_10
+ x_7_11 + x_7_12 + x_7_13 + x_7_14 + x_7_15 >= 1;
x_8_1 + x_8_2 + x_8_3 + x_8_4 + x_8_5 + x_8_6 + x_8_7 + x_8_8 + x_8_9 + x_8_10
+ x_8_11 + x_8_12 + x_8_13 + x_8_14 + x_8_15 >= 1;
x_9_1 + x_9_2 + x_9_3 + x_9_4 + x_9_5 + x_9_6 + x_9_7 + x_9_8 + x_9_9 + x_9_10
+ x_9_11 + x_9_12 + x_9_13 + x_9_14 + x_9_15 >= 1;
x_1_1 + x_1_2 + x_1_3 + x_1_4 + x_1_5 + x_1_6 + x_1_7 + x_1_8 + x_1_9 + x_1_10
+ x_1_11 + x_1_12 + x_1_13 + x_1_14 + x_1_15 <= 2.666666666666667;
x_2_1 + x_2_2 + x_2_3 + x_2_4 + x_2_5 + x_2_6 + x_2_7 + x_2_8 + x_2_9 + x_2_10
+ x_2_11 + x_2_12 + x_2_13 + x_2_14 + x_2_15 <= 2.666666666666667;
x_3_1 + x_3_2 + x_3_3 + x_3_4 + x_3_5 + x_3_6 + x_3_7 + x_3_8 + x_3_9 + x_3_10
+ x_3_11 + x_3_12 + x_3_13 + x_3_14 + x_3_15 <= 2.666666666666667;
x_4_1 + x_4_2 + x_4_3 + x_4_4 + x_4_5 + x_4_6 + x_4_7 + x_4_8 + x_4_9 + x_4_10
+ x_4_11 + x_4_12 + x_4_13 + x_4_14 + x_4_15 <= 2.666666666666667;
x_5_1 + x_5_2 + x_5_3 + x_5_4 + x_5_5 + x_5_6 + x_5_7 + x_5_8 + x_5_9 + x_5_10
+ x_5_11 + x_5_12 + x_5_13 + x_5_14 + x_5_15 <= 2.666666666666667;
x_6_1 + x_6_2 + x_6_3 + x_6_4 + x_6_5 + x_6_6 + x_6_7 + x_6_8 + x_6_9 + x_6_10
+ x_6_11 + x_6_12 + x_6_13 + x_6_14 + x_6_15 <= 2.666666666666667;
x_7_1 + x_7_2 + x_7_3 + x_7_4 + x_7_5 + x_7_6 + x_7_7 + x_7_8 + x_7_9 + x_7_10
+ x_7_11 + x_7_12 + x_7_13 + x_7_14 + x_7_15 <= 2.666666666666667;
x_8_1 + x_8_2 + x_8_3 + x_8_4 + x_8_5 + x_8_6 + x_8_7 + x_8_8 + x_8_9 + x_8_10
+ x_8_11 + x_8_12 + x_8_13 + x_8_14 + x_8_15 <= 2.666666666666667;
x_9_1 + x_9_2 + x_9_3 + x_9_4 + x_9_5 + x_9_6 + x_9_7 + x_9_8 + x_9_9 + x_9_10
+ x_9_11 + x_9_12 + x_9_13 + x_9_14 + x_9_15 <= 2.666666666666667;
x_1_1 + x_1_2 + x_1_3 + x_1_4 + x_1_5 + x_1_6 + x_1_7 + x_1_8 + x_1_9 + x_1_10
```

```
+ x_1_11 + x_1_12 + x_1_13 + x_1_14 + x_1_15 >= 0.6666666666666667;
x_2_1 + x_2_2 + x_2_3 + x_2_4 + x_2_5 + x_2_6 + x_2_7 + x_2_8 + x_2_9 + x_2_10
+ x_2_11 + x_2_12 + x_2_13 + x_2_14 + x_2_15 >= 0.6666666666666667;
x_3_1 + x_3_2 + x_3_3 + x_3_4 + x_3_5 + x_3_6 + x_3_7 + x_3_8 + x_3_9 + x_3_10
+ x_3_11 + x_3_12 + x_3_13 + x_3_14 + x_3_15 >= 0.6666666666666667;
x_4_1 + x_4_2 + x_4_3 + x_4_4 + x_4_5 + x_4_6 + x_4_7 + x_4_8 + x_4_9 + x_4_10
+ x_4_11 + x_4_12 + x_4_13 + x_4_14 + x_4_15 >= 0.6666666666666667;
x_5_1 + x_5_2 + x_5_3 + x_5_4 + x_5_5 + x_5_6 + x_5_7 + x_5_8 + x_5_9 + x_5_10
+ x_5_11 + x_5_12 + x_5_13 + x_5_14 + x_5_15 >= 0.6666666666666667;
x_6_1 + x_6_2 + x_6_3 + x_6_4 + x_6_5 + x_6_6 + x_6_7 + x_6_8 + x_6_9 + x_6_10
+ x_6_11 + x_6_12 + x_6_13 + x_6_14 + x_6_15 >= 0.6666666666666667;
x_7_1 + x_7_2 + x_7_3 + x_7_4 + x_7_5 + x_7_6 + x_7_7 + x_7_8 + x_7_9 + x_7_10
+ x_7_11 + x_7_12 + x_7_13 + x_7_14 + x_7_15 >= 0.6666666666666667;
x_8_1 + x_8_2 + x_8_3 + x_8_4 + x_8_5 + x_8_6 + x_8_7 + x_8_8 + x_8_9 + x_8_10
+ x_8_11 + x_8_12 + x_8_13 + x_8_14 + x_8_15 >= 0.6666666666666667;
x_9_1 + x_9_2 + x_9_3 + x_9_4 + x_9_5 + x_9_6 + x_9_7 + x_9_8 + x_9_9 + x_9_10
+ x_9_11 + x_9_12 + x_9_13 + x_9_14 + x_9_15 >= 0.6666666666666667;
d_1 = x_1_1 + x_1_2 + x_1_3 + x_1_4 + x_1_5 + x_1_6 + x_1_7 + x_1_8 + x_1_9 + x_1_10
+ x_1_11 + x_1_12 + x_1_13 + x_1_14 + x_1_15;
d_2 = x_2_1 + x_2_2 + x_2_3 + x_2_4 + x_2_5 + x_2_6 + x_2_7 + x_2_8 + x_2_9 + x_2_10
+ x_2_11 + x_2_12 + x_2_13 + x_2_14 + x_2_15;
d_3 = x_3_1 + x_3_2 + x_3_3 + x_3_4 + x_3_5 + x_3_6 + x_3_7 + x_3_8 + x_3_9 + x_3_10
+ x_3_11 + x_3_12 + x_3_13 + x_3_14 + x_3_15;
d_4 = x_4_1 + x_4_2 + x_4_3 + x_4_4 + x_4_5 + x_4_6 + x_4_7 + x_4_8 + x_4_9 + x_4_10
+ x_4_11 + x_4_12 + x_4_13 + x_4_14 + x_4_15;
d_5 = x_5_1 + x_5_2 + x_5_3 + x_5_4 + x_5_5 + x_5_6 + x_5_7 + x_5_8 + x_5_9 + x_5_10
+ x_5_11 + x_5_12 + x_5_13 + x_5_14 + x_5_15;
d_6 = x_6_1 + x_6_2 + x_6_3 + x_6_4 + x_6_5 + x_6_6 + x_6_7 + x_6_8 + x_6_9 + x_6_10
+ x_6_11 + x_6_12 + x_6_13 + x_6_14 + x_6_15;
d_7 = x_7_1 + x_7_2 + x_7_3 + x_7_4 + x_7_5 + x_7_6 + x_7_7 + x_7_8 + x_7_9 + x_7_10
+ x_7_11 + x_7_12 + x_7_13 + x_7_14 + x_7_15;
d_8 = x_8_1 + x_8_2 + x_8_3 + x_8_4 + x_8_5 + x_8_6 + x_8_7 + x_8_8 + x_8_9 + x_8_10
+ x_8_11 + x_8_12 + x_8_13 + x_8_14 + x_8_15;
d_9 = x_9_1 + x_9_2 + x_9_3 + x_9_4 + x_9_5 + x_9_6 + x_9_7 + x_9_8 + x_9_9 + x_9_10
+ x_9_11 + x_9_12 + x_9_13 + x_9_14 + x_9_15;
b_1 = 1 * x_1_1 + 1 * x_1_2 + 1 * x_1_3 + 0 * x_1_4 + 1 * x_1_5 + 0 * x_1_6 + 0 * x_1_7
+ 1 * x_1_8 + 1 * x_1_9 + 0 * x_1_10 + 1 * x_1_11 + 1 * x_1_12 + 1 * x_1_13 + 0 * x_1_14
+ 1 * x_1_15;
b_2 = 1 * x_2_1 + 1 * x_2_2 + 1 * x_2_3 + 0 * x_2_4 + 1 * x_2_5 + 0 * x_2_6 + 0 * x_2_7
+ 1 * x_2_8 + 1 * x_2_9 + 0 * x_2_10 + 1 * x_2_11 + 1 * x_2_12 + 1 * x_2_13 + 0 * x_2_14
+ 1 * x_2_15;
b_3 = 1 * x_3_1 + 1 * x_3_2 + 1 * x_3_3 + 0 * x_3_4 + 1 * x_3_5 + 0 * x_3_6 + 0 * x_3_7
+ 1 * x_3_8 + 1 * x_3_9 + 0 * x_3_10 + 1 * x_3_11 + 1 * x_3_12 + 1 * x_3_13 + 0 * x_3_14
+ 1 * x_3_15;
b_4 = 1 * x_4_1 + 1 * x_4_2 + 1 * x_4_3 + 0 * x_4_4 + 1 * x_4_5 + 0 * x_4_6 + 0 * x_4_7
+ 1 * x_4_8 + 1 * x_4_9 + 0 * x_4_10 + 1 * x_4_11 + 1 * x_4_12 + 1 * x_4_13 + 0 * x_4_14
+ 1 * x_4_15;
b_5 = 1 * x_5_1 + 1 * x_5_2 + 1 * x_5_3 + 0 * x_5_4 + 1 * x_5_5 + 0 * x_5_6 + 0 * x_5_7
+ 1 * x_5_8 + 1 * x_5_9 + 0 * x_5_10 + 1 * x_5_11 + 1 * x_5_12 + 1 * x_5_13 + 0 * x_5_14
+ 1 * x_5_15;
b_6 = 1 * x_6_1 + 1 * x_6_2 + 1 * x_6_3 + 0 * x_6_4 + 1 * x_6_5 + 0 * x_6_6 + 0 * x_6_7
+ 1 * x_6_8 + 1 * x_6_9 + 0 * x_6_10 + 1 * x_6_11 + 1 * x_6_12 + 1 * x_6_13 + 0 * x_6_14
+ 1 * x_6_15;
b_7 = 1 * x_7_1 + 1 * x_7_2 + 1 * x_7_3 + 0 * x_7_4 + 1 * x_7_5 + 0 * x_7_6 + 0 * x_7_7
+ 1 * x_7_8 + 1 * x_7_9 + 0 * x_7_10 + 1 * x_7_11 + 1 * x_7_12 + 1 * x_7_13 + 0 * x_7_14
```

```
+ 1 * x_7_15;
b_8 = 1 * x_8_1 + 1 * x_8_2 + 1 * x_8_3 + 0 * x_8_4 + 1 * x_8_5 + 0 * x_8_6 + 0 * x_8_7
+ 1 * x_8_8 + 1 * x_8_9 + 0 * x_8_10 + 1 * x_8_11 + 1 * x_8_12 + 1 * x_8_13 + 0 * x_8_14
+ 1 * x_8_15;
b_9 = 1 * x_9_1 + 1 * x_9_2 + 1 * x_9_3 + 0 * x_9_4 + 1 * x_9_5 + 0 * x_9_6 + 0 * x_9_7
+ 1 * x_9_8 + 1 * x_9_9 + 0 * x_9_10 + 1 * x_9_11 + 1 * x_9_12 + 1 * x_9_13 + 0 * x_9_14
+ 1 * x_9_15;
1000000 * y_1 >= 2 * b_1 - d_1 - 0.5;
1000000 * y_2 >= 2 * b_2 - d_2 - 0.5;
1000000 * y_3 >= 2 * b_3 - d_3 - 0.5;
1000000 * y_4 >= 2 * b_4 - d_4 - 0.5;
1000000 * y_5 >= 2 * b_5 - d_5 - 0.5;
1000000 * y_6 >= 2 * b_6 - d_6 - 0.5;
1000000 * y_7 >= 2 * b_7 - d_7 - 0.5;
1000000 * y_8 >= 2 * b_8 - d_8 - 0.5;
1000000 * y_9 >= 2 * b_9 - d_9 - 0.5;
1000000 * y_1 - 1000000 < 2 * b_1 - d_1 - 0.5;
1000000 * y_2 - 1000000 < 2 * b_2 - d_2 - 0.5;
1000000 * y_3 - 1000000 < 2 * b_3 - d_3 - 0.5;
1000000 * y_4 - 1000000 < 2 * b_4 - d_4 - 0.5;
1000000 * y_5 - 1000000 < 2 * b_5 - d_5 - 0.5;
1000000 * y_6 - 1000000 < 2 * b_6 - d_6 - 0.5;
1000000 * y_7 - 1000000 < 2 * b_7 - d_7 - 0.5;
1000000 * y_8 - 1000000 < 2 * b_8 - d_8 - 0.5;
1000000 * y_9 - 1000000 < 2 * b_9 - d_9 - 0.5;
bin x_1_1, x_1_2, x_1_3, x_1_4, x_1_5, x_1_6, x_1_7, x_1_8, x_1_9, x_1_10, x_1_11,
x_1_12, x_1_13, x_1_14, x_1_15, x_2_1, x_2_2, x_2_3, x_2_4, x_2_5, x_2_6, x_2_7, x_2_8,
x_2_9, x_2_10, x_2_11, x_2_12, x_2_13, x_2_14, x_2_15, x_3_1, x_3_2, x_3_3, x_3_4,
x_3_5, x_3_6, x_3_7, x_3_8, x_3_9, x_3_10, x_3_11, x_3_12, x_3_13, x_3_14, x_3_15,
x_4_1, x_4_2, x_4_3, x_4_4, x_4_5, x_4_6, x_4_7, x_4_8, x_4_9, x_4_10, x_4_11, x_4_12,
x_4_13, x_4_14, x_4_15, x_5_1, x_5_2, x_5_3, x_5_4, x_5_5, x_5_6, x_5_7, x_5_8, x_5_9,
x_5_10, x_5_11, x_5_12, x_5_13, x_5_14, x_5_15, x_6_1, x_6_2, x_6_3, x_6_4, x_6_5,
x_6_6, x_6_7, x_6_8, x_6_9, x_6_10, x_6_11, x_6_12, x_6_13, x_6_14, x_6_15, x_7_1,
x_7_2, x_7_3, x_7_4, x_7_5, x_7_6, x_7_7, x_7_8, x_7_9, x_7_10, x_7_11, x_7_12, x_7_13,
x_7_14, x_7_15, x_8_1, x_8_2, x_8_3, x_8_4, x_8_5, x_8_6, x_8_7, x_8_8, x_8_9, x_8_10,
x_8_11, x_8_12, x_8_13, x_8_14, x_8_15, x_9_1, x_9_2, x_9_3, x_9_4, x_9_5, x_9_6, x_9_7,
x_9_8, x_9_9, x_9_10, x_9_11, x_9_12, x_9_13, x_9_14, x_9_15, y_1, y_2, y_3, y_4, y_5,
y_6, y_7, y_8, y_9;
```

A.3 Lpsolve output for Arizona Data

All non-zero variables are shown below.

The value of the objective function is the number of districts voting red.

The value of y_i is 1 if district i voted red.

x_{i_j} is 1 if precinct j has been assigned to district i.

d_i indicates the size of district i.

b_i indicates the number of precincts voting red in district i.

```
Value of objective function: 6.00000000

Actual values of the variables:
y_1                             1
y_2                             1
y_4                             1
y_5                             1
y_6                             1
y_7                             1
x_1_1                           1
x_8_2                           1
x_7_3                           1
x_3_4                           1
x_7_5                           1
x_8_6                           1
x_3_7                           1
x_6_8                           1
x_5_9                           1
x_9_10                          1
x_2_11                          1
x_1_12                          1
x_4_13                          1
x_9_14                          1
x_2_15                          1
d_1                             2
d_2                             2
d_3                             2
d_4                             1
d_5                             1
d_6                             1
d_7                             2
d_8                             2
d_9                             2
b_1                             2
b_2                             2
b_4                             1
b_5                             1
b_6                             1
b_7                             2
b_8                             1


real    8m16.955s
user    6m38.991s
sys     0m2.026s
```

Figure 7: LPSolve output for Arizona Data

A.4 Lpsolve output for Variation One

All non-zero variables are shown below.

The value of the objective function is the number of districts voting red.

The value of y_i is 1 if district i voted red.

x_{i_j} is 1 if precinct j has been assigned to district i.

d_i indicates the size of district i.

b_i indicates the number of precincts voting red in district i.

```
Value of objective function: 3.00000000

Actual values of the variables:
y_1                            1
y_2                            1
y_3                            1
x_2_1                          1
x_1_2                          1
x_2_3                          1
x_4_4                          1
x_2_5                          1
x_4_6                          1
x_1_7                          1
x_4_8                          1
x_3_9                          1
x_2_10                         1
x_4_11                         1
x_3_12                         1
x_3_13                         1
x_3_14                         1
x_4_15                         1
x_2_16                         1
x_1_17                         1
x_2_18                         1
x_1_19                         1
x_1_20                         1
x_4_21                         1
x_1_22                         1
x_3_23                         1
d_1                            6
d_2                            6
d_3                            5
d_4                            6
b_1                            4
b_2                            4
b_3                            3

real    0m9.275s
user    0m9.155s
sys     0m0.017s
```

Figure 8: LPSolve output for Variation one

A.5 Lpsolve output for Variation Three

All non-zero variables are shown below.

The value of the objective function is the number of districts voting red.

The value of y_i is 1 if district i voted red.

x_{i_j} is 1 if precinct j has been assigned to district i.

d_i indicates the size of district i.

b_i indicates the number of precincts voting red in district i.

```
Value of objective function: 8.00000000

Actual values of the variables:
y_2                             1
y_3                             1
y_4                             1
y_5                             1
y_6                             1
y_7                             1
y_8                             1
y_9                             1
x_6_1                           1
x_8_2                           1
x_7_3                           1
x_1_4                           1
x_5_5                           1
x_6_6                           1
x_5_7                           1
x_5_8                           1
x_6_9                           1
x_1_10                          1
x_4_11                          1
x_3_12                          1
x_2_13                          1
x_1_14                          1
x_9_15                          1
d_1                             3
d_2                             1
d_3                             1
d_4                             1
d_5                             3
d_6                             3
d_7                             1
d_8                             1
d_9                             1
b_2                             1
b_3                             1
b_4                             1
b_5                             2
b_6                             2
b_7                             1
b_8                             1
b_9                             1


real    0m4.003s
user    0m3.877s
sys     0m0.018s
```

Figure 9: LPSolve output for Variation Three

A.6 Graph Representation of Arizona Data

Vertices refer to the counties/precincts of Arizona. The number within the vertices only act as a labeling system.

Edges indicate whether two counties/precincts are adjacent to each other.

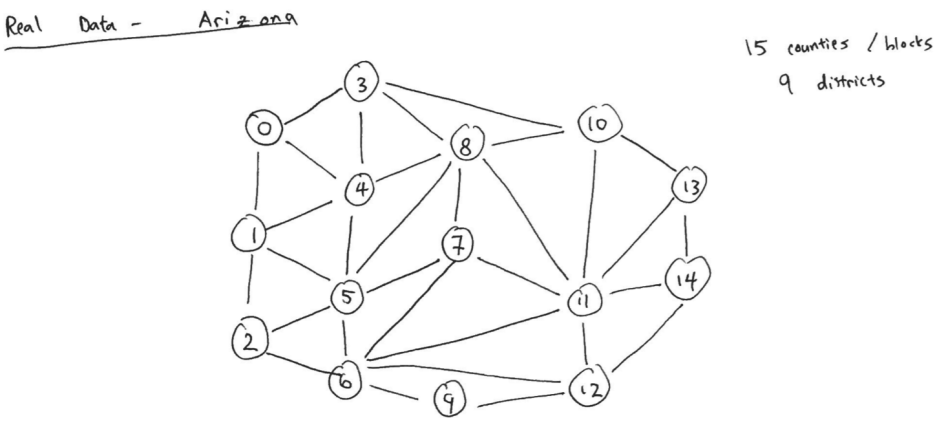

Figure 10: Graph Representation of Arizona Data