Oklahoma State University

IEM 4013-Operations Research

Final Project:

Arkansas State Voting Redistricting

SPRING 2025

Dre Johnson, Sam Boozer, Byron Cave

Executive Summary

In the United States maintaining proper voter redistricting on the state level has been a reoccurring issue country wide. As this country was built on the essence of democratic beliefs and the post-civil rights movement our government prioritizes taking all bias out of districting and trying to achieve the most optimal split of votes. Voting Integrity is extremely important; our group will solve this problem for the state of Arkansas using the population as of 2020. This Report will outline the criteria for our districting, breakdown our math models, digress code, and give a result that can be understood by legislators.

Introduction

Arkansas redistricting is a decennial process initiated after the U.S. Census to ensure districts reflect population shifts and maintain roughly equal populations, a constitutional mandate that keeps our representation fair and balanced. The population of states is every changing, but it is still a focus of both legislators and mathematicians to try and gain the most “optimal” voting districts within their state for every election as this fair and democratic style of representation is the backbone of our society. The main issues concerning this problem will be using models to represent the detailed criteria in the state law of Arkansas, and making our conclusions and models make sense. Trying different approaches, we will find solutions within the State approved range and then suggest one.

Criteria

The most important criteria is that congressional districts have to be near equal in population according to the supreme court justice Hugo Black "as nearly as practicable one man's vote in a congressional election is to be worth as much as another's” from Wesberry v Sanders and from Baker v Carr we get “one person one vote” which reinforced the earlier Wesberry v Sanders causing most states to redistrict. Another criterion involves contiguity which means districts must be contiguous

without any disconnected locations so no “islands". The next criteria is compliance with the voting rights act of 1995 which state that “No voting qualification or prerequisite to voting or standard, practice, or procedure shall be imposed or applied by any State or political subdivision in a manner which results in a denial or abridgement of the right of any citizen of the United States to vote on account of race or color”. The next criteria is compactness and “Minimizing Splitting Political Subdivisions”; compactness is the measure of a district that has smooth boundaries without irregularity of shape, and it is preferred to minimizing the splitting of counties, cities, and towns. This is the criteria outlined in the Arkansas State Board of Apportionment.

Problem statement

Arkansas’s congressional district lines, last drawn in the 2010 Census, no longer meet modern geographical and demographic standards. In this project, the Short Kings using the 2020 Census designed and implemented a partition of the state into four contiguous, compact districts of nearly equal population with a deviation ≤1%.Our solution identify major criteria that needs to be addressed in the project like “one person one vote", contiguous, compactness and compliance with the Voting Rights Act of 1995 and minimizing splitting political subdivisions while providing neat and precise data for readability. Optimal Voting districts is an ongoing issue and will continue to be, this model will not only solve the problem at hand but give a model to follow for future districting in Arkansas. This problem will be solved while still answering the format of this project and its rubric in common or easy to understand language.

Operations Research Model

Process 1: Minimum Perimeter

Sets: C: *Set of counties (nodes) 1, 2, ..., n*

J: *Set of districts 1, 2, ..., k*

E: *Set of edges 1, 2, ..., e*

Indices: i: *Represents a count*

j: *Represents a district*

u,v: *Counties that are being checked for contiguity*

Parameters: P_i: *Population of county i*

T_j: *Total population of district j*

L: *Lower bound maximum deviation*

U: *Upper bound maximum deviation*

n: *Total number of counties*

k: *Total number of districts*

e: *Total number of edges*

M: *n – k + 1 (Big-M parameter)*

P_{uv}: *Shared perimeter length between counties u and v*

Variables: $$x_{ij} = \begin{cases} 1 & \text{if county } i \text{ is part of district } j \\ 0 & \text{Otherwise} \end{cases}$$

$$y_{uv} = \begin{cases} 1 & \text{if there is a district boundary between counties } u \text{ and } v \\ 0 & \text{Otherwise} \end{cases}$$

$$r_{ij} = \begin{cases} 1 & \text{if county } i \text{ is the root of district } j \\ 0 & \text{Otherwise} \end{cases}$$

f_{ij}: Flow of population from county i to county j

In words	In math
Minimize the total perimeter of cut edges:	$\text{Minimize} \sum_{(u,v)\in E} P_{uv} \cdot y_{uv}$
Each county is assigned to a district	$\sum_{j\in J} x_{ij} = 1 \quad \forall i \in C$
Each district has a population at most U	$\sum_{i\in C} P_i \cdot x_{ij} \leq U \quad \forall j \in J$
Each district has a population at least L	$\sum_{i\in C} P_i \cdot x_{ij} \geq L \quad \forall j \in J$
An edge is cut if u is assigned to district j but v is not	$x_{uj} - x_{vj} \leq y_{uv} \quad \forall (u,v) \in E, \forall j \in J$
Each district should have one root	$\sum_{i\in C} r_{ij} = 1 \quad \forall j \in J$
If node i isn't assigned to district j, then it cannot be its root	$r_{ij} \leq x_{ij} \quad \forall i \in C, \forall j \in J$
Only send flow to a root	$\sum_{j\in N(i)} (f_{ji} - f_{ij}) \leq 1 - M \times \sum_{j\in J} r_{ij} \quad \forall i \in C$
Do not send flow across cut edges	$f_{ij} + f_{ji} \leq M \times (1 - y_{ij}) \quad \forall (i,j) \in E$

Process 2: Minimum moment of inertia

Sets: C: *Set of counties (nodes) 1, 2, ..., n*

J: *Set of districts 1, 2, ..., k*

E: *Set of edges 1, 2, ..., e*

Indices: i: *Represents a county*

j: *Represents a district*

u,v: *Nodes used to define edges and flow connections*

Parameters: P_i: *Population of county i*

L: *Lower bound maximum deviation*

U: *Upper bound maximum deviation*

n: *Total number of counties*

k: *Total number of districts*

e: *Total number of edges*

M: *n – k + 1 (Big-M parameter)*

$dist_{ij}$: *Distance between county i and county j*

Variables: $x_{ij} = \begin{cases} 1 & \text{if county } i \text{ is assigned to district } j \\ 0 & \text{Otherwise} \end{cases}$

f_{ij}: Flow of population from county i to county j

f_{ii}: Self-flow variable, defined as 0 to prevent self-flow

In words	In math
Minimize the moment of inertia, calculated as the weighted sum of squared distances between each county and its assigned district center	$\text{Minimize} \sum_{i \in C} \sum_{j \in J} dist_{ij}^2 \times P_i \times x_{ij}$
Each county is assigned to a district	$\sum_{j \in J} x_{ij} = 1 \quad \forall i \in C$
There should be exactly k district centers	$\sum_{j \in J} x_{jj} = k$
Population in each district is at most U	$\sum_{i \in C} P_i \times x_{ij} \leq U \cdot x_{jj} \quad \forall j \in J$
Population in each district is at least L	$\sum_{i \in C} P_i \times x_{ij} \geq L \cdot x_{jj} \quad \forall j \in J$
Coupling constraint	$x_{ij} \leq x_{jj} \quad \forall i, j \in C$
Flow constraint for connectivity	$\sum_{u \in N(i)} (f_{ui} - f_{ij}) = x_{ij} \quad \forall i \in C, \forall j \in J$
Flow capacity constraint	$\sum_{u \in N(i)} f_{ui} \leq M \times x_{ij} \quad \forall i \in C, \forall j \in J$
Node cannot receive its own flow & Flow non-negativity	$f_{ii} = 0 \quad \forall i \in C$ $f_{ij} \geq 0, \quad \forall (i, j) \in E$

Process 3: Minimum cut edges

Sets: C: *Set of counties (nodes) 1, 2, ..., n*

J: *Set of districts 1, 2, ..., k*

E: *Set of edges 1, 2, ..., e*

Indices: i: *Represents a count*

j: *Represents a district*

u,v: *Counties that are being checked for contiguity*

Parameters: P_i: *Population of county i*

T_j: *Total population of district j*

L: *Lower bound maximum deviation*

U: *Upper bound maximum deviation*

n: *Total number of counties*

k: *Total number of districts*

e: *Total number of edges*

M: *n – k + 1 (Big-M parameter)*

P_{uv}: *Shared perimeter length between counties u and v*

Variables:

$$x_{ij} = \begin{cases} 1 & \text{if county } i \text{ is part of district } j \\ 0 & \text{Otherwise} \end{cases}$$

$$y_{uv} = \begin{cases} 1 & \text{if there is a district boundary between counties } u \text{ and } v \\ 0 & \text{Otherwise} \end{cases}$$

$$r_{ij} = \begin{cases} 1 & \text{if county } i \text{ is the root of district } j \\ 0 & \text{Otherwise} \end{cases}$$

f_{ij}: Flow of population from county i to county j

In words	In math
Minimize the number of cut edges	$\text{Minimize} \sum_{(u,v)\in E} y_{uv}$
Each county is assigned to a district	$\sum_{j\in J} x_{ij} = 1 \quad \forall i \in C$
Each district must have one root node	$\sum_{i\in C} r_{ij} = 1 \quad \forall j \in J$
Population in each district is at most U	$\sum_{i\in C} P_i \times x_{ij} \leq U \quad \forall j \in J$
Population in each district is at least L	$\sum_{i\in C} P_i \times x_{ij} \geq L \quad \forall j \in J$
An edge is cut if u is assigned to district j but v is not	$x_{uj} - x_{vj} \leq y_{uv} \quad \forall (u,v) \in E, \forall j \in J$
If node i isn't assigned to district j, then it cannot be its root	$r_{ij} \leq x_{ij} \quad \forall i \in C, \forall j \in J$
Flow conservation for contiguity	$\sum_{j\in N(i)} (f_{ji} - f_{ij}) \geq 1 - M \times r_{ij} \quad \forall i \in C$
Prevent flow across cut edges	$f_{ij} + f_{ji} \leq M \times (1 - y_{ij}) \quad \forall (i,j) \in E$

Python / Gurobi Code

Process 1: Minimum Perimeter

In [1]:
```
import gerrychain
import geopandas as gpd
from gerrychain import Graph
import gurobipy as gp
from gurobipy import GRB
```

In [2]:
```
filepath = 'C:\\Users\\sambo\\OneDrive\\Desktop\\IEM4013 Redistricting Project\\'
filename= 'AR_county.json'

G = Graph.from_json(filepath + filename)
```

In [3]:
```
for node in G.nodes:
    G.nodes[node]['TOTPOP'] = G.nodes[node]['P0010001']
```

In [4]:
```
dev = 0.005

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)
```

```
Using L = 750999 and U = 754763 and k = 4
```

1. In the initial section of the minimizing perimeter code, the project data is imported and processed. The dataset is loaded using the Gerry Chain library, and each county's population is assigned based on the census data key 'P0010001'.

   Next, the deviation parameter is set to 0.5%, and the bounds L and U are calculated to ensure that the population of each district remains within the specified range. This calculation is based on the total population divided by the number of districts k, allowing for a slight variation to maintain contiguity and balance among districts.

   In this case, the calculated bounds are:

   - Lower Bound L = 750,999
   - Upper Bound U = 754,763

   This establishes the initial population constraints for the districts before proceeding with the optimization model.

In [5]:
```
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
```

```
Set parameter Username
Set parameter LicenseID to value 2663177
Academic license - for non-commercial use only - expires 2026-05-09
```

In [6]:
```
m.setObjective( gp.quicksum( G.edges[u,v]['shared_perim'] * y[u,v] for u,v in G.edges ), GRB.MINIMIZE )
```

In [7]:
```
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()
```

2. This segment defines the model, variables, objective function (minimizing shared perimeter), and key constraints (assignment, population bounds, cut edges).

In [8]:
```
# 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 )
```

In [9]:
```
# 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()
```

3. This segment defines root nodes, flow variables, and Big-M constraints to maintain district connectivity and prevent flow across cut edges.

```
In [10]: m.optimize()
```

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (win64 - Windows 11.0 (26100.2))

CPU model: Intel(R) Core(TM) Ultra 7 155H, instruction set [SSE2|AVX|AVX2]
Thread count: 16 physical cores, 22 logical processors, using up to 22 threads

Optimize a model with 1422 rows, 1176 columns and 5748 nonzeros
Model fingerprint: 0xeffaf14e
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: 0.02s
Presolved: 1033 rows, 838 columns, 4255 nonzeros
Variable types: 382 continuous, 456 integer (456 binary)

Root relaxation: objective 6.028589e+00, 1041 iterations, 0.09 seconds (0.04 work units)

Cutting planes:
  Gomory: 17
  Cover: 37
  MIR: 29
  StrongCG: 1
  Flow cover: 61
  Inf proof: 41
  Mod-K: 3
  RLT: 142
  BQP: 86

Explored 44461 nodes (4083961 simplex iterations) in 142.87 seconds (35.19 work units)
Thread count was 22 (of 22 available processors)

Solution count 8: 12.4802 12.4802 13.169 ... 15.6213

Optimal solution found (tolerance 1.00e-04)
Best objective 1.248019253386e+01, best bound 1.248019253386e+01, gap 0.0000%
```

4. In this segment. the model was successfully optimized using Gurobi, minimizing the perimeter of the cut edges based on the defined objective function. The optimal solution was found with an objective value of approximately 12.4802, indicating the minimized perimeter length of the district boundaries. The optimization process involved exploring 44,461 nodes and performing over 4 million simplex iterations,

completing in 142.87 seconds with zero optimality gap.

In [11]:
```
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("")
```

```
The number of cut edges is 12.480192533864251
District 0 has population 752826 and contains counties ['Faulkner', 'Conway', 'Pulaski', 'Pope', 'Logan', 'Carroll', 'Van Bur
en', 'Johnson', 'Searcy', 'Scott', 'Perry', 'Yell', 'Newton']

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

District 2 has population 753942 and contains counties ['Jackson', 'Clay', 'Baxter', 'Boone', 'St. Francis', 'Sharp', 'Green
e', 'Woodruff', 'White', 'Crittenden', 'Marion', 'Prairie', 'Lawrence', 'Poinsett', 'Stone', 'Independence', 'Lonoke', 'Fulto
n', 'Randolph', 'Izard', 'Craighead', 'Cleburne', 'Mississippi', 'Cross']

District 3 has population 753002 and contains counties ['Little River', 'Ashley', 'Desha', 'Montgomery', 'Lee', 'Howard', 'Ne
vada', 'Grant', 'Dallas', 'Cleveland', 'Lafayette', 'Saline', 'Chicot', 'Bradley', 'Drew', 'Pike', 'Union', 'Hempstead', 'Pol
k', 'Clark', 'Miller', 'Arkansas', 'Garland', 'Sevier', 'Jefferson', 'Lincoln', 'Hot Spring', 'Columbia', 'Ouachita', 'Monro
e', 'Calhoun', 'Phillips']
```

5. In this segment, the code calculates and outputs the number of cut edges, which is approximately 12.48. It also displays the population and list of counties contained within each of the four districts formed.

In [12]:
```
import geopandas as gpd

filepath = 'C:\\Users\\sambo\\OneDrive\\Desktop\\IEM4013 Redistricting Project\\'
filename = 'AR_county.shp'
df = gpd.read_file( filepath + filename)
```

In [13]:
```
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()
```

6. This segment uses GeoPandas to map the district assignments visually. It creates a new 'assignment' column in the shapefile data and assigns each county to its respective district based on the optimization results. The map is then plotted to visually represent the districting solution.

Process 2: Minimum moment of inertia

In [1]:
```
from gerrychain import Graph
import gurobipy as gp
from gurobipy import GRB
import math
from geopy.distance import geodesic
import geopandas as gpd
```

In [2]:
```
#Set filepath and filename equal to the path/name of the data used respectively
filepath = 'C:\\Users\\sambo\\OneDrive\\Desktop\\IEM4013 Redistricting 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']
```

In [3]:
```
# 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
```

In [4]:
```
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)
```

```
Using L = 749117 and U = 756645 and k = 4
```

1. This segment imports necessary libraries, reads the county-level data, and initializes the graph with population and centroid coordinates for each node. It then creates a distance matrix using geodesic distances between county centroids and calculates population bounds (L and U) based on a 1% deviation for four districts.

In [5]:
```
m = gp.Model()

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

```
Set parameter Username
Set parameter LicenseID to value 2663177
Academic license - for non-commercial use only - expires 2026-05-09
```

In [6]:
```
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
```

In [7]:
```
# 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()
```

2. This segment defines the Gurobi model, establishes binary variables for county-district assignments, and sets the objective function to minimize the moment of inertia based on squared distances and population. It also includes constraints for county assignments, district centers, population limits, and coupling constraints ensuring that a county can only be a center if it is assigned as one.

In [8]:
```
# 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 !

# 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()
```

3. This segment introduces contiguity constraints by creating flow variables to model the flow of each district's assignment across arcs. It establishes constraints to ensure nodes consume and send flow

appropriately, restricts flow reception to assigned nodes only, and prevents nodes from receiving flow of their own type.

In [9]:
```
m.Params.MIPGap = 0.0

m.optimize()
```

```
Set parameter MIPGap to value 0
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (win64 - Windows 11.0 (26100.2))

CPU model: Intel(R) Core(TM) Ultra 7 155H, instruction set [SSE2|AVX|AVX2]
Thread count: 16 physical cores, 22 logical processors, using up to 22 threads

Non-default parameters:
MIPGap  0

Optimize a model with 17026 rows, 34425 columns and 124782 nonzeros
Model fingerprint: 0x1234a37a
Variable types: 28800 continuous, 5625 integer (5625 binary)
Coefficient statistics:
  Matrix range     [1e+00, 8e+05]
  Objective range  [2e+06, 2e+10]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 4e+00]
Warning: Model contains large objective coefficients
         Consider reformulating model or setting NumericFocus parameter
         to avoid numerical issues.
Presolve removed 161 rows and 398 columns
Presolve time: 0.83s
Presolved: 16865 rows, 34027 columns, 123984 nonzeros
Variable types: 28402 continuous, 5625 integer (5625 binary)
Deterministic concurrent LP optimizer: primal and dual simplex
Showing primal log only...

Concurrent spin time: 0.15s (can be avoided by choosing Method=3)

Solved with dual simplex

Root relaxation: objective 6.308102e+09, 2387 iterations, 1.07 seconds (0.49 work units)


Cutting planes:
  Gomory: 1
  Lift-and-project: 1
  Cover: 41
  Implied bound: 1
  MIR: 7
  StrongCG: 1
  Flow cover: 28
  GUB cover: 5
  Network: 1
  RLT: 9

Explored 93 nodes (11913 simplex iterations) in 15.22 seconds (4.76 work units)
Thread count was 22 (of 22 available processors)

Solution count 8: 6.8351e+09 6.86149e+09 6.88348e+09 ... 7.05688e+09

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

4. The optimization process was executed with a MIP gap set to 0, indicating a search for the exact optimal solution. The model involved 17,026 rows, 34,425 columns, and 124,782 non-zero elements. The optimal solution was found with an objective value of 6.8351×10^9, with a gap of 0.0000% after exploring 93 nodes and performing 11,913 simplex iterations in 15.22 seconds.

In [10]:
```
# 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("")
```

```
6835095216.710739
District 0 has population 753326 and contains counties ['Jackson', 'Clay', 'Baxter', 'St. Francis', 'Sharp', 'Greene', 'Woodr
uff', 'White', 'Lee', 'Crittenden', 'Marion', 'Prairie', 'Lawrence', 'Poinsett', 'Independence', 'Lonoke', 'Fulton', 'Arkansa
s', 'Randolph', 'Izard', 'Craighead', 'Cleburne', 'Monroe', 'Mississippi', 'Cross', 'Phillips']

District 1 has population 749461 and contains counties ['Franklin', 'Faulkner', 'Boone', 'Conway', 'Pulaski', 'Madison', 'Pop
e', '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', 'Cl
ark', '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']
```

5. This segment calculates and prints the objective value, which represents the minimized moment of inertia. It then identifies the districts based on the binary decision variables, calculates the population for each district, and lists the counties included in each district along with their populations.

In [11]:
```
filepath = 'C:\\Users\\sambo\\OneDrive\\Desktop\\IEM4013 Redistricting Project\\'
filename = 'AR_county.shp'
df = gpd.read_file( filepath + filename)
```

In [12]:
```
# 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()
```

6. This segment reads the shapefile data using GeoPandas and assigns each county to a specific district based on the previously determined labeling. It then maps these assignments to the dataframe and visualizes the district assignments on a map using the 'assignment' column.

Process 3: Minimum cut edges

In [1]:
```
#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
```

In [2]:
```
#Set filepath and filename equal to the path/name of the data used respectively
filepath = 'C:\\Users\\sambo\\OneDrive\\Desktop\\IEM4013 Redistricting Project\\'
filename= 'AR_county.json'
#Create a new Graph object G from the file
G = Graph.from_json(filepath + filename)
```

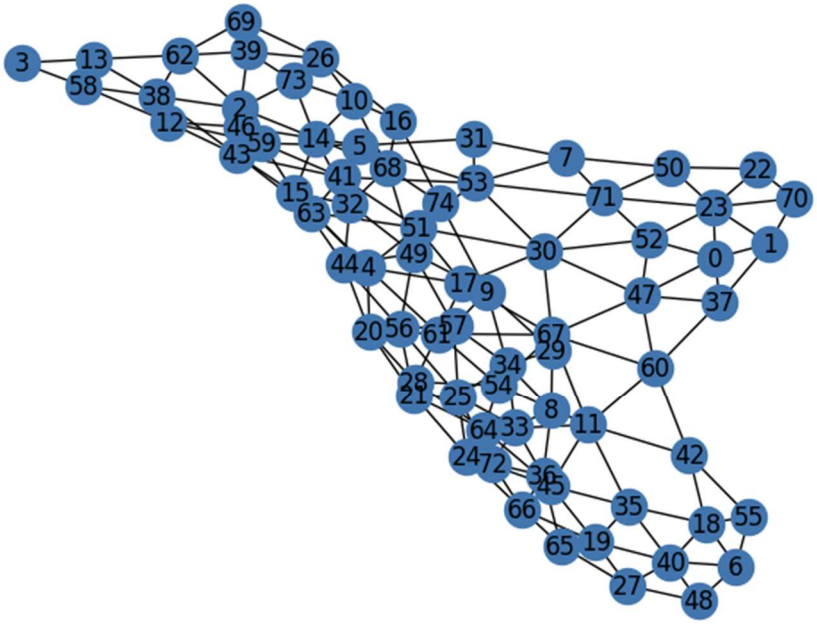
In [3]:
```
#Set each node in G to be equal to the population of their respective county
for node in G.nodes:
    G.nodes[node]['TOTPOP'] = G.nodes[node]['P0010001']
```

In [4]:
```
#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)
```

1. This segment imports necessary packages, loads the data from a JSON file to create a graph object, assigns population data to each county node, and prints each node's county name along with its 2020 population.

In [5]:

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



2. This segment visualizes the graph structure of Arkansas counties, with each node representing a county and labeled with its node identifier.

In [6]:
```
#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)
```

```
Using L = 749117 and U = 756645 and k = 4
```

In [7]:
```
#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 parameter Username
Set parameter LicenseID to value 2663177
Academic license - for non-commercial use only - expires 2026-05-09
```

In [8]:
```
#set objective to minimize cut edges
m.setObjective( gp.quicksum( y[u,v] for u,v in G.edges ), GRB.MINIMIZE )
```

In [9]:
```
# 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()
```

3. This segment defines the population deviation limits, creates the Gurobi model and binary variables for node and edge assignments, sets the objective to minimize cut edges, and establishes constraints for district assignments and population limits.

In [10]:
```
# 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)
```

In [11]:
```
# 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()
```

4. This segment adds root variables to indicate if a node is a district root, defines flow variables for edges, establishes the Big-M parameter, and implements constraints ensuring each district has a root, controlling flow only to assigned nodes, and restricting flow across cut edges.

```
In [12]: # sole IP model
         m.optimize()
```

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (win64 - Windows 11.0 (26100.2))

CPU model: Intel(R) Core(TM) Ultra 7 155H, instruction set [SSE2|AVX|AVX2]
Thread count: 16 physical cores, 22 logical processors, using up to 22 threads

Optimize a model with 1422 rows, 1176 columns and 5748 nonzeros
Model fingerprint: 0xa1a4a853
Variable types: 384 continuous, 792 integer (792 binary)
Coefficient statistics:
  Matrix range     [1e+00, 4e+05]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 8e+05]
Presolve time: 0.02s
Presolved: 1422 rows, 1176 columns, 5748 nonzeros
Variable types: 384 continuous, 792 integer (792 binary)

Root relaxation: objective 0.000000e+00, 579 iterations, 0.03 seconds (0.02 work units)


Cutting planes:
  Gomory: 22
  Cover: 3
  MIR: 2
  Flow cover: 11
  Inf proof: 1
  Zero half: 1
  RLT: 34
  BQP: 18

Explored 192599 nodes (19919899 simplex iterations) in 536.00 seconds (171.17 work units)
Thread count was 22 (of 22 available processors)

Solution count 6: 33 33 36 ... 63

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

5. The optimization process executed the integer programming model to minimize the number of cut edges across the districts. The solver identified the optimal solution with an objective value of 33, indicating 33 cut edges, with a computational time of approximately 536 seconds and a 0.0000% optimality gap, confirming the solution's optimality.

In [13]:

```
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("")
```

```
The number of cut edges is 33.0
District 0 has population 751754 and contains counties ['Franklin', 'Crawford', 'Benton', 'Madison', 'Sebastian', 'Washingto
n']

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', 'Clar
k', 'Logan', 'Miller', 'Arkansas', 'Johnson', 'Garland', 'Sevier', 'Jefferson', 'Lincoln', 'Scott', 'Hot Spring', 'Columbia',
'Ouachita', 'Yell', 'Calhoun', 'Phillips']

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

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

In [14]:

```
# Read Oklahoma county shapefile from "OK_county.shp"
filepath = 'C:\\Users\\sambo\\OneDrive\\Desktop\\IEM4013 Redistricting Project\\'
filename = 'AR_county.shp'
# Read geopandas dataframe from file
df = gpd.read_file( filepath + filename)
```

6. The code retrieves and prints the objective value (33 cut edges) and the composition of each district, displaying the population and the list of counties assigned to each district. Additionally, it prepares to read the county shapefile for visualization purposes.

In [15]:

```
# 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()
```

7. The code assigns each county to a district based on the optimization results and updates the data frame with these assignments. It then visualizes the district map by plotting the assignments, allowing for a visual representation of how counties are grouped into districts.

Experiments

The computational experiments were conducted using a system with the following specifications: Intel(R) Core(TM) Ultra 7 155H processor with a base speed of 3.80 GHz, 16 cores, and 22 logical processors. The system was equipped with 16 GB of SODIMM DDR5 RAM operating at 5600 MT/s and utilized an SSD (NVMe) as the primary storage device.

The optimization solver used for the experiments was Gurobi Optimizer version 12.0.2. The objective values and solution times for each of the three models are as follows:

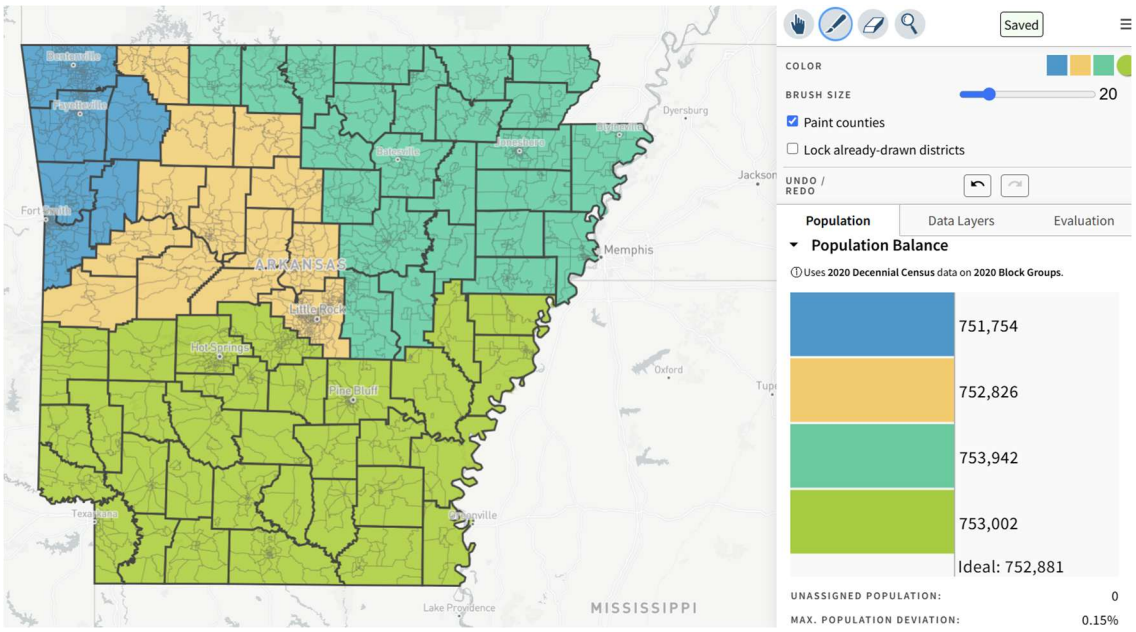
1. Minimizing Perimeter Model:
    - Objective Value: 12.48
    - Solution Time: 142.87 seconds
2. Minimizing Moment of Inertia Model:
    - Objective Value: 6.8351e+09
    - Solution Time: 15.22 seconds
3. Minimizing Cut Edges Model:
    - Objective Value: 33.0
    - Solution Time: 536.00 seconds

Each model was solved to optimality with a gap of 0.0000% at termination. There were no numerical issues or model infeasibilities reported, and the solver utilized a deterministic concurrent LP optimizer (primal and dual simplex) throughout the process.

Districting Plans & Maps

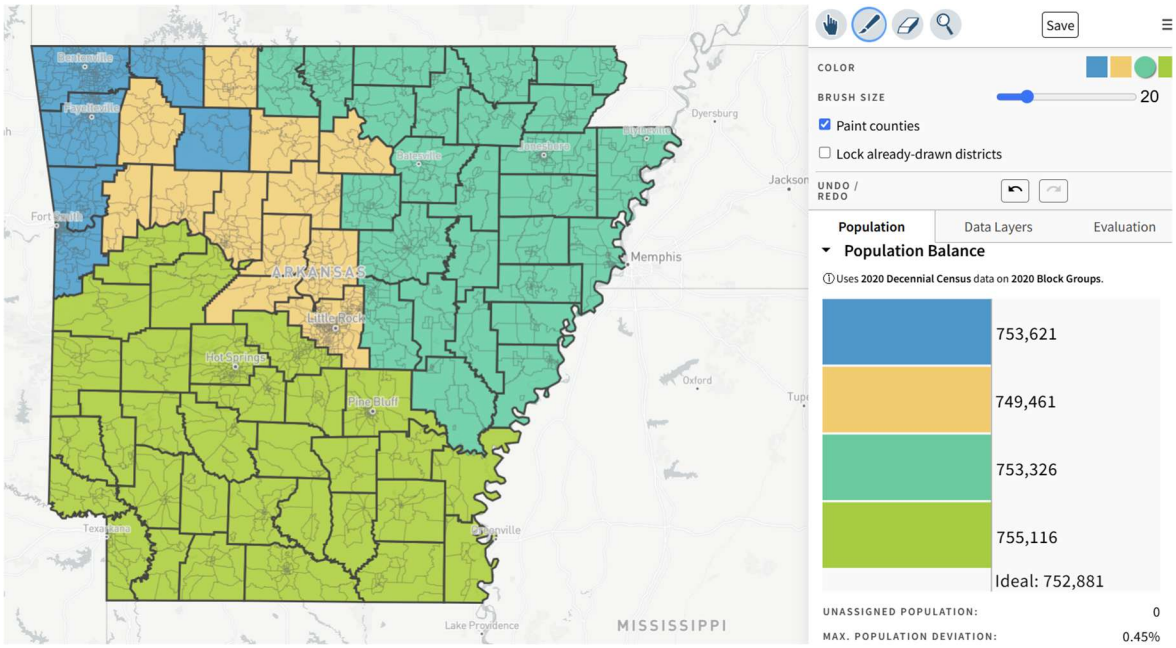
Process 1: Minimizing Perimeter

https://districtr.org/plan/298099



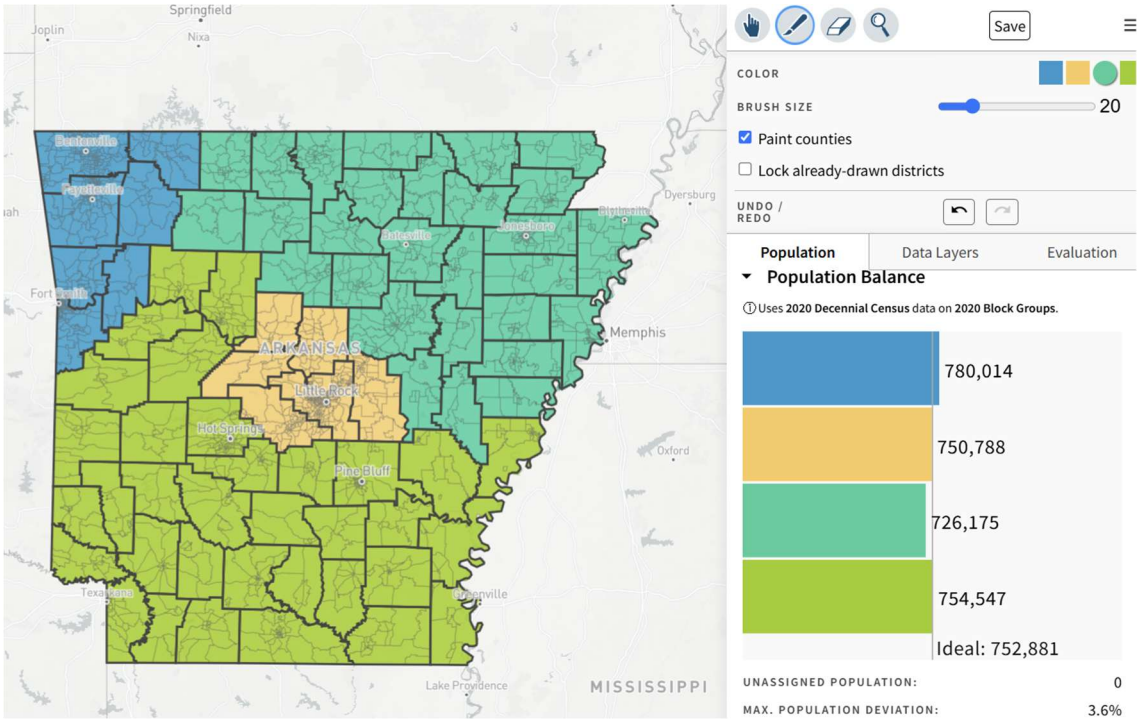
Process 2: Minimizing Moment of Inertia

https://districtr.org/plan/298100



Process 3: Minimizing Cut Edges

https://districtr.org/plan/298101



Evaluation of Plans

Three redistricting plans were developed for Arkansas, each adhering to established redistricting criteria, including population equality, contiguity, compactness, and compliance with the Voting Rights Act of 1965.

1. Minimizing Perimeter: This plan aimed to minimize the perimeter of district boundaries, thereby enhancing compactness. The maximum population deviation achieved was 0.15%, ensuring near-perfect population equality across districts in line with the "one person, one vote" principle established in *Wesberry v. Sanders* and *Baker v. Carr*. The plan successfully maintains contiguity and complies with the Voting Rights Act by avoiding racial gerrymandering. However, while perimeter reduction promotes compactness, it unintentionally split political subdivisions, potentially impacting community representation.

2. Minimizing Moment of Inertia: The objective here was to minimize the distance between the geographic center of each district and its constituent counties, thus optimizing compactness. The maximum population deviation was 0.45%, slightly higher than the previous plan but still within acceptable limits under federal guidelines. The plan maintains contiguity and meets Voting Rights Act requirements by ensuring that minority populations are not disenfranchised. Despite the focus on minimizing travel distances, this plan also split political subdivisions, reducing the overall compactness of district shapes.

3. Minimizing Cut Edges: This plan focused on minimizing the number of district boundary edges, aiming to keep political subdivisions intact as much as possible. The maximum population deviation achieved was 3.6%, the highest among the three plans. While the plan effectively minimizes splitting of political subdivisions, its higher population deviation raises concerns about unequal voter representation, potentially conflicting with *Wesberry v. Sanders* standards. However, it maintains contiguity and complies with the Voting Rights Act by avoiding racial gerrymandering.

Final recommendation: The Minimizing Perimeter plan is recommended as the optimal choice, as it achieves the lowest population deviation (0.15%), maintains compactness, and respects federal and state redistricting criteria, including the Voting Rights Act. Despite some potential splitting of political subdivisions, it provides

the most balanced approach to maintaining equal voter representation while adhering to contiguity and compactness standards.

Conclusion

In conclusion, the redistricting plan adheres to Arkansas's constitutional and statutory mandates by ensuring each congressional district contains as close to equal populations in accordance with the federal "one person, one vote "mandate, while also maintaining geographic contiguity and maximizing compactness. It as well complies with the Voting Rights Act of 1995 by keeping minority communities together to the greatest extent practicable, preserves county, municipal, and township boundaries only splitting when required to achieve population equality giving a map that is both good looking and legally sound. Using techniques and references given from IEM 4013 we were able to give an Arkansas State legislative a plausible map for redistricting and the future and model to help do this again moving forward. We have answered all given directives for this project and laid out this report to not only be read by engineers, citizens and legislators. The given maps and Results of standard deviation of 0.15% is far within the range of the state-based criteria and is stellar proposal for redistricting.

Citations

Arkansas Board of Apportionment. (2021, July 23). *The legal requirement to redistrict - Arkansas Board of Apportionment*. https://arkansasredistricting.org/about-the-process/the-legal-requirements-to-redistrict/

AustinLBuchanan. (n.d.). *GitHub - AustinLBuchanan/Districting-Examples-2020: Districting Examples using Python, Gurobi, NetworkX, and GeoPandas using 2020 Census data*. GitHub. https://github.com/austinlbuchanan/districting-examples-2020

History. (n.d.). The Rose Institute of State and Local Government. https://roseinstitute.org/redistricting/history/

GitHub Repository: https://github.com/sambooze1/Arkansas-Redistricting-Project