ARKANSAS(AR) REDISTRICTING REPORT

Dr. Hamidreza Validi, IE5318

DECEMBER 1, 2023

IMSE|TTU
Ali Azizi Deh Sorkh

Contents

Executive Summary Letter

Political redistricting holds immense significance in the United States due to the nation's deep-rooted commitment to democratic ideals. Ensuring equitable design of political districts is paramount as it safeguards the integrity of Americans' votes and fosters a more balanced society. This report will elaborate on our approach to addressing the challenge of political redistricting in the state of Arkansas.

Initially, we examined both federal and state redistricting criteria, followed by gathering data to support our ongoing development. Subsequently, a linear program was devised to construct a model showcasing a potential districting layout for Arkansas.

The devised redistricting blueprint revolves around establishing districts with an approximate populace of 750,000 individuals per district. This plan maintains a population deviation of 0.28%, aligning faithfully with the redistricting standards mandated at both state and federal levels.

Introduction

Every decade, as mandated by the United States Constitution, a census is conducted. The outcomes of this survey dictate whether federal and state governing bodies need to undergo redistricting, reassignment, or redistribution.

Redistricting refers to the process of dividing a political entity into smaller, nearly equal population segments to prevent favoritism toward specific demographics or regions. Any proposed district scheme must adhere to the redistricting criteria established by both federal and state authorities.

Given variations in population across diverse spatial areas and the constraints imposed by geographic, political, and social factors, redistricting poses a highly intricate challenge often resolved through optimization models. The approach employed in this solution utilizes the minimum cut edges technique.

Criteria

Federal Criteria

Every state within the United States of America must abide by the following criteria points:

- Each district must have a nearly equal population in relation to one another.
  - Apportionment Clause, Article, Section 2, U.S. Constitution
- Each district must not intentionally discriminate racially.
  - Voting Rights Act, 1965

These points are enforced at the federal level. In addition to this, each state has the freedom to

impose their criteria.

State of Arkansas Criteria

The state of Arkansas requires that the following criteria points are followed:

- Each district must maintain a +/- 5% population deviation.
- Each district must not intentionally discriminate or favor any race.
- Districts cannot be redrawn strictly based on race.

Source: https://arkansasredistricting.org/about-the-process/redistricting-criteria-and-goals/

In the redistricting process in Arkansas, there are few specific constraints imposed. Ideally, districts should be compact and connected, while also maintaining the integrity of counties, political divisions, communities of shared interests, and existing districts. Whenever feasible, efforts are made to avoid grouping incumbents within the same district. Factors considered in selecting a districting plan involve assessing its impact on incumbents, its use of partisan information, and its overall competitiveness.

Problem Statement

We'll employ operations research methodology to craft a districting blueprint for Arkansas that complies with both federal and state governmental constraints.

Problem Statement and OR model (words, math)

The models for Processes 1 and 2 exhibit considerable similarity. Hence, we will solely focus on the model that generated the most optimal redistricting map. The objective of this process is to minimize the total length of cut edges, ensuring compactness for each district. Preserving counties in their entirety is a priority during this process.

I'm unable to directly manipulate external files or formats, but I can provide the information in a format that you can copy and paste into a Word document:

Sets:

- (C) represents the set of counties (Nodes) 1, 2, ..., 74
- (J) represents the set of districts 1, 2, ..., (k)
- (E) represents the set of edges 1, 2, ..., (e)

- (N(i)) is the set of neighbors of county (i)

Indices:

- (i) denotes a county
- (j) denotes a district
- (u) and (v) are counties that are being checked for continuity

Parameters:

- (P_i) = population of county (i)
- (totalpop_j) = total population of district (j)
- (e) = total number of edges
- (n) = number of counties (nodes)
- (k) = total number of desired districts
- (L) = lower bound maximum deviation
- (U) = upper bound maximum deviation
- (M) = number of counties - number of districts + 1

Variables:

- (x{ij}) = {1 if county (i) is a part of district (j), 0 otherwise}
- (y_{uv}) = {1 if there is a district boundary between counties (u) and (v), 0 otherwise}
- (r_{ij}) = {1 if county (i) is the root of district (j), 0 otherwise}
- (f_{ij}) = {1 if flow is being sent from (i) to (j), 0 otherwise}

This information can be copied and pasted into a Word document or any text editor of your choice.

Minimize the sum of cut edges.

$$\min \sum_u \sum_v y_{uv}$$

Subject to:

Each county i is assigned to a district j:

$$\sum_i x_{ij} = 1 \quad \text{for all } j \in J$$

Each district j has a population at most U:

$$\sum_i P_i x_{ij} \leq U \quad \text{for all } j \in J$$

Each district j has a population at least L:

$$\sum_i P_i x_{ij} \geq L \quad \text{for all } j \in J$$

An edge is cut if u is assigned to district j but v is not:

$$x_{uj} - x_{vj} \leq y_{uv} \quad \text{for all } u \in E, v \in E, j \in J$$

Each district should have one root:

$$\sum_i r_{ij} = 1 \quad \text{for all } j \in J$$

If node i isn't assigned to district j, then it cannot be its root:

$$r_{ij} \leq x_{ij} \quad \text{for all } j \in J, i \in C$$

Only send flow to a root:

$$\sum_{j \in N(i)} (f_{ij} - f_{ji}) \leq 1 - M \times \left(\sum_j r_{ij} \right) \quad \text{for all } i \in C$$

Do not send flow across cut edges:

$$f_{ij} + f_{ji} \leq M \times (1 - y_{ij}) \quad \text{for all } i \in E, j \in E$$

Python/Gurobi Code

I have run three different models based on Dr. Buchanan's provided information on this link: https://github.com/AustinLBuchanan/Districting-Examples-2020

Min Cut edge

The code: (All codes are available at my GitHub Directory)

```
#import all necessary packages
import gurobipy as gp
from gurobipy import GRB
from gerrychain import Graph
import networkx as nx
import geopandas as gpd
import math

#Set filepath and filename equal to the path/name of the data used respectively
filepath = r'C:\Users\aliaz\Downloads\IEM40132020RedistrictingProject-main\IEM40132020RedistrictingProject-main/'
filename= 'AR_county.json'
#Create a new Graph object G from the file
G = Graph.from_json(filepath + filename)

#Set each node in G to be equal to the population of their respective county
for node in G.nodes:
    G.nodes[node]['TOTPOP'] = G.nodes[node]['P0010001']

#Print each node, the county it represents, and their 2020 population
for node in G.nodes:
```

```
    name = G.nodes[node]['NAME20']
    population = G.nodes[node]['TOTPOP']
    print("Node",node,"represents",name,"County with 2020 population of",population)

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

#set the ceiling and floor of the model equal to the maximum deviation/2 * the average
population
dev = 0.01

k = 4
tot_pop = sum(G.nodes[node]['TOTPOP'] for node in G.nodes)

L = math.ceil((1-dev/2)*tot_pop/k)
U = math.floor((1+dev/2)*tot_pop/k)
print("Using L =",L,"and U =",U,"and k =",k)

#create a new model object and create variables
m = gp.Model()

x = m.addVars(G.nodes, k, vtype=GRB.BINARY)
y = m.addVars(G.edges, vtype=GRB.BINARY)

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

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

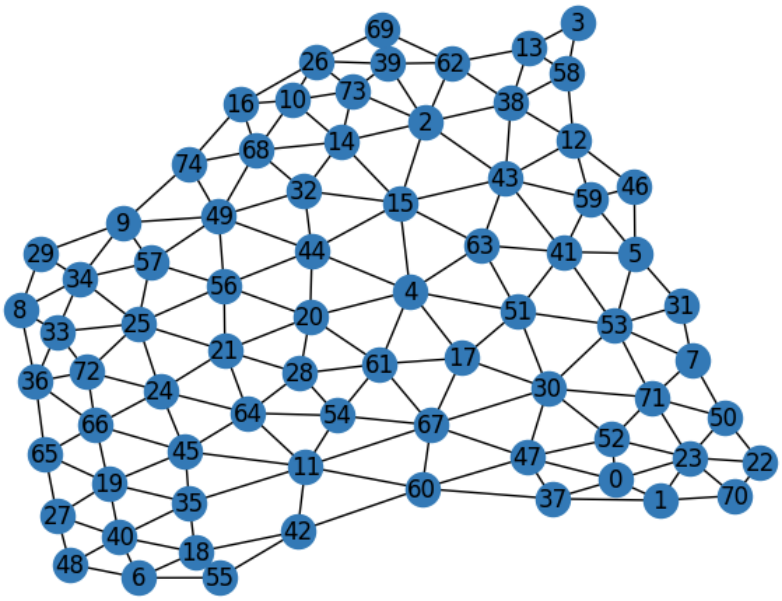
```
# each district j has a population at least L and at most U
m.addConstrs( gp.quicksum( G.nodes[i]['TOTPOP'] * x[i,j] for i in G.nodes) >= L for j in range(k))
m.addConstrs( gp.quicksum( G.nodes[i]['TOTPOP'] * x[i,j] for i in G.nodes) <= U for j in range(k))
# an edge is cut if u is assigned to district j but v is not.
m.addConstrs( x[u,j] - x[v,j] <= y[u,v] for u,v in G.edges for j in range(k))
m.update()

# add root variables: r[i,j] equals 1 if node i is the root of district j
r = m.addVars( G.nodes, k, vtype=GRB.BINARY)

import networkx as nx

DG = nx.DiGraph(G)

f = m.addVars(DG.edges)

# The big-M proposed by Hojny et al.
M = G.number_of_nodes() - k + 1
# each district should have one root
m.addConstrs( gp.quicksum( r[i,j] for i in G.nodes ) == 1 for j in range(k) )
# If node i isn't assigned to district j, then it cannot be its root
m.addConstrs( r[i,j] <= x[i,j] for i in G.nodes for j in range(k) )
# If not a root, consume some flow
# If a root, only send out (so much) flow
m.addConstrs( gp.quicksum( f[j,i] - f[i,j] for j in G.neighbors(i) )
        >= 1 - M * gp.quicksum( r[i,j] for j in range(k) ) for i in G.nodes )
# Do not send flow across cut edges
```

```
m.addConstrs( f[i,j] + f[j,i] <= M * (1-y[i,j] )for i,j in G.edges)


m.update()


# sole IP model
m.optimize()


print("The number of cut edges is",m.objval)


# retrieve the districts and their population
districts = [[i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
district_counties = [[G.nodes[i]["NAME20"] for i in districts[j] ] for j in range(k)]
district_populations = [sum(G.nodes[i]["TOTPOP"] for i in districts[j]) for j in range(k)]
# print it
for j in range(k):
    print("District",j,"has population",district_populations[j],"and contains
counties",district_counties[j])
    print("")


# Read Arkansas county shapefile from "AR_county.shp"
filepath = r'C:\Users\aliaz\Downloads\IEM40132020RedistrictingProject-
main\IEM40132020RedistrictingProject-main/'
filename = 'AR_county.shp'
# Read geopandas dataframe from file
df = gpd.read_file( filepath + filename)


# Which district is each county assigned to?
assignment = [ -1 for i in G.nodes ]
```

```
labeling = { i : j for i in G.nodes for j in range(k) if x[i,j].x > 0.5 }
# add assignments to a column of the dataframe and map it
node_with_this_geoid = { G.nodes[i]['GEOID20'] : i for i in G.nodes }
# pick a position u in the data frame
for u in range(G.number_of_nodes()):

    geoid = df['GEOID20'][u]
    i = node_with_this_geoid[geoid]
    assignment[u] = labeling[i]
#print the map
df['assignment'] = assignment
my_fig = df.plot(column='assignment').get_figure()
```

Result:

1- Node Graph



2- Optimization

```
Root relaxation: objective 0.000000e+00, 581 iterations, 0.01 seconds (0.01 work units)

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

     0     0    0.00000    0  316          -    0.00000      -     -    0s
     0     0    1.10613    0  333          -    1.10613      -     -    0s
     0     0    3.93013    0  322          -    3.93013      -     -    0s
     0     0    4.34882    0  327          -    4.34882      -     -    0s
     0     0    4.39393    0  327          -    4.39393      -     -    0s
     0     0    4.40626    0  327          -    4.40626      -     -    0s
     0     0    4.41369    0  325          -    4.41369      -     -    0s
     0     0    4.41843    0  325          -    4.41843      -     -    0s
     0     0    5.48044    0  333          -    5.48044      -     -    0s
     0     0    5.48044    0  327          -    5.48044      -     -    0s
     0     0    5.48044    0  326          -    5.48044      -     -    0s
     0     0    5.48044    0  330          -    5.48044      -     -    0s
     0     0    5.48044    0  324          -    5.48044      -     -    0s
     0     0    5.48044    0  323          -    5.48044      -     -    0s
     0     0    5.48044    0  322          -    5.48044      -     -    0s
     0     0    5.48044    0  321          -    5.48044      -     -    0s
     0     2    5.48044    0  316          -    5.48044      -     -    0s
  7992  6755   54.48426   53  125          -   17.41645      -  39.2    5s
 30757 21818   48.08989   51  166          -   19.92197      -  65.5   10s
H30793  6088                       38.0000000   19.92197  47.6%  65.5   10s
H30886  4929                       36.0000000   19.95255  44.6%  65.5   11s
H30997  4416                       35.0000000   19.95255  43.0%  65.6   11s
H32939  4616                       34.0000000   20.20777  40.6%  67.2   12s
 41076  7705   31.11349   26  341   34.00000   24.10354  29.1%  72.1   15s
 59394 12196     cutoff   25        34.00000   25.79783  24.1%  77.5   25s
 73745 15138   29.83848   26  214   34.00000   26.60561  21.7%  79.3   30s
 82708 16770   29.29823   26  324   34.00000   27.01935  20.5%  79.9   36s
 93109 18357   28.47355   27  203   34.00000   27.39762  19.4%  80.0   40s
 105986 19870     cutoff   27        34.00000   27.82121  18.2%  79.8   45s
 117417 21027     cutoff   33        34.00000   28.13270  17.3%  79.7   51s
 128219 21736   32.60574   31  205   34.00000   28.40277  16.5%  79.5   55s
 141903 22510     cutoff   35        34.00000   28.69585  15.6%  79.0   60s
 156269 23039   30.19681   25  231   34.00000   29.01743  14.7%  78.5   66s
 169404 23078     cutoff   28        34.00000   29.25911  13.9%  78.0   71s
 179061 22854     cutoff   42        34.00000   29.43253  13.4%  77.6   75s
 191725 22390   30.89467   32  224   34.00000   29.63450  12.8%  77.1   80s
 205205 21549   32.82422   42  158   34.00000   29.88787  12.1%  76.6   85s
 218594 20246   31.44972   32  244   34.00000   30.12245  11.4%  76.1   90s
 231496 18649   32.85426   36  236   34.00000   30.35092  10.7%  75.5   95s
 245066 16363   32.98620   37  272   34.00000   30.58627  10.0%  75.0  100s
 258719 13527   32.53408   31  139   34.00000   30.87634  9.19%  74.4  105s
 271684  9629   32.36960   31  133   34.00000   31.18686  8.27%  73.8  110s
*273145  6287              40      33.0000000   31.19236  5.48%  73.7  110s

Cutting planes:
  Gomory: 29
  Cover: 1
  MIR: 2
  Flow cover: 4
  RLT: 20

Explored 282341 nodes (20914392 simplex iterations) in 113.57 seconds (123.61 work units)
Thread count was 32 (of 32 available processors)

Solution count 6: 33 33 34 ... 38

Optimal solution found (tolerance 1.00e-04)
Best objective 3.300000000000e+01, best bound 3.300000000000e+01, gap 0.0000%
```

The number of cut edges is 33.0.

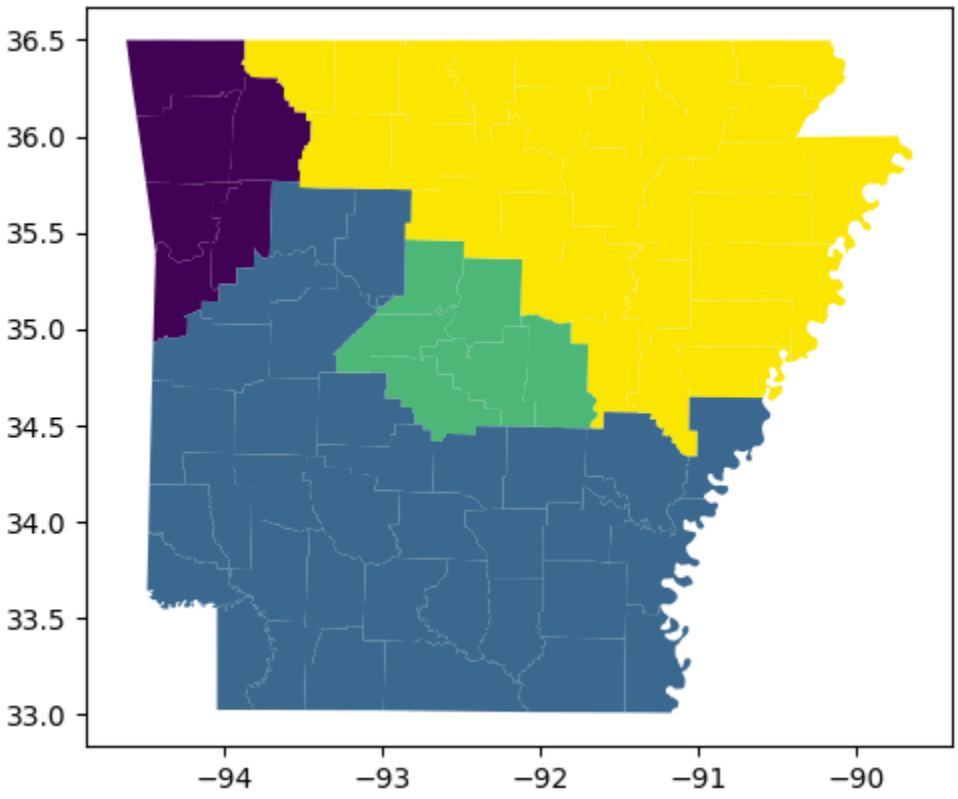
District 0 has a population of 751754 and contains counties ['Franklin', 'Crawford', 'Benton', 'Madison', 'Sebastian', 'Washington']

District 1 has population 754547 and contains counties ['Little River', 'Ashley', 'Desha', 'Montgomery', 'Howard', 'Nevada', 'Grant', 'Dallas', 'Cleveland', 'Lafayette', 'Chicot', 'Pope', 'Bradley', 'Drew', 'Pike', 'Union', 'Hempstead', 'Polk', 'Clark', 'Logan', 'Miller', 'Arkansas', 'Johnson', 'Garland', 'Sevier', 'Jefferson', 'Lincoln', 'Scott', 'Hot Spring', 'Columbia', 'Ouachita', 'Yell', 'Calhoun', 'Phillips']

District 2 has a population of 750788 and contains counties ['Faulkner', 'Conway', 'Pulaski', 'Saline', 'Lonoke', 'Perry']

District 3 has a population of 754435 and contains counties ['Jackson', 'Clay', 'Baxter', 'Boone', 'St. Francis', 'Sharp', 'Greene', 'Woodruff', 'White', 'Lee', 'Crittenden', 'Marion', 'Prairie', 'Lawrence', 'Poinsett', 'Stone', 'Independence', 'Fulton', 'Carroll', 'Van Buren', 'Searcy', 'Randolph', 'Izard', 'Craighead', 'Cleburne', 'Monroe', 'Mississippi', 'Newton', 'Cross']

3- Map



Min Moment of Inertia

The code:

```
from gerrychain import Graph
import gurobipy as gp
from gurobipy import GRB
import math
from geopy.distance import geodesic
import geopandas as gpd

#Set filepath and filename equal to the path/name of the data used respectively
filepath = r'C:\Users\aliaz\OneDrive\Desktop\Progress\Courses\Fall2023\OR\Final Project/'
filename= 'AR_county.json'

#Create a new Graph object G from the file
G = Graph.from_json(filepath + filename)

for node in G.nodes:
    G.nodes[node]['TOTPOP'] = G.nodes[node]['P0010001']
    G.nodes[node]['C_X'] = G.nodes[node]['INTPTLON20']
    G.nodes[node]['C_Y'] = G.nodes[node]['INTPTLAT20']

# create distance dictionary
dist = { (i,j) : 0 for i in G.nodes for j in G.nodes }
for i in G.nodes:
    for j in G.nodes:
        loc_i = ( G.nodes[i]['C_Y'], G.nodes[i]['C_X'] )
        loc_j = ( G.nodes[j]['C_Y'], G.nodes[j]['C_X'] )
        dist[i,j] = geodesic(loc_i,loc_j).miles
```

```
dev = 0.01

k = 4

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

L = math.ceil((1-dev/2)*tot_pop/k)

U = math.floor((1+dev/2)*tot_pop/k)

print("Using L =",L,"and U =",U,"and k =",k)

m = gp.Model()

x = m.addVars(G.nodes, G.nodes, vtype=GRB.BINARY)

m.setObjective( gp.quicksum( dist[i,j] * dist[i,j] * G.nodes[i]['TOTPOP'] * x[i,j] for i in G.nodes
for j in G.nodes ), GRB.MINIMIZE )


# add constraints saying that each county i is assigned to one district

m.addConstrs( gp.quicksum( x[i,j] for j in G.nodes ) == 1 for i in G.nodes )


# add constraint saying there should be k district centers

m.addConstr( gp.quicksum( x[j,j] for j in G.nodes ) == k )


# add constraints that say: if j roots a district, then its population is between L and U.

m.addConstrs( gp.quicksum( G.nodes[i]['TOTPOP'] * x[i,j] for i in G.nodes ) >= L * x[j,j] for j
in G.nodes )

m.addConstrs( gp.quicksum( G.nodes[i]['TOTPOP'] * x[i,j] for i in G.nodes ) <= U * x[j,j] for j
in G.nodes )


# add coupling constraints saying that if i is assigned to j, then j is a center.

m.addConstrs( x[i,j] <= x[j,j] for i in G.nodes for j in G.nodes )


m.update()
```

```
# add contiguity constraints
import networkx as nx
DG = nx.DiGraph(G)

# add flow variables
#    f[i,j,v] = flow across arc (i,j) that is sent from source/root v
f = m.addVars( DG.edges, G.nodes )

# add constraints saying that if node i is assigned to node j,
#   then node i must consume one unit of node j's flow
m.addConstrs( gp.quicksum( f[u,i,j] - f[i,u,j] for u in G.neighbors(i) ) == x[i,j] for i in G.nodes
for j in G.nodes if i != j )

# add constraints saying that node i can receive flow of type j
#   only if node i is assigned to node j
M = G.number_of_nodes() - 1
m.addConstrs( gp.quicksum( f[u,i,j] for u in G.neighbors(i) ) <= M * x[i,j] for i in G.nodes for j
in G.nodes if i != j )

# add constraints saying that node j cannot receive flow of its own type
m.addConstrs( gp.quicksum( f[u,j,j] for u in G.neighbors(j) ) == 0 for j in G.nodes )

m.update()
m.Params.MIPGap = 0.0
m.optimize()

# print the objective value
print(m.ObjVal)
```

```
# retrieve the districts and their populations
#    but first get the district "centers"
centers = [ j for j in G.nodes if x[j,j].x > 0.5 ]
districts = [ [ i for i in G.nodes if x[i,j].x > 0.5 ] for j in centers ]
district_counties = [ [ G.nodes[i]["NAME20"] for i in districts[j] ] for j in range(k)]
district_populations = [ sum(G.nodes[i]["TOTPOP"] for i in districts[j]) for j in range(k) ]
# print district info
for j in range(k):
   print("District",j,"has population",district_populations[j],"and contains
counties",district_counties[j])
   print("")


filepath = 'C:/Users/Logan/Desktop\College/IEM4013Project/'
filename = 'AR_county.shp'
df = gpd.read_file( filepath + filename)


# Which district is each county assigned to?
assignment = [ -1 for i in G.nodes ]


labeling = { i : -1 for i in G.nodes }
for j in range(k):
   district = districts[j]
   for i in district:
      labeling[i] = j


# Now add the assignments to a column of the dataframe and map it
node_with_this_geoid = { G.nodes[i]['GEOID20'] : i for i in G.nodes }
```

```
# pick a position u in the dataframe
for u in range(G.number_of_nodes()):

    geoid = df['GEOID20'][u]

    # what node in G has this geoid?
    i = node_with_this_geoid[geoid]

    # position u in the dataframe should be given
    # the same district # that county i has in 'labeling'
    assignment[u] = labeling[i]

# now add the assignments to a column of our dataframe and then map it
df['assignment'] = assignment
my_fig = df.plot(column='assignment').get_figure()
```

Results

District 0 has population 753326 and contains counties ['Jackson', 'Clay', 'Baxter', 'St. Francis', 'Sharp', 'Greene', 'Woodruff', 'White', 'Lee', 'Crittenden', 'Marion', 'Prairie', 'Lawrence', 'Poinsett', 'Independence', 'Lonoke', 'Fulton', 'Arkansas', 'Randolph', 'Izard', 'Craighead', 'Cleburne', 'Monroe', 'Mississippi', 'Cross', 'Phillips']

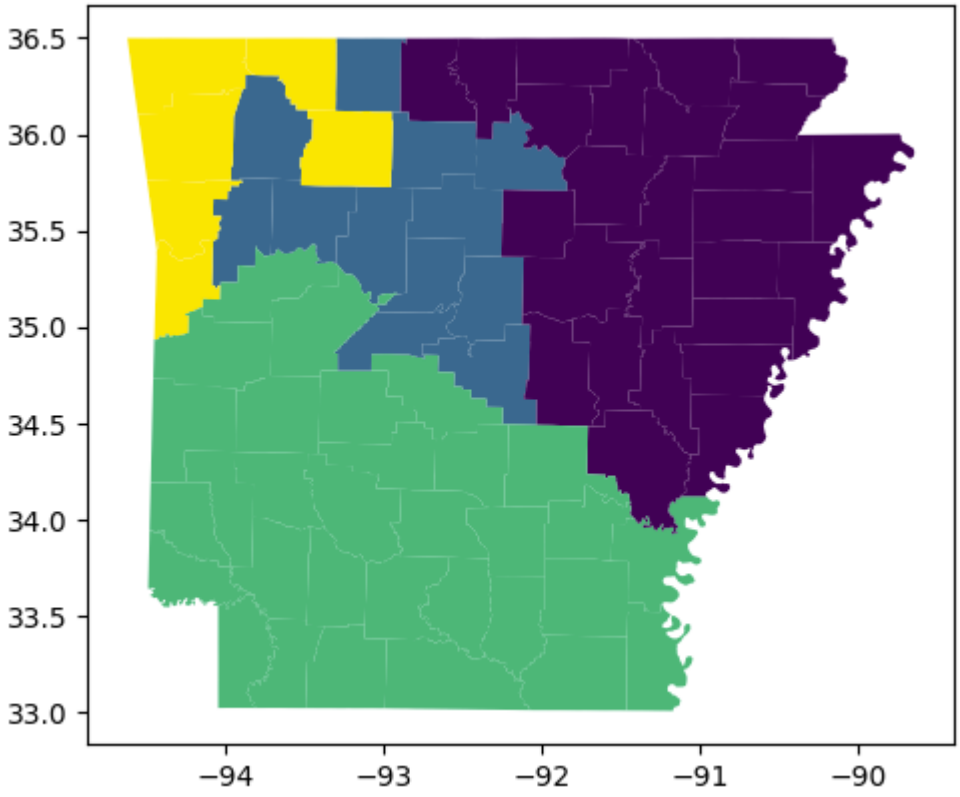
District 1 has population 749461 and contains counties ['Franklin', 'Faulkner', 'Boone', 'Conway', 'Pulaski', 'Madison', 'Pope', 'Stone', 'Van Buren', 'Johnson', 'Searcy', 'Perry']

District 2 has population 755116 and contains counties ['Little River', 'Ashley', 'Desha', 'Montgomery', 'Howard', 'Nevada', 'Grant', 'Dallas', 'Cleveland', 'Lafayette', 'Saline', 'Chicot', 'Bradley', 'Drew', 'Pike', 'Union', 'Hempstead', 'Polk', 'Clark', 'Logan', 'Miller', 'Garland', 'Sevier', 'Jefferson', 'Lincoln', 'Scott', 'Hot Spring', 'Columbia', 'Ouachita', 'Yell', 'Calhoun']

District 3 has population 753621 and contains counties ['Crawford', 'Benton', 'Sebastian', 'Carroll', 'Washington', 'Newton']

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

     0     0 6.3081e+09    0   44          - 6.3081e+09      -      -    0s
H    0     0                      6.998876e+09 6.3081e+09  9.87%      -    0s
H    0     0                      6.883477e+09 6.3081e+09  8.36%      -    0s
     0     0 6.3997e+09    0  158 6.8835e+09 6.3997e+09  7.03%      -    0s
     0     0 6.3997e+09    0   44 6.8835e+09 6.3997e+09  7.03%      -    1s
     0     0 6.3997e+09    0  158 6.8835e+09 6.3997e+09  7.03%      -    1s
     0     0 6.4904e+09    0  160 6.8835e+09 6.4904e+09  5.71%      -    1s
     0     0 6.4917e+09    0  160 6.8835e+09 6.4917e+09  5.69%      -    1s
     0     0 6.5012e+09    0  160 6.8835e+09 6.5012e+09  5.55%      -    1s
     0     0 6.5114e+09    0  132 6.8835e+09 6.5114e+09  5.41%      -    1s
     0     0 6.5294e+09    0  130 6.8835e+09 6.5294e+09  5.14%      -    1s
     0     0 6.5984e+09    0  134 6.8835e+09 6.5984e+09  4.14%      -    1s
     0     0 6.6054e+09    0  135 6.8835e+09 6.6054e+09  4.04%      -    1s
     0     0 6.6060e+09    0  137 6.8835e+09 6.6060e+09  4.03%      -    1s
     0     0 6.6259e+09    0  199 6.8835e+09 6.6259e+09  3.74%      -    2s
     0     0 6.6481e+09    0  199 6.8835e+09 6.6481e+09  3.42%      -    2s
     0     0 6.6536e+09    0  200 6.8835e+09 6.6536e+09  3.34%      -    2s
     0     0 6.6536e+09    0  201 6.8835e+09 6.6536e+09  3.34%      -    2s
H    0     0                      6.861489e+09 6.6633e+09  2.89%      -    2s
     0     0 6.6633e+09    0  205 6.8615e+09 6.6633e+09  2.89%      -    2s
     0     0 6.6633e+09    0   44 6.8615e+09 6.6633e+09  2.89%      -    2s
H    0     0                      6.835095e+09 6.6633e+09  2.51%      -    2s
     0     0 6.6633e+09    0  217 6.8351e+09 6.6633e+09  2.51%      -    3s
     0     0 6.6633e+09    0  251 6.8351e+09 6.6633e+09  2.51%      -    3s
     0     0 6.6633e+09    0  267 6.8351e+09 6.6633e+09  2.51%      -    3s
     0     0 6.6633e+09    0  231 6.8351e+09 6.6633e+09  2.51%      -    3s
     0     0 6.6633e+09    0  221 6.8351e+09 6.6633e+09  2.51%      -    3s
     0     0 6.6633e+09    0  225 6.8351e+09 6.6633e+09  2.51%      -    3s
     0     0 6.6633e+09    0  234 6.8351e+09 6.6633e+09  2.51%      -    3s
     0     0 6.6633e+09    0  234 6.8351e+09 6.6633e+09  2.51%      -    3s
     0     0 6.6752e+09    0  229 6.8351e+09 6.6752e+09  2.34%      -    3s
     0     0 6.7068e+09    0  225 6.8351e+09 6.7068e+09  1.88%      -    3s
     0     0 6.7112e+09    0  219 6.8351e+09 6.7112e+09  1.81%      -    3s
     0     0 6.7147e+09    0  232 6.8351e+09 6.7147e+09  1.76%      -    3s
     0     0 6.7453e+09    0  283 6.8351e+09 6.7453e+09  1.31%      -    3s
     0     0 6.7695e+09    0  293 6.8351e+09 6.7695e+09  0.96%      -    3s
     0     0 6.7737e+09    0  292 6.8351e+09 6.7737e+09  0.90%      -    3s
     0     0 6.8167e+09    0  293 6.8351e+09 6.8167e+09  0.27%      -    3s
     0     0 6.8180e+09    0  293 6.8351e+09 6.8180e+09  0.25%      -    3s
     0     0 6.8180e+09    0  143 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  213 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  215 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  238 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  192 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  204 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  225 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  229 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  167 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  167 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  249 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  271 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  271 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  266 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     0 6.8180e+09    0  266 6.8351e+09 6.8180e+09  0.25%      -    4s
     0     2 6.8180e+09    0  266 6.8351e+09 6.8180e+09  0.25%      -    4s
     3     4 6.8180e+09    2  235 6.8351e+09 6.8180e+09  0.25%    275    5s

Cutting planes:
  Gomory: 5
  Cover: 24
  Clique: 1
  MIR: 6
  StrongCG: 3
  Flow cover: 41
  GUB cover: 7
  Network: 1
  RLT: 2

Explored 47 nodes (13960 simplex iterations) in 5.15 seconds (6.99 work units)
Thread count was 32 (of 32 available processors)

Solution count 4: 6.8351e+09 6.86149e+09 6.88348e+09 6.99888e+09

Optimal solution found (tolerance 0.00e+00)
Best objective 6.835095216711e+09, best bound 6.835095216711e+09, gap 0.0000%
```

Map



Min Perimeter

The code:

```
from gerrychain import Graph
filepath = r'C:\Users\aliaz\OneDrive\Desktop\Progress\Courses\Fall2023\OR\Final Project/'
filename= 'AR_county.json'
G = Graph.from_json(filepath + filename)
for node in G.nodes:
    G.nodes[node]['TOTPOP'] = G.nodes[node]['P0010001']


dev = 0.01
import math
k = 4
```

```
tot_pop = sum(G.nodes[node]['TOTPOP'] for node in G.nodes)

L = math.ceil((1-dev/2)*tot_pop/k)
U = math.floor((1+dev/2)*tot_pop/k)
print("Using L =",L,"and U =",U,"and k =",k)

import gurobipy as gp
from gurobipy import GRB

# create model
m = gp.Model()

# create variables
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

m.setObjective( gp.quicksum( G.edges[u,v]['shared_perim'] * y[u,v] for u,v in G.edges ), GRB.MINIMIZE )
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 {u,v} is cut if u is assigned to district j but v is not.
m.addConstrs( x[u,j] - x[v,j] <= y[u,v] for u,v in G.edges for j in range(k) )
```

```
m.update()

# Add root variables: r[i,j] equals 1 if node i is the "root" of district j
r = m.addVars( G.nodes, k, vtype=GRB.BINARY)

# To solve the MIP faster, fix some district roots:

r[20,0].LB = 1  # fix Oklahoma county as root of district 0
r[37,1].LB = 1 # fix Tulsa county as root of district 1
r[62,2].LB = 1  # fix Comanche county as root of district 2
r[56,3].LB = 1
# Add flow variables: f[u,v] = amount of flow sent across arc uv
#  Flows are sent across arcs of the directed version of G which we call DG

import networkx as nx
DG = nx.DiGraph(G)      # directed version of G

f = m.addVars( DG.edges )

# The big-M proposed by Hojny et al.
M = G.number_of_nodes() - k + 1

# Each district j should have one root
m.addConstrs( gp.quicksum( r[i,j] for i in G.nodes ) == 1 for j in range(k) )

# If node i is not assigned to district j, then it cannot be its root
m.addConstrs( r[i,j] <= x[i,j] for i in G.nodes for j in range(k) )
```

```
# if not a root, consume some flow.
# if a root, only send out (so much) flow.
m.addConstrs( gp.quicksum( f[j,i] - f[i,j] for j in G.neighbors(i) )
        >= 1 - M * gp.quicksum( r[i,j] for j in range(k) ) for i in G.nodes )

# do not send flow across cut edges
m.addConstrs( f[i,j] + f[j,i] <= M * ( 1 - y[i,j] ) for i,j in G.edges )

m.update()

m.optimize()

print("The number of cut edges is", m.ObjVal)

districts = [[i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
district_counties = [[G.nodes[i]["NAME20"] for i in districts[j] ] for j in range(k)]
district_populations = [sum(G.nodes[i]["TOTPOP"] for i in districts[j]) for j in range(k)]

for j in range(k):
   print("District",j,"has population",district_populations[j],"and contains
counties",district_counties[j])
   print("")

import geopandas as gpd

filepath = r'C:\Users\aliaz\OneDrive\Desktop\Progress\Courses\Fall2023\OR\Final Project/'
filename = 'AR_county.shp'
df = gpd.read_file( filepath + filename)
```

```
assignment = [ -1 for i in G.nodes ]

labeling = { i : j for i in G.nodes for j in range(k) if x[i,j].x > 0.5 }

node_with_this_geoid = { G.nodes[i]['GEOID20'] : i for i in G.nodes }

for u in range(G.number_of_nodes()):

    geoid = df['GEOID20'][u]
    i = node_with_this_geoid[geoid]
    assignment[u] = labeling[i]

df['assignment'] = assignment
my_fig = df.plot(column='assignment').get_figure()
```

Result:

The number of cut edges is 12.00964554654077.

District 0 has population 750788 and contains counties ['Faulkner', 'Conway', 'Pulaski', 'Saline', 'Lonoke', 'Perry']

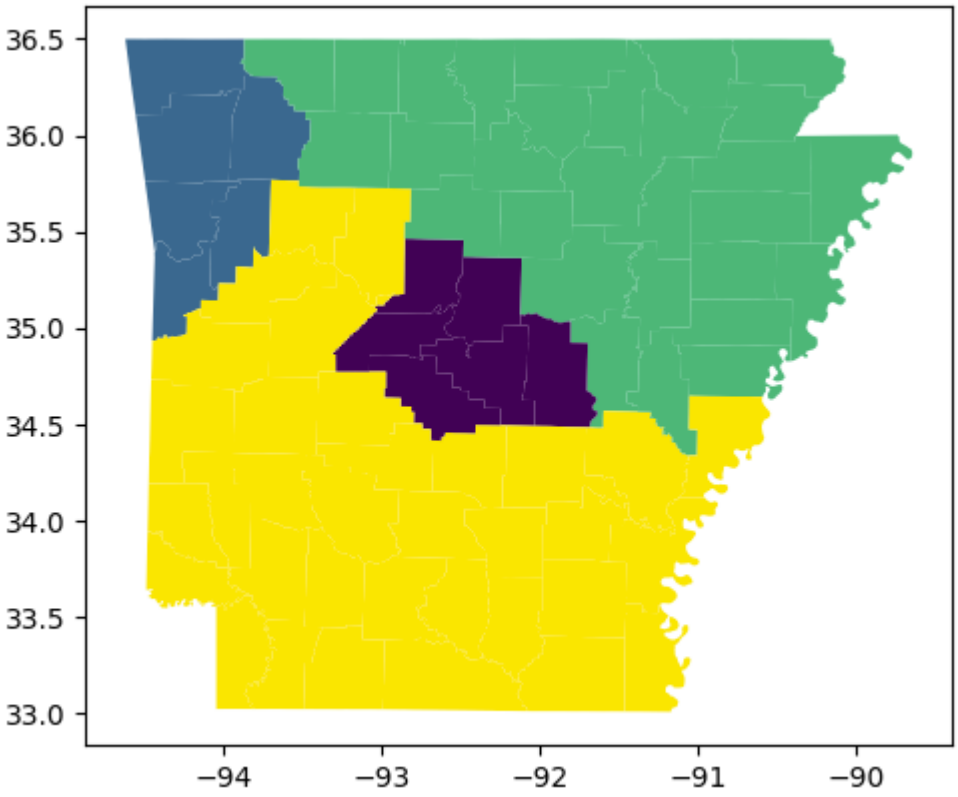
District 1 has population 751754 and contains counties ['Franklin', 'Crawford', 'Benton', 'Madison', 'Sebastian', 'Washington']

District 2 has population 754435 and contains counties ['Jackson', 'Clay', 'Baxter', 'Boone', 'St. Francis', 'Sharp', 'Greene', 'Woodruff', 'White', 'Lee', 'Crittenden', 'Marion', 'Prairie', 'Lawrence', 'Poinsett', 'Stone', 'Independence', 'Fulton', 'Carroll', 'Van Buren', 'Searcy', 'Randolph', 'Izard', 'Craighead', 'Cleburne', 'Monroe', 'Mississippi', 'Newton', 'Cross']

District 3 has population 754547 and contains counties ['Little River', 'Ashley', 'Desha', 'Montgomery', 'Howard', 'Nevada', 'Grant', 'Dallas', 'Cleveland', 'Lafayette', 'Chicot', 'Pope', 'Bradley', 'Drew', 'Pike', 'Union', 'Hempstead', 'Polk', 'Clark', 'Logan', 'Miller', 'Arkansas', 'Johnson', 'Garland', 'Sevier', 'Jefferson', 'Lincoln', 'Scott', 'Hot Spring', 'Columbia', 'Ouachita', 'Yell', 'Calhoun', 'Phillips']

```
Optimize a model with 1422 rows, 1176 columns and 5748 nonzeros
Model fingerprint: 0xf1f792f7
Variable types: 384 continuous, 792 integer (792 binary)
Coefficient statistics:
  Matrix range     [1e+00, 4e+05]
  Objective range  [2e-03, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 8e+05]
Presolve removed 389 rows and 338 columns
Presolve time: 0.01s
Presolved: 1033 rows, 838 columns, 4255 nonzeros
Variable types: 382 continuous, 456 integer (456 binary)

Root relaxation: objective 6.025434e+00, 1119 iterations, 0.02 seconds (0.03 work units)

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

     0     0    6.02543    0  291          -    6.02543      -     -    0s
     0     0    6.73934    0  305          -    6.73934      -     -    0s
     0     0    6.98085    0  336          -    6.98085      -     -    0s
     0     0    6.98098    0  337          -    6.98098      -     -    0s
     0     0    7.15631    0  308          -    7.15631      -     -    0s
     0     0    7.17367    0  316          -    7.17367      -     -    0s
     0     0    7.17614    0  362          -    7.17614      -     -    0s
     0     0    7.17754    0  327          -    7.17754      -     -    0s
     0     0    7.17759    0  339          -    7.17759      -     -    0s
     0     0    7.20103    0  311          -    7.20103      -     -    0s
     0     0    7.20481    0  322          -    7.20481      -     -    0s
     0     0    7.20514    0  322          -    7.20514      -     -    0s
     0     0    7.20514    0  322          -    7.20514      -     -    0s
     0     2    7.20561    0  322          -    7.20561      -     -    0s
* 8801  5194              88      18.3744340    9.02530  50.9%  48.3    4s
H 9887  5109                      17.8865652    9.02530  49.5%  52.5    5s
H 9892  4674                      17.2155061    9.02530  47.6%  52.5    5s
H 9895  4086                      16.2091966    9.02530  44.3%  52.6    5s
H 9906  3297                      15.0977616    9.02530  40.2%  52.6    5s
H10162  2438                      13.1214414    9.20936  29.8%  53.8    6s
H16088  1858                      12.4801925   10.92051  12.5%  73.6    9s
 17192  1847   12.41016   29  214   12.48019   11.09022  11.1%  75.6   10s
H22996  1240                      12.0096455   11.73127  2.32%  80.3   11s

Cutting planes:
  Gomory: 8
  Cover: 19
  MIR: 22
  Flow cover: 91
  Inf proof: 35
  Zero half: 4
  RLT: 212

Explored 25062 nodes (2283630 simplex iterations) in 11.62 seconds (18.27 work units)
Thread count was 32 (of 32 available processors)

Solution count 8: 12.0096 12.4802 13.1214 ... 18.3744

Optimal solution found (tolerance 1.00e-04)
Best objective 1.200964554654e+01, best bound 1.200964554654e+01, gap 0.0000%
```

Map:



System Specifications

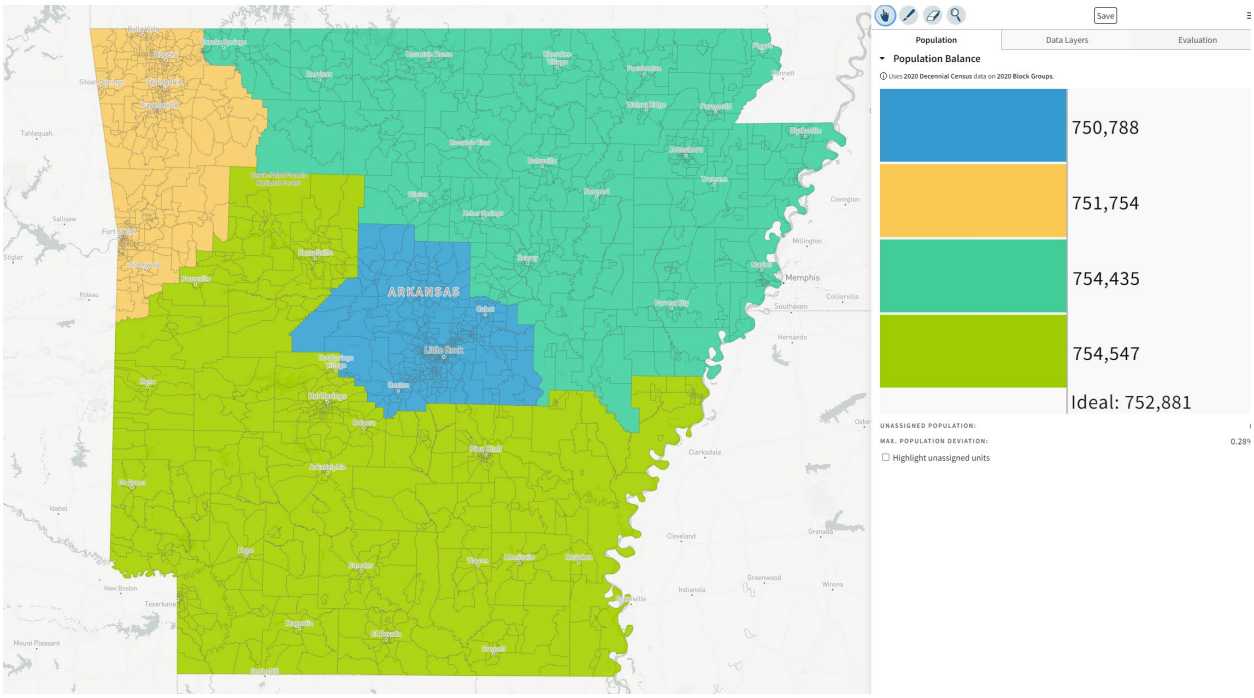
All models have been run on this system:

```
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (win64)

CPU model: 13th Gen Intel(R) Core(TM) i9-13900, instruction set [SSE2|AVX|AVX2]
Thread count: 24 physical cores, 32 logical processors, using up to 32 threads
```

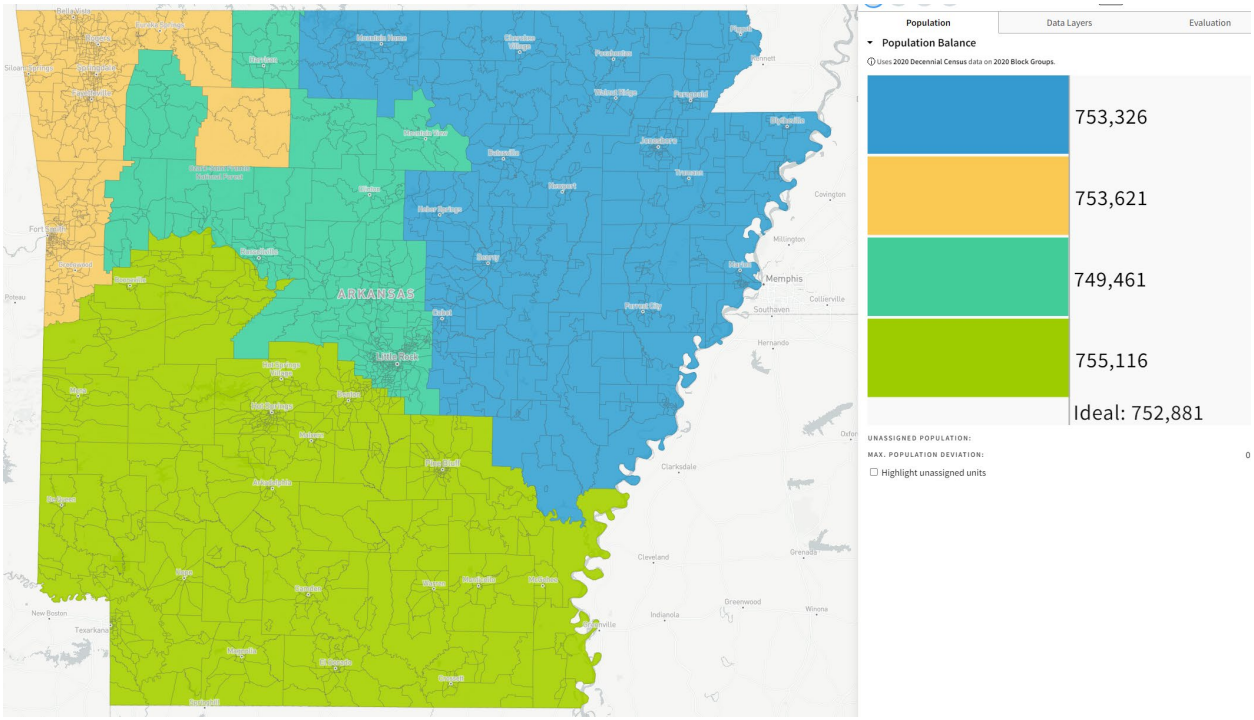
Maps and pictures

Min Perimeter and Min Cut edge

https://districtr.org/plan/214548



Min Moment of Inertia and
https://districtr.org/plan/126890



Evaluation Plans

Two new district plans for Arkansas were created. The first plan employed minimum cutting edges, resulting in a maximum population deviation of 0.28%. Process two utilized the minimum moment of inertia, determining a maximum population deviation of 0.45%. Process one is recommended due to its compliance with all federal and state criteria.

Conclusion

Upon reviewing federal and state redistricting criteria, data was gathered to develop a linear program depicting potential districting plans for the state of Arkansas. The criteria outlined in the proposed districting plan were covered. Each district was determined to encompass a population of 750,000 people. The population for each district, along with the counties they cover, is as follows:

- District 1: Population of 751,754, including counties such as Franklin, Crawford, Benton, Madison, Sebastian, and Washington.

- District 2: Population of 754,435, covering counties including Jackson, Clay, Baxter, Boone, St. Francis, Sharp, Greene, Woodruff, White, Lee, Crittenden, Marion, Prairie, Lawrence, Poinsett, Stone, Independence, Fulton, Carroll, Van Buren, Searcy, Randolph, Izard, Craighead, Cleburne, Monroe, Mississippi, Newton, and Cross.

- District 3: Population of 750,788, encompassing counties like Faulkner, Conway, Pulaski, Saline, Lonoke, and Perry.

- District 4: Population of 754,547, containing counties such as Little River, Ashley, Desha, Montgomery, Howard, Nevada, Grant, Dallas, Cleveland, Lafayette, Chicot, Pope, Bradley, Drew, Pike, Union, Hempstead, Polk, Clark, Logan, Miller, Arkansas, Johnson, Garland, Sevier, Jefferson, Lincoln, Scott, Hot Spring, Columbia, Ouachita, Yell, Calhoun, and Phillips.

The mapped districts maintain contiguity following the guidelines. The maximum population deviation of 0.28% from the minimum cut-edge method indicates that the population is nearly equivalent across districts.

References

Github: https://github.com/DrsAz77/Districting-OR.git

Codes: https://github.com/logandavis2518/IEM4013_2020RedistrictingProject.git

Data Source: https://github.com/AustinLBuchanan/Districting-Examples-2020

Redistricting Sources:

https://arkansasredistricting.org/about-the-process/redistricting-criteria-and-goals/

Voting Rights Act, 1965

Apportionment Clause, Article, Section 2, U.S. Constitution