Mathematical Model of Political Districting: Optimization of Unfair Vote Distribution

Brian Kang

March 12, 2020

Abstract

This paper serves to use discrete mathematics to focus on the mechanistic modeling of unfair political redistricting, known as gerrymandering. As a large branch of literature about the legislative process, gerrymandering is widely studied for its controversy: favoring a specific individual or party over others by allocating votes into biased political boundaries. Most studies aim to develop methodologies for fair and ethical redistricting. However, in this paper, with the use of linear programming and the simplex algorithm, the goal is to maximize unfairness within a state to illustrate that gerrymandering is still feasible within legal bounds. The linear program model will be tested and discussed in context of the 2018 U.S. Senate vote results in the state of Arizona. Strengths and limitations of using linear programming over other widely used approaches such as network flow and heuristics (K-Means Clustering, Multi-kernel growth, Voronoi Approach, etc.) are discussed as well.

1 Problem and Simplifications

Gerrymandering, or unfair political redistricting, is the manipulation of the boundaries of electoral districts in order to favor one party over all other parties. With this "redrawing" of district lines, there can be outcomes where the majority population may vote for one party, but the majority district votes for another. This has been a political topic of study throughout the U.S. history[1] and an occurrence that has happened several times in U.S. elections, notably the presidential elections in 1876, 1888, 1992, 1996, 2000, and 2016 produced an Electoral College winner who did not receive the most votes in the general election.

According to National Conference of State Legislatures[2] and various studies[3], to draw the boundaries of political districts, there are several criteria that must be satisfied:

(Note: For clarification, each state has districts and each district has precincts within. This will be the vocabulary to be used throughout the entire paper.)

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.

There are other "emerging" and "future" criteria rules that have been considered or will be considered to be enacted, versus the five "traditional districting principles" listed above that are used in most or all states. In this project, We will simplify this modeling problem by

[1] Royce Crocker. *Congressional Redistricting: An Overview.* Nov. 2012. URL: `https://fas.org/sgp/crs/misc/R42831.pdf` (visited on 03/02/2020).

[2] *Redistricting Criteria.* Apr. 2019. URL: `https://www.ncsl.org/research/redistricting/redistricting-criteria.aspx` (visited on 03/02/2020).

[3] *Where are the lines drawn?* URL: `http://redistricting.lls.edu/where-state.php` (visited on 03/03/2020).

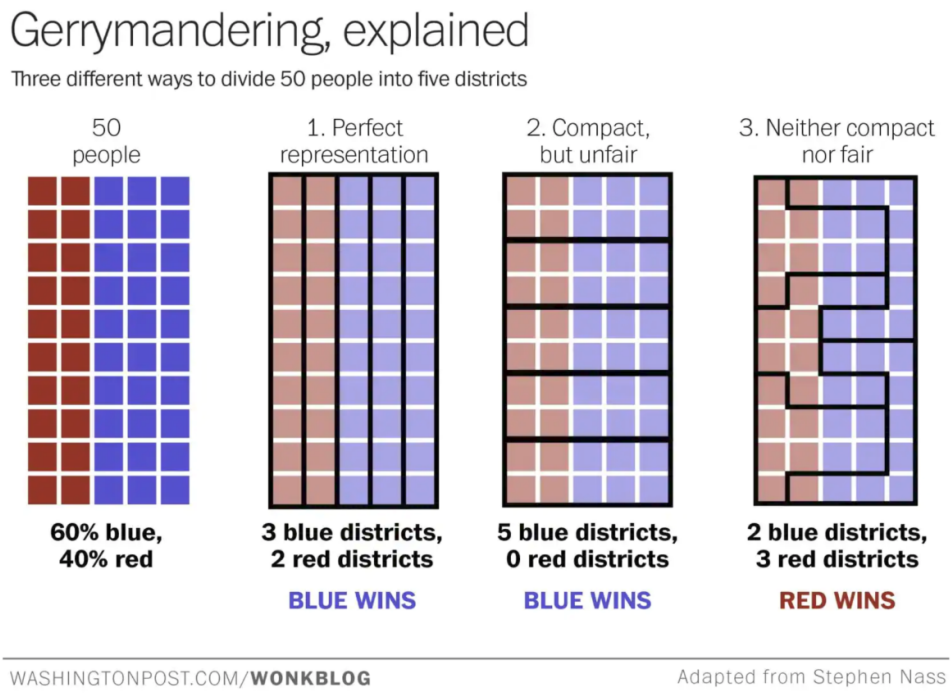

Figure 1: Illustration of Gerrymandering

assuming the "emerging" and "future" criteria as negligible. We will also assume that special cases of state-by-state as negligible as they are very hard to model using one generalized model.

When it comes down to past literature about political districting and gerrymandering, most researchers share a common objective: develop a method to minimize unfairness and allow each citizen's voice be heard. Then they usually tackle this problem in one of two methods. First is the majority used approach, Linear Programming supported with Graph Theory, or network flow approach. The very first paper researching applications of Linear Programming to gerrymandering was in Hess' 1965 paper.[4] 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. The second approach are Heuristic approaches, allowed with the development of

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

powerful and efficient modern computational power. Known as unsupervised machine learning in the data science field, common algorithms used in the gerrymandering context are K-Nearest Neighbors[5] and Principal Components Analysis.

This project, through a contradictory yet realistic objective, will aim to provide extra perspective into how the current electoral system can be easily gerrymandered in favor for a select party, all while statisfying the "traditional districting principles" required for political redistricting. We will assume that all Population Equality is satisfied across all states' districts. Since this criterion requires real world data on the population sizes, which we are not focusing on, and since this can easily be met by multiplying a weight constant to district variables, we assume this principle is already met. Also, as noted above, some additional redistricting rules will be omitted because there are limitations to satisfy special case scenarios and come up with original model for unfairness maximization, unlike Hess' approach to ensure fairness by minimizing population variance between different districts. Additionally, although this is not the main point of this paper, some real world data will be applied to clearly illustrate how our model can potentially influence the already implemented system by our government.

2 Mathematical Model

2.1 Derivation

For this project, just like the majority of whom study political districting, we use linear programming to tackle the topic of maximizing the sum of number of districts that vote for a party in a state so that the party wins in a two-party competition. We only consider a bi-partisan election because this is the prevalent behavior in today's elections. Often times, third parties rarely get the chance to win a state. We have districts encompassing precincts that either vote for Republican (red) or Democratic (blue). Then, the districts at the end

[5]Olivia Guest, Frank J. Kanayet, and Bradley C. Love. "Gerrymandering and computational redistricting". In: *Journal of Computational Social Science* 13 (2019), pp. 119–131. DOI: https://doi.org/10.1007/s42001-019-00053-9.

are also either Republican (red) or Democratic (blue) depending on which party has the majority number of precincts in a district. Without lose of generality, we want to make red win. It is trivial to make blue win if desired.

Now we construct the district boundaries, precinct allocations, and overall model to maximizing the unfairness in the election to make a single party win. Recall integrity, contiguity, absence of enclaves, and compactness principles that has to be satisfied. Population equality is assumed. In the model, we have n precincts to be assigned into k districts, implying $n \geq k$ for all states in interest. Each district can have a different number of precincts, but the difference cannot be more than two precincts in number, ensuring that each district cannot be abnormally small or large. We define the model's four variables: y_i denotes the party (red or blue) that the district i, as a whole, voted for; x_{ij} denotes 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 $i, j \in \mathbb{N}$.

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

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

$d_i =$ 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 $i, \forall i \in \{1, ..., k\}$

The above variables are needed to meet integrity. We introduce two more parameters to satisfy absence of enclaves: c_j denotes the party (red or blue) that the precinct j votes for, a binary variable with values randomly assigned. M is a real number with some large value to attain the inequality constraints needed for the model.

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

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

With all the variables define, we now introduce the linear program model's objective function subject to the constraints:

$$\max \sum_{i=1}^{k} y_i \quad \text{subject to,}$$

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

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

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

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

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

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

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

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

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

Since we want to draw district boundaries to favor the Republican party (red), the objective function 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 and precincts in the model. Constraint (2) ensures that all districts are assigned with at least one precinct.

Though the 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 the 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, constructing the fact that the difference across districts cannot be more than two precincts in number.

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 defined d_i as the number of precincts in each district, and constraint (5) helps record the value.

we have a list of binary number c_j, indicating if the precinct votes for red or not. We want to know how many precincts voted 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 district.

Since we want to maximize the the sum of districts who vote for red, y_i, in the 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. These constraints also satisfy the Absence of Enclaves and Compactness principles because, geometrically speaking, they avoid the "corners" of "squares" if we think of each precinct to have a square geometry. By avoiding the "corners"

the shapes get more compact to each other precincts within one district to optimize the vote outcome, which leads to Absence of Enclaves because if every district is compact, then no district can fully surround another. Returning to the mathematical explanation, 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 and third and minority parties may participate in political elections; however, we assumed that we are considering a bi-partisan election for our model.

Because we want integers in the model result (we do not want a decimal y_i), we modify the condition $y_i = 1$ to 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 define a more accurate range for y_i, we subtract $\frac{1}{2}$ on the right-hand side in both constraints (7) and (8). This will not affect the results since $(2b_i - d_i)$ is guaranteed to be an integer and y_i is binary.

If $y_i = 0$, we know $2b_i - d_i \leq \frac{1}{2}$ 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 $2b_i - d_i \geq 1$,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 than some negative number and $y_i - 1$ less than or equal to some negative number ranging from -1 to 0 exclusively because M>> $(2b_i - d_i) - \frac{1}{2}$. Thus, $y_i = 0$. If $(2b_i - d_i) - \frac{1}{2} > 0$, we know $0 < y_i \leq 1$ from the same constraints. Thus, $y_i = 1$. We do not worry about the equality in the 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 $y_i = 0$. Also, absence of enclaves and compactness is encouraged.

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

2.2 Real Data

Real data from Arizona's 2018 U.S. Senate voting history gathered from American political opinion company, Politico, is used as the basis of implementing our Linear Programming model[6]. Arizona specifically has been selected because county and district numbers are both moderate (15 counties (precincts), 9 districts) and visually no district has absurd shapes, which gerrymandering often takes advantage of. Although gerrymandering is the objective, we are aiming to maximize unfairness with respect to all the principles that should encourage fairness. The feasibility of the model and the solution interpretation will be in context with this Arizona Senate coverage data. The graph in terms of vertices and edges can be observed at the Appendix (A.4).

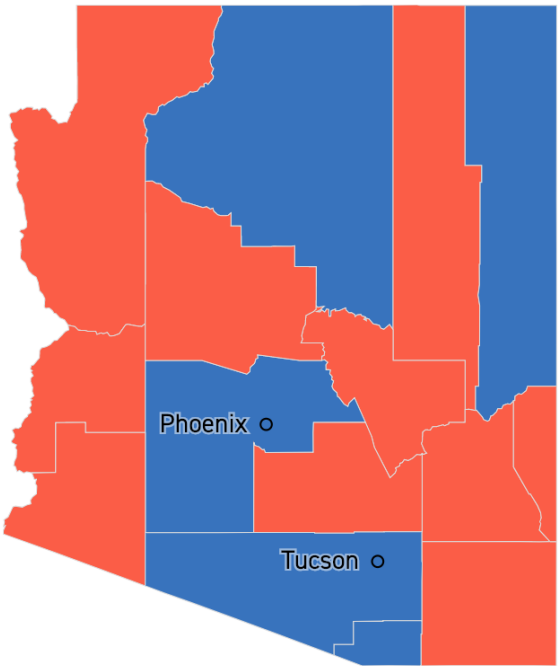

Figure 2: 2018 U.S. Senate Voting: Arizona

[6]Politico. *Arizona Election Results 2018: Live Midterm Map by County & Analysis.* URL: `www.politico.com/election-results/2018/arizona/.` (visited on 03/05/2020).

3 Solution

Java and Python was used to generate the Linear Program and output it into text format, compatible to be solved by a software (LPsolve) that uses the Simplex Algorithm to efficiently solve linear programs and return all feasible and optimal solutions if the program is terminable.

3.1 Input

To define the model, appropriate variables are created and the necessary values of some variable are assigned: n, total number of precincts; k, number of districts; and M, a big real value constant. Binary variable c_j, for each precinct j where $c_j = 1$ if the precinct votes for red and $c_j = 0$ otherwise, was randomly generated.

The final text format of the generalized model to be solved looks like:

```
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;
```

The specific model input text file generated using the Arizona data can be found at Appendix A.2.

3.2 Output

We test the model using the fact that Arizona has 15 precincts, 9 districts, only red and blue districts, i.e., $n = 15$, $k = 9$, $M = 1000000$, $c = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1]$.

The software LPsolve is used to solve the learn program generated. The program returns an optimal solution. The output can be found at Appendix A.3. Only the non-zero variables are printed, all other variables equal zero.

4 Result and Discussion

4.1 Interpretation

From the result using the Arizona data (Appendix A.3), the value of the objective function shows us that the optimal solution for the model has 6 districts out of $k = 9$ total districts voting for red, hence giving victory to the republicans for Arizona.

From the values of y_i, observe that red wins in district 1, 2, 4, 5, 6 and 7. The value of x_{ij} indicates which precincts j are assigned to which district i. 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

These results fit the value of $d_i, i \in \{1, \cdots, 9\}$. All 15 precincts are allocated to 9 districts. With each precinct voting outcome randomly generated (0 or 1/blue or red), the linear program optimizes the allocation of red and blue precincts to each district. We see blue precincts were "dumped" into district 3 and 9. This takes away 4 precincts from 15, only one blue is left, and 7 districts are left to be filled. From here, district 8 is used to "use up" the leftover blue precinct, causing one tied district. With the leftover 6 districts, red vote allocation is easily done by assigning one red vote for district 4,5,6 and again "dumping"

the leftover red precincts into district 1,2,7. Although this is not the best representation of gerrymandering, since, by chance, we started with a red majority vote, but the results are very reasonable and we can see how the optimization of unfairness is being in play.

Gerrymandering ultimately finds a away to use political redistricting within the bounds of the law by drawing weird looking boundaries to allocate precincts into these modified boundaries. And depending on state regulations, the restricting principles can be interpreted loosely. An example is Maryland's district 3, "favoring Democrats more strongly than 99.79 percent of the algorithm's maps — a result extremely unlikely to occur in the absence of an intentional gerrymander."[7]

Figure 4: An Example of Extreme Gerrymandering

4.2 Binding Constraints

In the case 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 and max difference of 2 across off districts, the possible values for the size of the district is 1 or 2. From the result (Appendix A.3), from the values of $d_i, i \in \{1, \cdots, 9\}$, the number of precincts in each district, note that when $i = 1$, 2, 7 and

[7]Erica Klarreich. *How to Quantify (and Fight) Gerrymandering.* URL: `https://www.quantamagazine.org/the-mathematics-behind-gerrymandering-20170404/` (visited on 03/07/2020).

8, constraint (3) from the model is binding. When $i = 3$, 4, 5, 6 and 9, constraint (4) from the model is binding.

5 Improvements

5.1 Contiguity

Our model fails to satisfy one of the five "traditional districting principles," contiguity. Contiguity is the requirement for all precincts in a district to be directly adjacent in **cardinal direction** to any one precinct 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. Also, if precincts can be connected diagonally, it is possible for two districts to cross each other while not violating compactness and absence of enclaves.

Observe district 1. Precincts 1 and 12 are assigned; however, per the graph of Arizona in Appendix A.4, it is clear that precinct 1 and 12 are not adjacent to each other.

Many other past studies show that there are methods to satisfy the contiguity rule. 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.[8] However, his solution was not able to serve as the foundation of this project because of my lack of understanding of planar graphs and primal/dual graphs in a network, not a linear program.

Another method to satisfy the contiguity contraint is illustrated in Nemoto's paper that used a network flow approach. He found that the classic flow balance constraint guarantees contiguity.[9] However, the issue with using Nemoto's approach was that network flow is used on direct graphs, where ours wasn't. Refer to the graph of the Arizona data in Appendix

[8] 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.

[9] 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.

A.4. Furthermore, the network flow balance constraint depends on having weighted edges, which we did not construct.

One attempt to formulate the contiguity constraint is basing off the entire model on the use of 2-D arrays. Given the planar graph of all districts and their locations in the large matrix, theoretically it is reasonable to check if for all precincts, there is at least one precinct located within a distance of 1, i.e., the precincts are adjacent. To do this, code will search all precincts' cardinal directions for an existing precincts. If all precincts are adjacent to another, this implies that the districts are contiguous. However, the issue is that not only this approach is cost heavy and low performance, but also we must assume that we already know the entire planar graph and shape of districts, which contradicts our object of being able to construct the boundaries of districts, which we don't have because we don't have the contiguity constraint.

Another attempt was to construct a constraint that forces precincts to be located within the same districts by eliminating sub-cycles, thus proving adjacency. However, the issue with this approach is that each and all districts can be sub-cycles and so contradicting again the fact that we do not know the shape and location of districts within the 2-D array. Hence, these factors encouraged to formulate the linear program model using simple vertices and not being able to construct the contiguity constraint.

5.2 Feasibility in Application

The linear program model not only does not satisfy contiguity, but also there are many factors that make this model not so feasible in real applications because the ideal solution should satisfy realistic scenarios, which is not the core objective of this project.

One factor is the population equality principle. We assume this is true for all legislative entities, so we did not construct this constraint. If the data for this is always provided this will allow our model to be closer to the ideal and realistic model.

Another factor, which has been discussed, is related to the reason Arizona was specifically

chosen for the purpose of this model. Arizona requires that that the districts are "compact, follow political boundaries, and preserve communities of interest."[10] The latter rule is one of the "emerging criteria" that is not practiced by all states. Satisfying all special cases of state legislative rules will be very difficult with a generalized linear program.

6 Conclusion

From this project where the objective is to maximize unfairness within a state in favor of one party, unlike most Gerrymandering research papers that aim to create fair and legal political districts, we conclude that gerrymandering is a complicated problem, difficult to model and solve. Through linear programming, the model was able to capture some behavior of political districting; however, without a deeper understanding of graph theory and heuristic approaches, it seems out of reach to construct a realistic algorithm.

With the use of the simplex algorithm, the model was able to successfully generate districts where red wins under reasonable initial conditions, allowing us to achieve our desired results. Creating this mechanistic model that mathematically represents real world standards is a different perspective only obtained through modelling, and it opens an opportunity to immerse in understanding the operational characteristics and consequences of real world features. Because unfair political districting will skew voting outcomes, the results will directly affect human lives, so this study has its implicit significance of its kinds. With more fine-tuning I hope this project serves as starting point for translating more advanced behaviors in numeric and ethical terms.

[10]Loyola Law School. *Where the Lines Are Drawn - Congressional Districts.* URL: `redistricting.lls.edu/where-tablefed.php.` (visited on 03/03/2020).

References

Crocker, Royce. *Congressional Redistricting: An Overview*. Nov. 2012. URL: https://fas.org/sgp/crs/misc/R42831.pdf (visited on 03/02/2020).

Guest, Olivia, Frank J. Kanayet, and Bradley C. Love. "Gerrymandering and computational redistricting". In: *Journal of Computational Social Science* 13 (2019), pp. 119–131. DOI: https://doi.org/10.1007/s42001-019-00053-9.

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

Klarreich, Erica. *How to Quantify (and Fight) Gerrymandering*. URL: https://www.quantamagazine.org/the-mathematics-behind-gerrymandering-20170404/ (visited on 03/07/2020).

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 03/05/2020).

Redistricting Criteria. Apr. 2019. URL: https://www.ncsl.org/research/redistricting/redistricting-criteria.aspx (visited on 03/02/2020).

School, Loyola Law. *Where the Lines Are Drawn - Congressional Districts*. URL: redistricting.lls.edu/where-tablefed.php. (visited on 03/03/2020).

Where are the lines drawn? URL: http://redistricting.lls.edu/where-state.php (visited on 03/03/2020).

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 input text file for LPsolve.

```
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:
```

```
46              lp += "x_{}_{} + ".format(i,j)
47      lp += "x_{}_{} <= {}; \n".format(i,n,n/k+1)
48
49  # This block generates the fourth constraint (lower bound of the number of
        precincts in each district).
50  for i in range(1,k+1):
51      for j in range(1,n+1):
52          if j != n:
53              lp += "x_{}_{} + ".format(i,j)
54      lp += "x_{}_{} >= {}; \n".format(i,n,n/k-1)
55
56  # This block generates a variable 'd_i' for the number of precincts in each
        district 'i'.
57  for i in range(1, k+1):
58      lp += "d_{} = ".format(i)
59      for j in range(1, n+1):
60          if j != n:
61              lp += "x_{}_{} + ".format(i,j)
62      lp+= "x_{}_{}; \n".format(i,n)
63
64  # This block generates a variable 'b_i' for the number of precincts in each
        district 'i' that is red.
65  for i in range(1,k+1):
66      lp += "b_{} = ".format(i)
67      for j in range(1,n+1):
68          if j != n:
69              lp += "{} * x_{}_{} + ".format(c[j-1],i,j)
70      lp += "{} * x_{}_{}; \n".format(c[n-1],i,n)
71
72
73  # This block generates the constraint to ensure that y_i is 1/0 if a district
        is majority red (more than half) or not.
74  for i in range(1, k+1):
75      lp += "{} * y_{} >= 2 * b_{} - d_{} - 0.5; \n".format(M,i,i,i)
76
77  for i in range(1, k+1):
78      lp += "{} * y_{} - {} < 2 * b_{} - d_{} - 0.5; \n".format(M,i,M,i,i)
79
80
81  # This block generates the binary constraint (All variables are binary).
82  lp += "bin "
83  for i in range(1,k+1):
84      for j in range(1,n+1):
85          if i != k or j != n:
86              lp += "x_{}_{}, ".format(i,j)
87  lp += "x_{}_{}, ".format(k,n)
88
89  for i in range(1, k+1):
90      if i != k:
91          lp += "y_{}, ".format(i)
92  lp += "y_{};".format(k)
93
94
95  text_file.write(lp)
96  text_file.close()
```

A.2 Linear Program Input File of Arizona Data for LPsolve software

```
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 Linear Program Output of Arizona Data from LPsolve

All non-zero variables are printed below.

The objective function displays the number of districts that voted red.

The value 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. (Hence, if $b_i = 1$, no precincts voted for 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
```

A.4 Graph Representation of Arizona Data

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

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