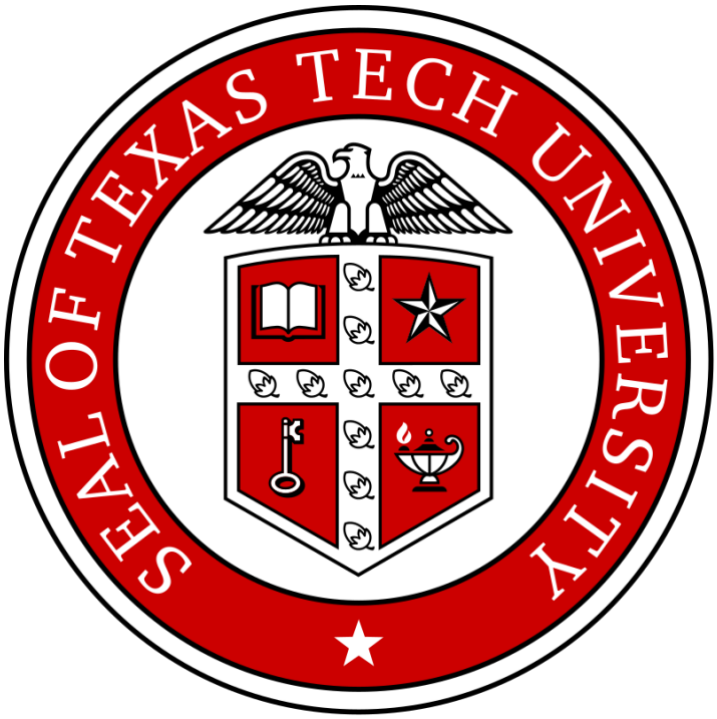
Final Report: Redistricting of Oregon State

Team Oregon Grape
Dedeaux, Fasheru, Ventura



IE 5318 Operations Research

Dr. Hamidreza Validi

12 December 2023

Contents

Executive Summary

The final report addresses redistricting Oregon state district boundaries using linear programming. The data set used is based on the 2020 census results. The paper first discusses redistricting, apportionments, and census results. This information is used to develop a linear program -integer program- that can be used to draw the boundary lines for the Oregon districts. The redistricting plan capitalizes on the following datasets: Population predictions and actuals in 2010 and 2020 for Oregon, Number of seats available of the 435 in Congressional, House, and Senate, Oregon - 2010: 5 seats and 2020: 6 seats. The constraints include the federal and state criteria for redistricting, along with the limitations associated with U.S government elections for the seats. Henceforth, the constraints are represented by the following: contiguous districts, districts of (mostly) equal population, herein defined as population deviation, districts utilizing existing political and geographic boundaries, districts connected by transportation links, districts must contain a minimum of two whole state legislative districts, districts shall include racial and language minority protections, districts should not reduce or dilute minority voting strength, districts should not divide communities of common interest, districts should not intentionally favor party, incumbent, or person. The variables are captured by the state boundary lines, counties, and populations. Using the constraints and the variables, the objective function is used to determine the minimization of the population deviation between counties. With the constraints and limitations in mind, a linear program was developed. However, the result concludes that through linear programming, Oregon redistricting is infeasible. This was due to a violation of the constraints.

The dataset and tool have been uploaded to Github. The link is below:

https://github.com/Jdedeaux/Districts-Counties/tree/main

1 Introduction

1.1 Problem Statement

With the release of the United States (U.S.) 2020 Census, Oregon state is redrawing its district lines. Redistricting is the act of redrawing legislative and congressional district lines. The goal is to promote districts of roughly equal population. In the state of Oregon, the legislature primarily creates and submits the redistricting plan which must be signed into law by the state Governor. Prior to the 2020 Census, Oregon had five congressional districts. We will intake the 2020 census data with the purpose to redistrict Oregon state. Within this report, our methodology and results will be compared to that of the official 2020 Oregon redistricting.

1.2 Congressional Redistricting Modeling Requirements and Criteria

A congressional district, much like a legislative district, attempts to equally partition the state's population while maintaining all redistricting criteria. Each district corresponds to a U.S. House representative, known colloquially as a congressman or congresswoman. The U.S. has maintained a consistent number of congressional seats at 435 seats, so all states alter their congressional district assignments to uphold the 435 seats. Therefore, the congressional districts have fluctuated, and the average population per seat has increased over time as the census is completed.

Our model is governed by various requirements set by the law. These laws were set by the state (Oregon redistricting criteria) and by the country (Federal redistricting criteria). The following two sections will list and discuss the requirements in detail.

1.2.1 Federal Redistricting Criteria

The federal redistricting criteria are initially based on the 14th Amendment to the U.S. Constitution, with interpretation by the Supreme Court in the 1962 for the "one person, one vote" doctrine. The criteria were further enforced and regulated by the Voting Rights Act of 1965. Regarding population equality, districts must be of equal population to ensure that the value of every person's vote is substantially equal. Regarding racial and language minority protections, district plans may not intentionally dilute the voting strength of members of a racial or ethnic minority group or contain districts drawn primarily on the basis of race or ethnicity without a compelling justification (Ballotpedia, 2011;2023).

Required congressional redistricting criteria:

- Population equality
- Racial and language minority protections

Prohibited congressional redistricting criteria:

- Reduce minority voting strength
- Intentionally favor party, incumbent, or person

1.2.2 Oregon Redistricting Criteria

According to the Oregon state redistricting plan, congressional district mapping is legislative and abides by the following criteria for the mapping (*Bills and Laws OrConst,* 2022; *Redistricting Criteria,* 2023; *Redistricting in Oregon*, 2023). The criteria are listed and defined in ORS 188.010 "Criteria for appointment of state into congressional and legislative districts" (ORS, 2021). Requirements marked by '*' are also within the federal redistricting criteria. Every

10 years, states redraw the boundaries of their congressional districts to reflect new population counts from the census. Oregon gained a congressional seat after the 2020 census. Oregon's map was passed by the Democratic-controlled legislature and signed by Democratic Gov. Kate Brown in September 2021, making it the first state to complete congressional redistricting.

The new map maintains one safe seat for Republicans. The other five districts lean Democratic, but the 5th District will be more competitive. The Oregon redistricting plan was completed on September 27, 2021. The Legislative Assembly completed the adoption of new congressional and legislative redistricting plans for Oregon with the passage of Senate Bill 881 A and Senate Bill 882. These plans were signed into law by Governor Brown and subsequently upheld by the courts. Elections using these new plans will take place in 2022 with those elected taking their seats in January of 2023.

Required Oregon redistricting criteria:

- Districts must be contiguous
- Be of equal population*
- Districts must utilize existing political and geographic boundaries
- Districts must be connected by transportation links
- Two state legislative districts within a single state congressional district

Prohibited Oregon redistricting criteria:

- Reduce or dilute minority voting strength*
- Districts should not divide communities of common interest
- Intentionally favor party, incumbent, or person

All requirements must be met as practically as possible. Oregon defines contiguous as a district where you can travel from any point in the district to any other point in the district without crossing the district's boundary. Politically within Oregon, boundaries are primarily between cities of the Willamette Valley and the remainder of the state. Lastly, a minimum of two legislative districts must be wholly within the congressional district for fair representation.

2 Programming Model

2.1 Modeling Process and Goal

With the use of Gurobi, we approached the modeling process as following. The first step is to define our model. With most of the model developed during our project proposal, we further defined the affluent variables of the model. Step [2] was to reduce the data to intake into our python script. Within this step, the data must be validated

. Likewise, during the reduction of data, we compiled the key metrics and expected outcomes. We verified that the dataset contains the essential data to meet these metrics and outcomes. Following this step, we must preprocess the data, as reflected by Step [3]. Preprocessing the data ensures the proper formatting will be ingested by the python script. In addition, we can apply weights to different factors within the dataset in the scenario where some criteria are a higher priority than others, especially in the cases of conflict between criteria. With the data reduced and preprocessed, we now use the data for our model. Step [4] involves the development of the code to create the districting plan(s) and map(s). This step involved

importing various libraries and tools to generate our desired outputs. This step also reflected our model to verify the minimum requirements for redistricting were met. Lastly, in a continuous learning process, we must alter the code to produce a sensible districting plan that meets the requirements of redistricting.

1. Model Development
    a. Utilize project proposal model
    b. Update and define all variables
2. Data Reduction
    a. Define key metrics
    b. Define expected outcomes
    c. Validate dataset
3. Data Preprocessing
    a. Format data
    b. Add weights to various criteria
4. Data Tooling Development
    a. Meet minimum requirements
    b. Code trials

2.2 Model: Mathematical Explanation

The operation research model approach utilizes the population deviation to understand the upper and lower bounds of the population estimations of each district. The populations of each district were analyzed via preprocessed data sets which align with the "state.population" text files. In the proposal, a 1% deviation was selected to analyze the model, although the final study utilized a 3% deviation to understand the effects of a greater deviation as a function of the total number of districts.

There are two optimization models in the overall Gurobi model. The first model is the population deviation model where the objective function evaluates the minimization of the delta between the population bounds. The output will influence the districting lines; moreover, the counties that will be assigned to the district that possess an additive population within the population bounds. The second models is the cut edges minimization model that address the node connections, county sizes, and district lines that geographical map the most optimal counties together in each district. The population model is defined first so that the population bounds can be defined. Further, the cut edges model will update the population optimization model to allow neighboring districts within the population bounds to create each district. It is imperative that the models run in a sequenced fashion, so that the model will build upon the driving factor which is the county population.

Objective Function:

<u>For the population deviation model:</u>

$$\min z - y$$

For the cut edges minimization model:

$$min \sum_{u=1}^{U} \sum_{v=1}^{V} y_{uv}$$, u = edge start, v = edge end

2.2.1 Variables

The variables need to analysis the model align with the counties, districts, and data sets that are being evaluated. The variables capitalize on the node variables, for the nodes represent each county. The counties will be a function of the district as each county will reside in a district. For the population deviation model, the comparison of the county populations will be utilized to resolve the objective function, and to output the optimal value. As for the cut edges model, the cut edges will be defined to access the minimization model to identity the number of cut edges needed to satisfy the redistricting of the state.

For the population model:

$$Node\ identifier = x = x_{ij}\ where\ i = county, j = district, as\ a\ function\ of\ county\ population\ and\ number\ of\ congressional\ district$$

$$population\ of\ least\ populated\ district = y = y_j\ where\ \ j = district$$

$$population\ of\ most\ populated\ district = z = z_j\ where\ \ j = district$$

For the cut edges minimization model:

$$Node\ identifier = x = x_{ij}\ where\ i = county, j = district, as\ a\ function\ of\ county\ population\ and\ number\ of\ congressional\ district$$

$$cut\ edge = y = y_{uv}\ where\ \ u = edge, j = cut$$

2.2.2 Model Constraints: Population, State, and County

The constraints of the population model are the means of calculating the populations. Each county only has one distinct population, and this population will populate a factor to relay on the total population of the district and the total population of the state. The parameters of the state and county federal regulations bound the counties to address the intentional planning of racial and equity segregation amongst districts. To ensure that the model is within the state and federal regulations, the neighboring counties must not exceed and be beyond the bounds of the population deviation and boundary. This prevents counties of "higher value" based on the social and monetary status to not be paired with a county of general likeness. The model addresses this phenomenon by the residuals or the population deviation. The population test deviation is 3% to study how the higher percentage, can affect the district grouping based on the desired number of districts.

For the population model:

Constraints:

$$Node\ identifier = x = x_{ij}\ where\ i = county, j = district, as\ a\ function\ of\ county\ population\ and\ number\ of\ congressional\ district$$

Population:

$$population = population_i, where\ i\ is\ the\ county$$

$$total\ population = \sum_{i=1}^{I} population_i$$

For the cut edges minimization model:

Constraints:

$$\sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij} = 1, \bigvee i$$

$$\sum_{i=1}^{I} population_i \times x_{ij}, \bigvee i \geq Lower\ Bound \bigvee j$$

$$\sum_{i=1}^{I} population_i \times x_{ij}, \bigvee i \leq Upper\ Bound \bigvee j$$

Population:

$$deviation = 3\%\ or\ 0.03$$

$$Lower\ Bound = \frac{1 - deviation}{2} \times \frac{total\ population}{k}$$

Districts:

$$k = number\ of\ desired\ districts\ based\ on\ the\ 2010\ or\ 2020\ census$$

In addition to the non-negativity clause for our model, we included the following additional constraints:

1. contiguous districts
2. districts of (mostly) equal population, herein defined as population deviation
3. districts utilizing existing political and geographic boundaries
4. districts connected by transportation links
5. districts must contain a minimum of two whole legislative districts
6. districts shall include racial and language minority protections
7. districts should not reduce or dilute minority voting strength
8. districts should not divide communities of common interest
9. districts should not intentionally favor party, incumbent, or person

For some of these constraints, they will be contained within an umbrella term; a county, also known as a legislative district. We will assume that whole counties in Oregon represent the following constraints: [3], [4], [5], [6], [7], [8], [9]. To satisfy constraint [1], we will include a mathematical constraint where the objective function value must include all counties within are adjacent to at least one other county within the value.

Additionally, to ensure that our new redistricting plan aligns with the 2020 map's population distribution, we will impose constraints for each district, ensuring that the population deviation falls within the 1.84% overall range of the 2020 map. The deviation constraint is as follows:

$$|P_i - P_{i,2020}| \leq 1.84\% \times P_{i,2020}$$

To achieve a 3% population deviation (+/- 0.5%), we will use the exact figures for the lower and upper bounds based on the 2020 data:

Lower Bound: 628,935 *Upper Bound: 648,090*

Lastly, the population of each district in our new plan will be subject to the following constraint:

$$628{,}935 \leq P_i \leq 648{,}090$$

2.3 Data Set

The data set used for the linear programming model stems from Lykhovoyd's data. As informed by Dr. Validi, this dataset had been previously preprocessed to reflect compliance with various constraints, such as protections for racial and language minorities and not dividing communities of common interest, whilst still using the 2020 Oregon census data.

The population data sets institute the 2020 census data for each counties population, while the dimacs sets institute the county lines that are shared with neighboring counties. The combination of the two sets make up the components of the json files which connect counties or nodes to a location within the state. This information is then aligned with the populations which can be used to model the districts.

Additionally, the data reflects the following file times to create the redistricting map. These files include shape files, which reflect the whole state of Oregon and its respective counties, and distances between county centers in a comma separated values file in conjustion with the distances file.

2.4 Data Reduction, Preprocessing, and Tooling Development

2.4.1 Data Reduction and Preprocessing

To run the data in a clean fashion the following had to be completed: removing file headers, removing non-integer values, converting the files to data frames, edge list, node views, or graphs from text files, and installing new Gurobi environments to analyze the redistricting data.

To ensure that the tool and the data sets were functioning properly, the team perform two analysis. The first analysis is the redistricting of Oklahoma. This is where we use our tool to output the responses for Oklahoma, and we compared it to the studies as shown on Dr. Austin Buchanan git hub. As a result, with the modified population deviation and minimum cut edge tool design as compared to and adapted from Dr. Buchanan - use of alternative data types, packages, code structure, variable definitions, similar constraints, similar objective functions, we were able to get results within one thousandth as compared to the study of Dr. Buchanan. Since we used a different data collection and data import from Dr. Buchanan, it is important to show the tool capability in respect to the redistricting output. With the understanding that Oregon, is assumed to be an infeasible model. The data sets and model functionality check with the

Oklahoma study shows that the model and the tool are an accurate depiction on the approach to redistricting the state of Oregon.

2.4.2 Tooling Development: Section 1

Step 0: Initialize and import required libraries and packages to run various cells

```
import gurobipy as gp
from gurobipy import GRB
import networkx as nx
import geopandas as gpd
import math
from gerrychain import Graph
import pandas as pd
import matplotlib.pyplot as plt
```

Figure 1: Package importing and function assignment

Step 1: Define the edges and nodes of the model

```
# Create the intial node and edges model using the edges list for Oregon
# Remove the non-integers from the text file and ensure it is a plain text file.
filepath = '/Users/jessi/Gurobi-Python Playground/OR.dimacs-mod.txt'

# reading from the data set
G = nx.read_edgelist(filepath,nodetype=int)

print(type(G))

# drawing the graph
nx.draw(G, with_labels=True)

# Current model description
print("The Oregon graph has",G.number_of_nodes(),"nodes")
print("The Oregon graph has",G.number_of_edges(),"edges")
print("Oregon graph has nodes",G.nodes )
print("Oregon graph has edges",G.edges )
```

Figure 2: Reading in the dimacs or graph file as an edge list to map the nodes

Like all python programming tools, a myriad of libraries and packages were called and imported into the file. Various functions of each of the listed libraries, from Gurobi to geopandas to matplotlib, were used to complete the assignment. The primary roles for each of the libraries in this tool are listed below:

- Gurobi: solver for the linear program. Note, Gurobi will solve the linear program using heuristic methods, as solving optimally will take significantly longer due to the immense computation
- Networkx: functions provided import and initial graphing of nodes and edges
- Geopandas: aided in importing dataframes and overall mapping of counties and district lines for the state
- Math: provide basic mathematical functions
- Gerrychain: aid in graphing of nodal elements
- Pandas: provides additional mathematical functions and representation, especially for building the frequently used dataframes
- Matplotlib: aids in visualization of the finalized redistricting maps

The first step of the tool was to create nodes and edges to eventually represent the finalized districts. As we continue through the code, the nodes will be assigned and associated with the data set mentioned previously. The solution of running this first section of the code is seen

below, where there is total of 36 identified nodes (0 thru 35). Each node corresponds to one of Oregon's counties. The assignment variables for each node are located within the uploaded data sets as seen on Github.

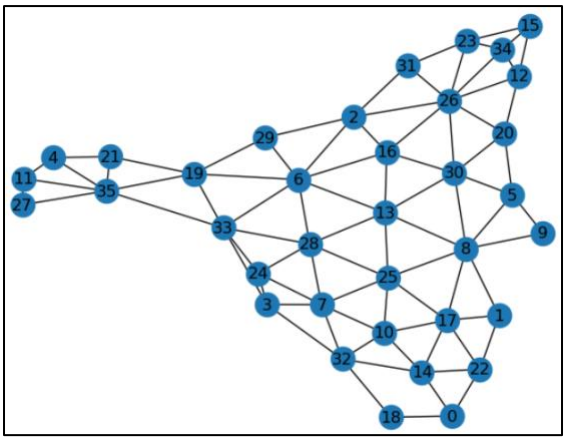


Figure 3: Initial node map of Oregon with county labels and no district affiliation

2.4.3 Tooling Development: Section 2

Step 2: List the county populations

```
# Integrate the populations by county to the model

# create population list
population = list()

# read text file into software system using files on local drive
filepath = "/Users/jessi/Gurobi-Python Playground/OR.population"
file = open( filepath,"r")

#remove the header to only read the integers
next(file)

# Create List for each county population
Listpop = []
for each in file:
    #split the list so the populations in each range
    each = each.split(' ')
    #print((each[1])
    Listpop.append(int(each[1]))

# displays the list of integers
print("Oregon's county populations = ", Listpop)

# the sums can be taken by list position
total_population = sum(Listpop)

#print the total population of OR
print("The total_population of Oregon = ", total_population)
population = Listpop #label for code clarity
n =len(population)

#print the county population as a sum districts of OR
print("Oregon's number of counties = ",n)
print("Population data is", type(population))

districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
```

Figure 4: Population Data integration to the tool, organization of the population data set

Next, each node must be assigned their corresponding populations. As the population is a key metric used for redistricting, all population values must correctly relate to their respective

counties. After running the code, we can validate that all counties were populated with their population data, in addition to verifying the sum of the county population and a count of the nodes.

```
Oregon's county populations =  [52589, 23977, 7495, 50395, 88090, 1995, 198253, 87433, 26670, 1870, 107722, 23446, 80075, 2450
2, 600372, 7391, 24738, 421401, 41072, 69413, 12186, 223259, 815428, 16668, 95184, 345920, 7233, 64929, 128610, 8160, 1451, 315
71, 27390, 382971, 26196, 111201]
The total_population of Oregon =  4237256
Oregon's number of counties =  36
Population data is <class 'list'>
```

Figure 5: Population Information for Oregon

2.4.4 Tooling Development: Section 3

Step 3: Population Deviation Model

```
# If model is feasible, use the population deviation operations research model.
# create model
m = gp.Model()
#number of districts per the 2020 census for OR
k = 6

# create variables adapted from districting starter models
x = m.addVars(n,k, vtype=GRB.BINARY) # x_ij equals one when county i is assigned to district j
y = m.addVar()                       # the population of least-populated district
z = m.addVar()                       # the population of most-populated district

# objective is to minimize absolute population deviation
m.setObjective(z-y, GRB.MINIMIZE)

# add constraints saying that each county i is assigned to one district
m.addConstrs( sum(x[i,j] for j in range(k)) == 1 for i in range(n) )

# add constraints saying that each district has population at least y
m.addConstrs(y <= sum(population[i] * x[i,j] for i in range(n)) for j in range(k) )

# add constraints saying that each district has population at most z
m.addConstrs( sum( population[i] * x[i,j] for i in range(n)) <= z for j in range(k) )

# solve IP model
m.optimize()
```

Figure 6: Operations Research Model with the Population Deviation approach to redistricting

In the following section of code, we create the initial model for Gurobi to optimize. In this model, a x_{ij} variable was added where i represents the county and j represents the district that i is assigned to. As the official Oregon redistricting of post-2020 census had 6 official districts, we used this as our starting value for k, as seen. Next, the objective of our model is to minimize the deviation between district populations, as described earlier in section 2.2 Model. Lastly, multiple constraints must be added prior to implementing the Gurobi solver. The primary constraint ensures all counties i are integers that may only be assigned to a district once. The other constraints identify the minimum and maximum population values for all districts. Additional constraints were not yet added to the model.

Without the additional constraints, the solution using Gurobi solver is seen below.

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 48 rows, 218 columns and 660 nonzeros
Model fingerprint: 0x00001389
Variable types: 2 continuous, 216 integer (216 binary)
Coefficient statistics:
  Matrix range     [1e+00, 8e+05]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 957825.00000
Presolve time: 0.00s
Presolved: 48 rows, 218 columns, 660 nonzeros
Variable types: 0 continuous, 218 integer (216 binary)

Root relaxation: objective 0.000000e+00, 52 iterations, 0.00 seconds (0.00 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.00000    0    9 957825.000    0.00000   100%     -    0s
H    0     0                     611097.00000    0.00000   100%     -    0s
H    0     0                     447504.00000    0.00000   100%     -    0s
     0     0    0.33333    0   14 447504.000    0.33333   100%     -    0s
H    0     0                     213397.00000    0.33333   100%     -    0s
H    0     0                     198044.00000 61593.0000  68.9%     -    0s
     0     0 61593.0000    0   13 198044.000 61593.0000  68.9%     -    0s
H    0     0                     196344.00000 61593.0000  68.6%     -    0s
H    0     0                     139288.00000 61593.0000  55.8%     -    0s
H    0     0                     132428.00000 61593.0000  53.5%     -    0s
     0     0 61593.0000    0   10 132428.000 61593.0000  53.5%     -    0s
     0     0 109219.000    0   13 132428.000 109219.000  17.5%     -    0s
```

Figure 7: Population Deviation Output 1

```
     0     0 109219.000    0   16 132428.000 109219.000  17.5%     -    0s
H    0     0                      132183.00000 109219.000  17.4%     -    0s
     0     0 109219.000    0   22 132183.000 109219.000  17.4%     -    0s
H    0     0                      132140.00000 109219.000  17.3%     -    0s
     0     0 109219.000    0   24 132140.000 109219.000  17.3%     -    0s
     0     0 109219.000    0   12 132140.000 109219.000  17.3%     -    0s
     0     0 109219.000    0   12 132140.000 109219.000  17.3%     -    0s
H    0     0                      132029.00000 109219.000  17.3%     -    0s
     0     0 109219.000    0   17 132029.000 109219.000  17.3%     -    0s
     0     0 109219.000    0   11 132029.000 109219.000  17.3%     -    0s
H    0     2                      131960.00000 109219.000  17.2%     -    0s
     0     2 109219.000    0   10 131960.000 109219.000  17.2%     -    0s
H   31    19                      131825.00000 131063.000  0.58%  10.1    0s
H   78    85                      131525.00000 131063.000  0.35%   7.2    0s
H  183   236                      131347.00000 131063.000  0.22%   4.7    0s
H  604   757                      131323.00000 131063.000  0.20%   3.7    0s
H  742   757                      131175.00000 131063.000  0.09%   3.7    0s
* 9935  3740              70     131116.00000 131063.000  0.04%   1.2    1s
*15801  2908              74     131080.00000 131063.000  0.01%   1.5    1s
*20150  2872              80     131069.00000 131063.000  0.00%   1.6    1s

Cutting planes:
  Cover: 2
  Flow cover: 3

Explored 20804 nodes (56687 simplex iterations) in 1.54 seconds (0.46 work
units)
Thread count was 8 (of 8 available processors)

Solution count 10: 131069 131080 131116 … 132029

Optimal solution found (tolerance 1.00e-04)
Best objective 1.310690000000e+05, best bound 1.310630000000e+05, gap 0.0046%
```

Figure 8: Population Deviation Output 2

In this model the optimal solution to the objective function is the population deviation in respect to persons. Since the population deviation in persons is over 100, then we can assume that the model for redistricting will be infeasible. The value of greater than 100 is defined through compliance with the federal and state constraints, along with the experimental comparison to the 1-person population deviation when running the Oklahoma model. With additional analysis, we found that the less districts a state has the smaller the population deviation. This is not a viable solution because federal and state laws would impede on the less congressional districts which leads to less house and senate representation.

2.4.5 Tooling Development: Section 4

Step 4: Current model status for population

```
# print the absolute population deviation
print("The absolute population deviation is",m.objval,"person(s).")
print("Oregon has districts =",k)

# Current district status per the population deviation, since the model is infeasible
districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]

# print district status
for j in range(k):
    print("Oregon: District",j,"and contains counties",districts[j])
```

Figure 9: Code of the status of population deviation in the redistricting model

In this next step, we verify the number of districts were correctly used for the Gurobi solver and find our objective value. Since our model is attempting to minimize the population deviation, we want to achieve the lowest value, indicating that all districts appropriately represent the population of Oregon.

```
The absolute population deviation is 131069.0 person(s).
Oregon has districts = 6
Oregon: District 0 and contains counties [0, 16, 7, 32, 33, 20, 30, 19, 34]
Oregon: District 1 and contains counties [22]
Oregon: District 2 and contains counties [14, 26, 3, 9, 13]
Oregon: District 3 and contains counties [17, 31, 11, 35, 12, 23]
Oregon: District 4 and contains counties [8, 2, 29, 24, 28, 25, 27, 15]
Oregon: District 5 and contains counties [18, 1, 6, 4, 21, 5, 10]
```

Figure 10: Current status of population deviation in the redistricting model

After running the code, our absolute population deviation was calculated as 131,069. Considering the total population of Oregon, this is an extremely high value and is not ideal for redistricting. We can also see the distribution of each county within the 6 districts. We especially noted the county labeled as 22, which is contained in a single district, district 1. We can assume this is Multnomah County Oregon based on the population and the location on the node map. We will discuss this district in detail later in the report.

2.4.6 Tooling Development: Section 5

```
# data from the county 2010 Census
filepath = '/Users/jessi/Gurobi-Python Playground/OR_county.json'
G = Graph.from_json(filepath)
print(G.nodes)

m = gp.Model()
#check the list of nodes and matches are align with the mapping data
print(G.nodes)
print(G.edges)

for node in G.nodes:
    # using the columns, located the population
    population = G.nodes[node]['TOTPOP']

total_population = sum(G.nodes[node]['TOTPOP'] for node in G.nodes)
print(total_population)

# Impose a 3% population deviation (+/- 0.5%) for the Operations Research model
deviation = 0.03
#number of counties
n
# number of districts
k

# Lower and Upper bound populations to ensure the populations of the districts␣
 ↪do not exceed the bounds
L = math.ceil((1-deviation/2)*total_population/k)
U = math.floor((1+deviation/2)*total_population/k)

# create variables adapted from redistricitng starter:
x = m.addVars(G.nodes, k, vtype=GRB.BINARY) # x[i,j] equals one when county i␣
 ↪is assigned to district j
y = m.addVars(G.edges, vtype=GRB.BINARY)  # y[u,v] equals one when edge {u,v}␣
 ↪is cut
```

Figure 11: Minimization of Cut Edges Model Part 1

```
# objective is to minimize cut edges
m.setObjective( gp.quicksum( y[u,v] for u,v in G.edges ), GRB.MINIMIZE )

# add constraints saying that each county i is assigned to one district
m.addConstrs( gp.quicksum(x[i,j] for j in range(k)) == 1 for i in G.nodes)

# add constraints saying that each district has population at least L and at␣
 ↪most U
m.addConstrs( gp.quicksum( G.nodes[i]['TOTPOP'] * x[i,j] for i in G.nodes) >= L␣
 ↪for j in range(k) )
m.addConstrs( gp.quicksum( G.nodes[i]['TOTPOP'] * x[i,j] for i in G.nodes) <= U␣
 ↪for j in range(k) )

# add constraints saying that edge {i,j} is cut if i is assigned to district u␣
 ↪but j is not.
m.addConstrs( x[i,v] - x[j,v] <= y[i,j] for i,j in G.edges for v in range(k))

m.update()

# solve IP model
m.optimize()

# print the optimal minization of cut edges
print("Oregon: The number of cut edges is",m.objval)
```

Figure 12: Minimization of Cut Edges Model Part 2

In step 5, we create a second model. This model uses a different set of variables and constraints for Gurobi to solve. However, it does utilize the same imported dataset and assumptions for using the value of 6 for ‘k’ districts. For this model, a 3% population deviation

was utilized to bound the population values. The objective, like the last model, is to minimize, but this model focuses on minimizing cut edges.

After running this code, we receive the output that this model is infeasible. Unlike the previous model, a population deviation constraint was not placed. According to this solution, we find that using linear programming with the required Oregon state and federal redistricting laws and our constraints produces an infeasible model. Due to model infeasibility in the tool, the tool ran for approximately 8 hours before the output read infeasible. Below are screenshots that were captured to show the runtime.

```
Solution count 0

Model is infeasible
```

Figure 13: Shows that the Oregon redistricting model is infeasible

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 3572 rows, 1830 columns and 13860 nonzeros
Model fingerprint: 0xb7d1b581
Variable types: 0 continuous, 1830 integer (1830 binary)
Coefficient statistics:
  Matrix range     [1e+00, 5e+05]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 6e+05]
Presolve time: 0.03s
```

Figure 14: Model Presolver data

```
     1767  1375   48.10591   29  807   77.00000   12.34477  84.0%   709  162s
H    1778  1311                       75.0000000   12.34477  83.5%   707  162s
H    1815  1290                       74.0000000   12.34477  83.3%   701  164s
     1837  1329   49.16924   33  589   74.00000   12.34477  83.3%   698  165s
H    1904  1268                       73.0000000   12.34477  83.1%   686  168s
     1957  1335   60.86902   39  587   73.00000   12.34477  83.1%   676  171s
     2116  1389   67.62673   49  769   73.00000   12.34477  83.1%   650  175s
     2214  1416   28.25583   19 1066   73.00000   18.39214  74.8%   646  180s
     2324  1473   45.97738   26 1069   73.00000   18.39214  74.8%   639  185s
     2402  1494   60.54922   32  998   73.00000   18.39214  74.8%   636  191s
H    2403  1424                       72.0000000   18.39214  74.5%   635  191s
     2446  1473   70.37679   36  887   72.00000   19.26548  73.2%   638  196s
     2542  1547   36.44970   22 1110   72.00000   19.26548  73.2%   634  200s
     2603  1541   41.29385   25  992   72.00000   19.26548  73.2%   629  214s
     2624  1569   42.14809   26  923   72.00000   19.26548  73.2%   627  217s
     2686  1591   51.04369   32  899   72.00000   19.26548  73.2%   625  220s
     2792  1666   67.33191   54  858   72.00000   19.26548  73.2%   623  225s
     2921  1682   27.27647   22 1183   72.00000   21.30443  70.4%   618  235s
     3005  1750   38.44172   29 1043   72.00000   21.30443  70.4%   619  242s
     3053  1804   43.74192   34  972   72.00000   21.30443  70.4%   617  246s
     3220  1899   60.37663   62  793   72.00000   21.50653  70.1%   609  253s
     3320  1939   30.38479   22 1118   72.00000   21.50653  70.1%   603  256s
     3418  1913   36.95148   28 1076   72.00000   21.50653  70.1%   599  289s
     3428  1938   37.86978   29 1154   72.00000   21.50653  70.1%   599  296s
```

Figure 15: Model Run time to complete the analysis is approximately 8 hours.

The outputs below show that the Oregon redistricting model once the populations were assigned to nodes and the nodes to districts is infeasible. This is due to Oregon's state population ability to split Multnomah County (22). Multnomah county population is outside of the

population bounds; therefore, it cannot be combined into a district. This impedes on the state and federal laws that one county cannot stand alone in a singular district.

```
Oregon: Using Lower Bound of Population = 628935 and Upper Bound of Population =
648090 and districts = 6
Oregon'population deviation is greater than 100; therefore, the model is assumed
to be infeasible. This redistricting of OR is a more complex tasking due to
counties being larger than others. The model proves the infeasibility.
```

Figure 16: Current Population and District Status

2.4.7 Tooling Development: Section 6

1.2.6 Step 6: Retrieve the districts and their populations

```
districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
district_populations = [ sum(G.nodes[i]["TOTPOP"] for i in districts[j]) for j
 ↪in range(k) ]

# print district status
for j in range(k):
```

Figure 17: Recalling the districts and applying the redistricting- infeasible

```
    print("Oregon: District",j,"has population",district_populations[j],"and
 ↪contains counties",districts[j])
```

Figure 18: Printing the redistricting information -infeasible

Moving forward, we attempt to display all the populations for each district within Oregon. However, as just described, because the solver determined the model was infeasible, it did not solve for our x_{ij} variable. Therefore, we cannot conclude further.

The error after running the code for Oregon is shown below.

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[47], line 2
      1 # Step 6:retrieve the districts and their populations
----> 2 districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
      3 district_populations = [ sum(G.nodes[i]["TOTPOP"] for i in districts[j]) for j in range(k) ]
      5 # print district status

Cell In[47], line 2, in <listcomp>(.0)
      1 # Step 6:retrieve the districts and their populations
----> 2 districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
      3 district_populations = [ sum(G.nodes[i]["TOTPOP"] for i in districts[j]) for j in range(k) ]
      5 # print district status

Cell In[47], line 2, in <listcomp>(.0)
      1 # Step 6:retrieve the districts and their populations
----> 2 districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
      3 district_populations = [ sum(G.nodes[i]["TOTPOP"] for i in districts[j]) for j in range(k) ]
      5 # print district status

File src/gurobipy/var.pxi:125, in gurobipy.Var.__getattr__()

File src/gurobipy/var.pxi:153, in gurobipy.Var.getAttr()

File src/gurobipy/attrutil.pxi:100, in gurobipy.__getattr()

AttributeError: Unable to retrieve attribute 'x'
```

Figure 19: Since the objective function cannot be calculated, the model is not able to define the number of cut edges.

2.4.8 Tooling Development: Section 7

In this step, we are readopting the initial nodes created in step 1. However, the aim is to visually identify each of the 36 nodes according to their district. This is done through color-

coding each of the districts, as seen above. After running the code, we can identify which counties belong to which district, with a total of 6 different colors to represent the different districts.

2.4.9 Tooling Development: Section 8

Step 8: State shape Mapping the counties

```
# Read Oregon county shapefile and distance files
file1 = '/Users/jessi/Gurobi-Python Playground/OR_distances.csv'
file2 = '/Users/jessi/Gurobi-Python Playground/OR_counties.shx'
file = '/Users/jessi/Gurobi-Python Playground/OR_counties.shp'
#create the data frame
df = gpd.read_file(file)

#checker
print((df))
print(type(df))


# Mapping
fig,ax = plt.subplots(figsize=(10,10))
df.plot(ax=ax, column = "geometry", cmap ='Blues')
ax.axis("off")
print("Oregon's counties are shown by varied blues in the diagram")
```

Figure 20: Calling in the shapefile and assigning colors to the counties

In this last step, we use the aforementioned shape files to map all of Oregon's counties. A dataframe is created to implement visual mapping. The result of running the code is below. As seen, there are 36 representative Oregon counties of various shades of blue.

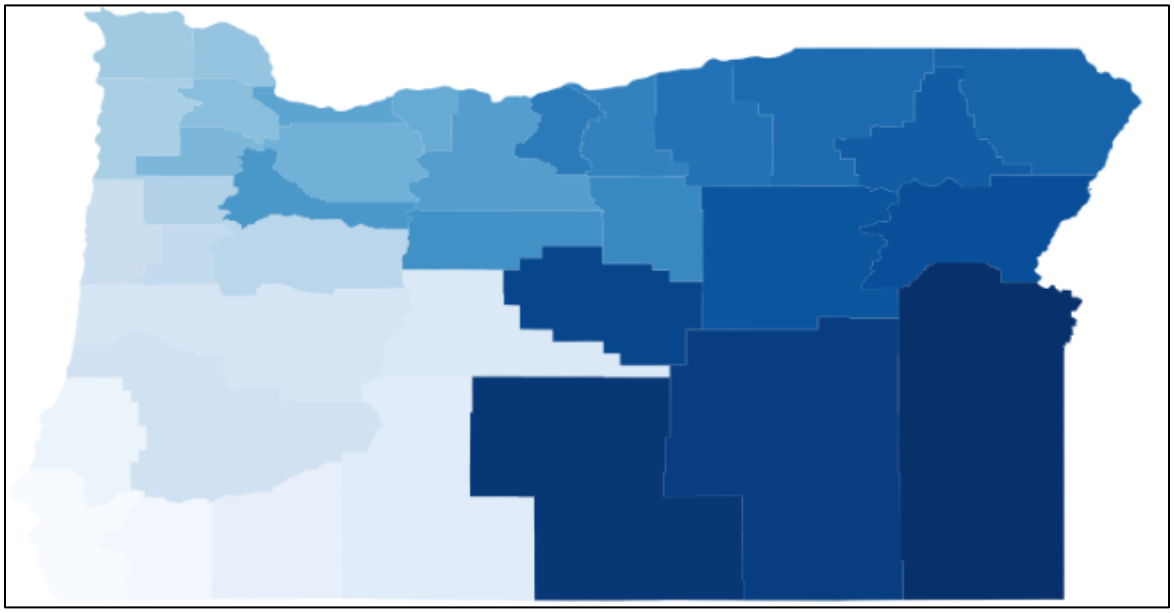


Figure 21: Oregon and its respective counties in hues of blue

Then, we mapped the counties to their respective redistricting location using the network package and county to district color list.

1.2.7 Step 7: Node Mapping the counties and districts

```
filepath = '/Users/jessi/Gurobi-Python Playground/OR.dimacs-mod.txt'
G = nx.read_edgelist(filepath,nodetype=int)

print(type(G))

districts = ["District 0","District 1","District 2","District 3","District␣
 ↪4","District 5"]

# Assigning a dictionary for the districts
counties_district_dict = {
"District 0": [0, 16, 7, 32, 33, 20, 30, 19, 34],
"District 1":  [22],
"District 2":  [14, 26, 3, 9, 13],
```

Figure 22: Node Color Map Part 1

```
"District 3":  [17, 31, 11, 35, 12, 23],
"District 4":  [8, 2, 29, 24, 28, 25, 27, 15],
"District 5":  [18, 1, 6, 4, 21, 5, 10]
}

# Assigning a color to each district
counties_district_colors_dict = {
    "District 0": "tab:red",
    "District 1": "tab:orange",
    "District 2": "tab:olive",
    "District 3": "tab:green",
    "District 4": "tab:blue",
    "District 5": "tab:brown",

}


# Assigning the colors
color_list = []
for node in G.nodes:   # iteration over nodes
    if node in counties_district_dict['District 0']:
        color_list.append(counties_district_colors_dict['District 0'])

    elif node in counties_district_dict['District 1']:
        color_list.append(counties_district_colors_dict['District 1'])

    elif node in counties_district_dict['District 2']:
        color_list.append(counties_district_colors_dict['District 2'])

    elif node in counties_district_dict['District 3']:
        color_list.append(counties_district_colors_dict['District 3'])

    elif node in counties_district_dict['District 4']:
        color_list.append(counties_district_colors_dict['District 4'])

    else:
        color_list.append(counties_district_colors_dict['District 5'])

# Map output
nx.draw(G, pos, node_color=color_list, with_labels=True)
plt.show()
```

Figure 23: Node Color Map Part 2

2.4.10 Tooling Development: Section 9

Following the tool development for Oregon state, we worked towards verification of our tool. We used a representative Oklahoma data set to import into the tool. All previously described 8 steps were applied to the Oklahoma data set. Unlike Oregon, the model was feasible and able to run with valid solutions for steps 5 through 8 of the tool. All constraints were met and not violated. Screenshots of the Oklahoma solution are available on Github.

2.5 Trials and Experimentation

With coding, the process is trial and error, research, and time. We spent approximately 45 hours total on the tool, and we have become better coders from the experience. We failed on the modeling, functions, data sets, and more, but we were able to continue to resolve problems as we have discussed throughout the paper.

2.5.1 Lesson Learned 1: Data type and text manipulation

- The population data sets have headers, so to run the data frame the header will need to be removed. Use next(file) in the code to accomplish this.
    - The dimacs files includes non-integers, so the following code can support the removal of the integers to run the data set. Otherwise, one will need to manually manipulate the data. The code ran with a modified text file to ensure consistency across testing the tool between Oklahoma and Oregon.

```
# If using the unmodified .dimacs file, it will need to be manipulated to only
 utilize integers. Removing column 1, row 1, last row
# filepath = "/Users/jessi/Gurobi-Python Playground/MA.dimacs"
# file = open(filepath)

# #remove the header and tail to over mixing int and str
# next(file)
# last_line = None
# for line in file:
#     if last_line:
#         for line in file:
#             if line.strip():
#                 file.write("\t".join(line.split()[1:]) + "\n")
#         print (last_line)
#         last_line = line
```

Figure 24: Data Manipulation Part 1

```
# # for line in file:
# #         try:
# #             r = int(line)
# #             if r > 999 or line.strip() == '-0':
# #                 #filtering numbers >999 and strings with '-0'
# #                 continue
# #             file.append(r)
# #         except ValueError:
# #             pass

# # Create List of integers
# # nodes_edges = []
# # for each in file:
# #     each = each.strip()
# #     #split the list so the populations in each range
# #     each = each.split(' ')
# #     print((each[1], each[2]))
# #     nodes_edges.append(int(each[1], each[2]))
```

Figure 25: Data Manipulation Part 2

2.5.2 Lesson Learned 2: For Loops

- The for loops in the model were built using practice for loops as shown below.

```
# #for loop examples and set-up
# ## x is the place holder that looks in the range
# for x in range(1, 11):
#     print(x)
# print("Range")

# #for loop counting backwards by two which is the step function
# for x in reversed (range(1, 11, 2)):
#     print(x)
# print("Backward")

# #printing within a sequence
# ## x holds current position
# Number = "1234-1976-3467"
# for x in Number:
#     print(x)
# print("Number sequence")

# #skipping a number and looping
# for x in range(1,21):
#     if x == 13:
#         continue
#     else:
#         print(x)

# #skipping a number and looping
```

Figure 26: For Loops practice part 1

```
# for x in range(1,21):
#     if x == 13:
#         break
#     else:
#         print(x)

# # Nested for loop examples
# # outer and inner loop, make sure the varible/placeholder/ counters are␣
 ↪different

# #call and response
# rows =int(input("Enter the # of rows: "))
# columns =int(input("Enter the # of columns: "))
# symbol = input("Enter a symbol to use: ")
# for x in range(rows): # means 3 iterations, outer loop
# # everytime you print it ends with a new character and proceeds to the next␣
 ↪line,
# # so if we use a string then it will be next to each other
#     for y in range(columns):
#         print (symbol, end =" ")
#     print() #exit inner loop

# # number string
# for x in range(3): # means 3 iterations, outer loop
# # everytime you print it ends with a new character and proceeds to the next␣
 ↪line,
# # so if we use a string then it will be next to each other
#     for y in range(1,10):
#         print (y, end ="")
#     print() #exit inner loop
```

Figure 27: For Loops Practice Part 2

2.5.3 Experimentation: Computational Data

The model was developed and run on a 2023 edition of the Apple M2 chip Macbook Air, and the description have been captured below. The optimization model for the Oklahoma model ran in under 3 minutes, while the Oregon model ran in approximately 8 hours. To run the model more efficiently, a window i7 processor computer or a desktop gaming computer with additional storage and RAM could run the model in approximately 1 hour. Since the model is infeasible, the optimization model would terminate at approximately 81% completion on both windows and apple equipment; therefore, the model would give the same out independent of the computation equipment.

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads
```

Figure 2828: Gurobi Python version

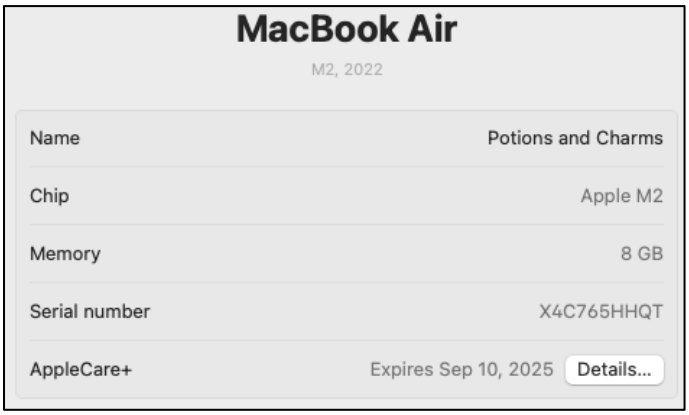


Figure 2929: Computational Equipment

3 Analysis and Evaluation

3.1 Districting Map

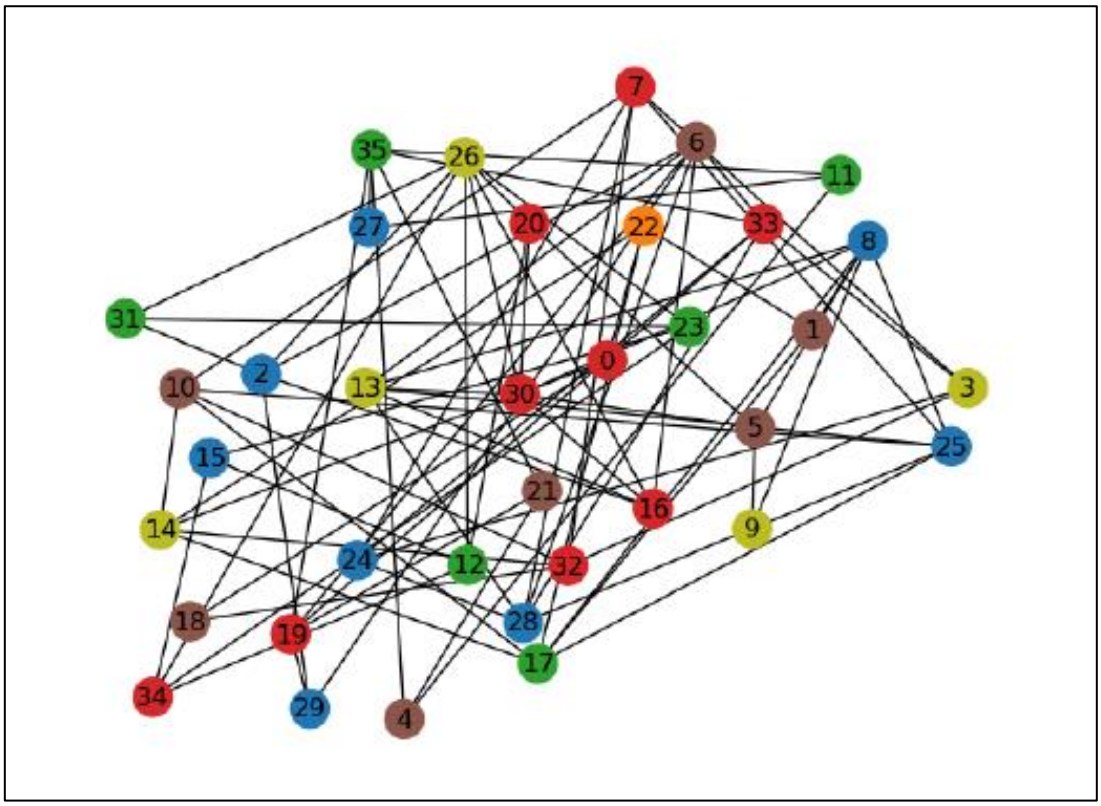


Figure 3030: Oregon Redistricting

Utilizing the tool we developed, the figure above is the output district map determined for Oregon. As mentioned previously in the report, this model does not represent and adhere to all requirements and constraints as described in section 1.2 Congressional Redistricting Modeling Requirements and Criteria. This is primarily due to:

1. Multnomah County, Oregon, node (22)
    a. This county is the smallest in physical area but contains the largest population out of all counties of Oregon. The county houses approximately 800,000 persons.
2. The limited number of districts allotted to Oregon due to the state's population.
3. A combination of constraints that disallowed model feasibility
    a. As described in proceeding section 3.2 Congressional Redistricting Criteria & Dataset Limitations

The figure does have 6 varying colors, as expected, to represent the 6 districts as well as the 36 nodes that represent each county.

3.2 Congressional Redistricting Criteria & Dataset Limitations

With the given dataset, redistricting through our developed tool alone is insufficient to meet all the constraints given to the program. The limiting constraints that were not met were the population deviation of 3% as well as identified constraints [2] and [3], "districts of (mostly) equal population" with "districts utilizing existing political and geographic boundaries", respectively.

3.3 Analysis Limitations

We were unable to meaningfully compare the previous (established prior to 2023) congressional districts with our proposed (post 2023) congressional district due to the infeasibility of the model, as mentioned in section 3.2 Congressional Redistricting Criteria & Dataset Limitations. Our model violates the constraint criteria and could not be solved.

4 Conclusion

In conclusion, the redistricting tool that has been created does work for feasible models. For infeasible models, the code does output the expected result. The tool was utilized to identity the county populations, district population deviations, lower and upper bounds of population, minimize the cut edges, county to district assignments, and an overall analysis of the redistricting of Oregon. The project successfully tested the hypothesis determination that Oregon is an infeasible model, and that Oklahoma is a feasible model.

Appendix A: References

Code Acknowledgements, Reference, Adaptations:

GitHub Project file: https://github.com/Jdedeaux/Districts-Counties/tree/main

Credit to:

- *Dr. Hamidreza Validi transportation, short path codes, and assignment codes: IE 5318 Principles of Operations Research at Texas Tech University*
- *Dr. Austin Buchanan districting example starter code: https://github.com/AustinLBuchanan/Districting-Examples/tree/main*
- *Gurobi Optimization Tutorial via the primary website: https://www.gurobi.com/resources/lp-chapter-6-modeling-and-solving-linear-programming-problems/*
- *Dr. Gabrielle Taylor: https://github.com/gabrielletay96*
- *shapefiles: https://www.youtube.com/watch?v=xxg4Vm-Xg9g*
- *for loops: https://www.youtube.com/watch?v=KWgYha0clzw&t=172s*
- *Data: http://people.csail.mit.edu/ddeford/dual_graphs*
- *Data: https://lykhovyd.com/files/public/districting/2020/*
- *nested for loops: https://www.youtube.com/watch?v=APWy6Pc83gE*
- *Networknx Package: https://networkx.org/documentation/stable/auto_examples/drawing/plot_labels_and_colors.html*

Report References:

2010 Redistricting Deviation Table. (2020, January 15). https://www.ncsl.org/redistricting-and-census/2010-redistricting-deviation-table

2020 Redistricting Deviation Table. (2023, October 17). https://www.ncsl.org/elections-and-campaigns/2020-redistricting-deviation-table

Bills and Laws OrConst. (2022). https://www.oregonlegislature.gov/bills_laws/Pages/OrConst.aspx

Bureau, U. C. (2021a, April 26). *Historical Apportionment Data (1910-2020)*. Census.Gov. https://www.census.gov/data/tables/time-series/dec/apportionment-data-text.html

Bureau, U. C. (2021b, October 8). *About Congressional Districts*. Census.Gov. https://www.census.gov/programs-surveys/geography/guidance/geo-areas/congressional-dist.html

Gerrymandering Project. (2023, July 19). *Redistricting Report Card—Oregon 2021 Final Congressional Map—Enacted*. https://gerrymander.princeton.edu/redistricting-report-card/?planId=rec6qj1vAOKsBnXnu

NCSL Document links. (2021). https://www.oregonlegislature.gov/bills_laws/ors/ors188.html

OR Redistricting Resources. (2023). Austin Buchanan. https://AustinLBuchanan.github.io/OR-redistricting-resources/

Oregon—Redistricting Data. (2023). *Redistricting Data Hub*. https://redistrictingdatahub.org/state/oregon/

ORS. (2021). *Chapter 188—Congressional and Legislative Districts; Reapportionment*. https://www.oregonlegislature.gov/bills_laws/ors/ors188.html

Policy Research. (2023). https://www.ncsl.org/research/redistricting/2010-ncsl-redistricting-deviation-table

Redistricting Criteria. (2023). https://www.ncsl.org/redistricting-and-census/redistricting-criteria

Redistricting in Oregon. (2023). Ballotpedia. https://ballotpedia.org/Redistricting_in_Oregon

Redistricting in Oregon after the 2010 census. (2011). Ballotpedia. https://ballotpedia.org/Redistricting_in_Oregon_after_the_2010_census

Vtd_data. (2023). [Computer software]. Dave's Redistricting. https://github.com/dra2020/vtd_data (Original work published 2020)