Final Report
IE 3301-001
Kansas
The Dream Team

Brandel Barrett, Brayden Crosnoe, Isaiah Soto-Mancillas, & Blain Thomas

Table of Contents

Executive Summary Letter

With the closure of the 2020 census data collection, the focus shifts to the allocation of congressional districts in response to population adjustments to ensure fair representation for each state. This report outlines the potential approach for states, specifically Kansas, to revise their districts considering population shifts. We must also consider specific guidelines that are mandated at both state and federal guidelines. Our aim is to present a starting point aligned with Kansas's and Washington D.C.'s criteria, offering guidelines for the legislature's redistricting plans according to The Apportionment Clause as well as the promotion of minority representation, as detailed in the Voting Rights Act of 1965.

The plan adheres to specific requirements tailored to the state's needs which includes maintaining compactness, contiguity, preserving communities of interest and prior district cores, safeguarding political subdivisions, and avoiding pairing of incumbents. Kansas, like many states, must specifically safeguard political subdivisions, communities of interest, and the core structures of previous districts. These constraints must align with the core principles such as mandating the assignment of all precincts to a district, ensuring connectedness of all counties within a district, preventing districts from overlapping, and maintaining approximate population parity across districts.

Using data collected from the 2020 Census and considering Kansas's redistricting criteria, this report proposes a potential map for redrawing congressional lines. Employing an Integer Programming Model, we successfully designed Kansas's congressional districts while adhering to both federal and state criteria based on population distribution. This offers a potential model for Kansas to adopt given that our map adheres to the guidelines, satisfies federal requirements, and upholds state constraints. Our research applied these constraints within the model, resulting in a map characterized by contiguity, compactness, and alignment with federal guidelines. This information aims to support Kansas officials in establishing a practical and effective starting point for any redistricting endeavors.

Introduction

Redistricting involves the crucial task of adjusting voting district boundaries to mirror population shifts. States experiencing population changes may need to redraw congressional district lines every decade based on census data. Upholding voters' rights and ensuring population equality are key considerations in this process.

Hand-solving redistricting can pose challenges, particularly in states permitting county divisions and emphasizing the preservation of communities of interest. Additionally, manually solving each state's redistricting may not guarantee the discovery of an optimal solution. In our project, we leveraged Python and Gurobi skills to generate a redistricting map tailored to Kansas. This approach allowed us to apply specific requirements effectively.

This report focuses on achieving a population deviation of less than one percent, a crucial benchmark for acceptable redistricting. Emphasis is placed on minimizing the distance from a district's center to other counties within it, promoting a more streamlined district map. Our approach prioritizes compactness and contiguity to maintain a clear and coherent congressional district layout.

Reducing population deviation and ensuring compactness are essential strategies for fairness and consistency in redrawing congressional districts. Considering our state's criteria, we've opted for the Hess model (moment-of-inertia) due to its alignment with contiguity and compactness constraints.

Criteria

Federal Criteria for Kansas

The federal requirements for every state including Kansas include equal population and minority representation. These requirements are obeyed by every state. The equal population clause requires all the districts to have equal populations within .5%. The minority clause prevents the redistricting to discriminate against races. The other federal requirements are adopted by only several states. Kansas adopted compactness, contiguous, preserved political subdivisions, preserved communities of interest, and avoided pairing incumbents. Compactness means that the state is encouraged to keep all constituents in a district as close as possible to one another. Contiguity implies that all the areas in a district are adjacent to one another. Districts in Kansas cannot be split in two. The preservation of political subdivisions requires districts to not cross counties, cities, or towns when drawing boundaries. The preservation of communities of interest encourages states to redistrict according to common interests in certain areas of the state. The last federal criterion is avoiding pairing incumbents. This avoids contests between incumbents.

State Criteria for Kansas

The state requirements for Kansas include contiguous, persevering political subdivisions, and communities of interest which have all been explained in the previous paragraph. The last state criteria is preserving the cores of prior districts. This maintains the previous redistricting maps as much as possible.

Problem Statement

Districts are drawn every 10 years due to the change in population and demographics of the state. States are required to acknowledge the changes and recreate them to follow the specific criteria established in the previous section. If the criteria are not met, then it could result in unfair advantages for politicians. Redistricting solves this problem by ensuring the district stays constitutional. So, in this project we used operations research to redistrict Kansas. With the proper constraints for the criteria, we can successfully draw a new demographic.

OR Model (In words)

1. Objective function-

The main objective of the model is to encourage contiguity for the districts of Kansas by minimizing the cut edges within the model.

2. Constraints of the model-

Population: Each district must have a population value within the lower(L) and upper(U) bounds [L, U].

3. Cutting edge-

If two adjacent counties are next to one another, then the edge is cut.

4. Contiguity Constraints-
    a. Flow consumption: This ensures that if a vertex (county) i is assigned to a district j (center), it consumes a unit of flow of type j. If vertex i is not assigned to center i, no flow of type j is consumed by i as a result.
    b. Non-Negative flow: This constraint prevents network edge flow that is negative.
    c. Self-Flow: We do not want any loops to occur within the model and so a node will not be able to receive its same flow.

OP Model (In math)

Sets/Indices

- i: Set of counties in Kansas, i= {1,2,3, 4, …., n}.
- j: Set of districts to be formed, j= {1,2,3,4}, for 4 total districts represented.
- V: This represents the set of all nodes or vertices in the network.
- N(i): This represents the set of neighboring nodes or vertices of node i.
- u and v: These are indices used to iterate over the sets of neighboring nodes in the constraints. Individual neighboring counties or districts within the set N(i) will be represented by u and v.

Parameters

- L: lower bound for each district.
- U: upper bound for each district.
- Pi: Population for all county *i*.
- *dij*: The total distance between counties *i* and j
- *K:* The number of districts made.

Variables

xij : Binary variable that equals 1 county i is assigned to district j.

yij: Binary variable that equals 1 if the edge between counties i and j is cut (i.e., they are in different districts).

Constraints

- For each county i, county i can only be assigned to one district j.
- If Y_{uv} equals 1, county u belongs to test district and county v does not.
- X_{ij} belongs to the set of numbers {0,1} for all I = {1, 2, ..., 104} and for all j= {1, 2, 3, 4}
- Total population j>= L
- Total population j<= U
- X*ij*, *Yuv*, ruv, fuv>= 0.

Python Code

Now this section will walk through the code and model used to redistrict the state of Kansas.

1. First, we had to download the necessary packages, and then find the required nodes and edges in the file path.

```
import gurobipy as gp
from gurobipy import GRB
import networkx as nx
from gerrychain import Graph
import geopandas as gpd
```

```
import json
from networkx.readwrite import json_graph

def read_graph_from_json(json_file):
    with open(json_file) as f:
        data = json.load(f)
    return json_graph.adjacency_graph(data)
```

```
#Read the KS county graph. Nodes and edges are included in .json file

filepath = 'C:\\Users\\Brayden\\DOR Files\\'
filename = 'COUNTY_KS.json'

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

2. Then we read the edges and nodes to ensure that they run and are correct

```
#Print nodes

print("The Kansas county graph has this many nodes total = ", G.number_of_nodes())
print("The Kansas county graph has these nodes = ", G.nodes)
```

```
The Kansas county graph has this many nodes total =  105
The Kansas county graph has these nodes =  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2
3, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104]
```

```
#Print edges

print("The Kansas county graph has this many edges total = ", G.number_of_edges())
print("The Kansas county graph has these edges = ", G.edges)
```

```
The Kansas county graph has this many edges total =  263
The Kansas county graph has these edges =  [(0, 36), (0, 80), (0, 47), (1, 67), (1, 91), (1, 98), (1, 59), (1, 104), (2, 82),
(2, 65), (2, 33), (2, 86), (3, 89), (3, 42), (3, 31), (3, 32), (3, 55), (3, 41), (4, 61), (4, 92), (4, 38), (4, 71), (4, 57),
(4, 53), (5, 43), (5, 76), (5, 94), (5, 90), (5, 40), (5, 54), (6, 28), (6, 85), (6, 29), (6, 63), (7, 40), (7, 54), (7, 78),
(8, 32), (8, 41), (9, 81), (9, 97), (9, 50), (10, 72), (10, 97), (10, 49), (10, 70), (10, 61), (10, 38), (11, 48), (11, 60), (1
1, 93), (11, 85), (11, 79), (12, 90), (12, 31), (12, 55), (12, 54), (13, 79), (13, 29), (13, 96), (13, 91), (14, 101), (14, 7
3), (14, 87), (14, 94), (14, 37), (14, 99), (15, 50), (15, 70), (15, 83), (15, 38), (15, 57), (15, 44), (15, 77), (16, 52), (1
6, 81), (16, 58), (16, 36), (17, 84), (17, 82), (17, 34), (17, 33), (17, 68), (18, 45), (18, 96), (18, 100), (18, 30), (18, 6
4), (19, 57), (19, 53), (19, 39), (19, 87), (19, 65), (19, 37), (20, 35), (20, 62), (21, 48), (21, 102), (21, 93), (21, 45), (2
2, 60), (22, 28), (23, 74), (23, 103), (23, 95), (23, 92), (23, 75), (23, 51), (23, 56), (24, 30), (24, 59), (24, 64), (24, 8
9), (24, 76), (24, 90), (24, 31), (25, 66), (25, 72), (25, 49), (26, 80), (26, 77), (26, 84), (26, 47), (26, 34), (26, 69), (2
7, 66), (27, 102), (27, 74), (28, 60), (28, 85), (29, 85), (29, 67), (30, 96), (30, 91), (30, 59), (30, 64), (31, 89), (31, 9
0), (31, 55), (32, 62), (32, 42), (32, 41), (33, 82), (33, 68), (34, 84), (34, 69), (34, 68), (34, 88), (35, 98), (35, 104), (3
5, 62), (36, 83), (36, 80), (37, 87), (37, 65), (37, 99), (37, 78), (37, 86), (38, 70), (38, 61), (38, 57), (39, 57), (39, 84),
(39, 82), (39, 65), (40, 94), (40, 99), (40, 54), (40, 78), (41, 55), (42, 104), (42, 89), (42, 62), (43, 100), (43, 64), (43,
73), (43, 76), (43, 94), (44, 57), (44, 77), (44, 84), (45, 93), (45, 95), (45, 96), (45, 75), (45, 100), (46, 69), (46, 88),
(47, 80), (47, 69), (48, 93), (49, 66), (49, 72), (49, 103), (49, 61), (50, 81), (50, 97), (50, 58), (50, 70), (51, 92), (51, 5
6), (51, 71), (51, 101), (52, 81), (53, 71), (53, 57), (53, 87), (54, 90), (56, 75), (56, 100), (56, 101), (56, 73), (57, 84),
(58, 81), (58, 83), (59, 91), (59, 104), (59, 89), (60, 85), (61, 103), (61, 92), (62, 104), (63, 67), (63, 98), (64, 100), (6
4, 76), (65, 82), (65, 86), (66, 74), (66, 103), (67, 91), (68, 88), (69, 88), (70, 97), (71, 92), (71, 101), (71, 87), (72, 9
7), (73, 101), (73, 94), (74, 102), (74, 103), (74, 95), (75, 95), (75, 100), (76, 90), (77, 83), (77, 80), (77, 84), (78, 99),
(78, 86), (79, 93), (79, 85), (79, 96), (80, 83), (82, 84), (87, 101), (89, 104), (91, 96), (92, 103), (93, 96), (94, 99), (95,
102)]
```

3. Name the attributes and read them to ensure they are correct for later.

```
#Print node # for each node, the county name, and Lat-Long coordinates

for node in G.nodes:
    county_name = G.nodes[node]['NAME10']

    G.nodes[node]['C_X'] = G.nodes[node]['INTPTLON10']
    G.nodes[node]['C_Y'] = G.nodes[node]['INTPTLAT10']

    print("Node", node, "represents", county_name, "County", ",", G.nodes[node]['C_Y'],")")
```

```
Node 0 represents Greeley County , +38.4804076 )
Node 1 represents Franklin County , +38.5580187 )
Node 2 represents Phillips County , +39.7845058 )
Node 3 represents Jackson County , +39.4109892 )
Node 4 represents Pawnee County , +38.1828730 )
Node 5 represents Clay County , +39.3449643 )
Node 6 represents Bourbon County , +37.8560997 )
Node 7 represents Republic County , +39.8289103 )
Node 8 represents Doniphan County , +39.7885021 )
Node 9 represents Seward County , +37.1805849 )
Node 10 represents Ford County , +37.6883653 )
Node 11 represents Montgomery County , +37.1895369 )
Node 12 represents Marshall County , +39.7827091 )
Node 13 represents Woodson County , +37.8884836 )
Node 14 represents Lincoln County , +39.0472763 )
Node 15 represents Finney County , +38.0498552 )
Node 16 represents Stanton County , +37.5659319 )
Node 17 represents Sheridan County , +39.3505425 )
Node 18 represents Chase County , +38.2985525 )
Node 19 represents Ellis County , +38.9145957 )
Node 20 represents Wyandotte County , +39.1153842 )
Node 21 represents Cowley County , +37.2345068 )
Node 22 represents Cherokee County , +37.1693922 )
Node 23 represents Reno County , +37.9481849 )
Node 24 represents Wabaunsee County , +38.9551537 )
Node 25 represents Comanche County , +37.1890712 )
Node 26 represents Logan County , +38.9132695 )
```

4. Next, create a dictionary to obtain and store the distance between each node.

```
# Obtaining and storing distance
from geopy.distance import geodesic

dist = dict()
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
```

5. We created the upper and lower bounds using a deviation of 1%. Then printed to ensure that it was correct. K equals the number of districts in Kansas which is four.

```
#Impose a deviation of 1%
deviation = 0.01

import math
k = 4         # number of districts
total_population = sum(G.nodes[node]['TOTPOP'] for node in G.nodes)

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

```
Using L = 709714 and U = 716845 and k = 4
```

6. We created the model to minimize the distance between nodes. The second cell of code has the objective function minimizing the distance.

```
# Model
m = gp.Model()

# Varibales
x = m.addVars(G.nodes, G.nodes, vtype=GRB.BINARY)  # this is creating a x[i,j] variable that is one when county i is
                                                   # assigned to district centered at j
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2024-09-10
```

```
# Minimize moment of interia
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 )
```

7. In this section, we added all the constraints. This makes sure that every node has a district, that the population is in-between the bounds, there are only four districts, and that each district chosen is in the center.

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

# Constraint for 4 districts
m.addConstr( gp.quicksum( x[j,j] for j in G.nodes ) == k )

# Districts are between U and L
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 )

# If i is assigneed to j, then j is the center
m.addConstrs( x[i,j] <= x[j,j] for i in G.nodes for j in G.nodes )

m.update()
```

8. Coded the contiguity constraints to keep all the districts connected and not split in two. The second cell shows that the gap is 0 and now we can move on.

```
# Contiguity constraints
DG = nx.DiGraph(G)

# Flow varible
f = m.addVars( DG.nodes, DG.edges, vtype=GRB.CONTINUOUS)
M = DG.number_of_nodes()-1

# Node j cannot recieve a flow of its own type
m.addConstrs( gp.quicksum( f[j,u,j] for u in DG.neighbors(j) ) == 0 for j in DG.nodes )

# Node i can receive flow of type j only if i is assigned to j
m.addConstrs( gp.quicksum( f[j,u,i] for u in DG.neighbors(i)) <= M * x[i,j] for i in DG.nodes for j in DG.nodes if i != j )

# If i is assigned to j, then i should consume one unit of j flow. Otherwise, i should consume no units of j flow.
m.addConstrs( gp.quicksum( f[j,u,i] - f[j,i,u] for u in DG.neighbors(i)) == x[i,j] for i in DG.nodes for j in DG.nodes if i != j

m.update()



# Solve model
m.Params.MIPGap = 0.0
m.optimize()
```

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (win64)

CPU model: AMD Ryzen 3 2200U with Radeon Vega Mobile Gfx, instruction set [SSE2|AVX|AVX2]
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads

Optimize a model with 55231 rows, 121485 columns and 427976 nonzeros
Model fingerprint: 0x5ae1d9b6
Variable types: 110460 continuous, 11025 integer (11025 binary)
Coefficient statistics:
  Matrix range     [1e+00, 7e+05]
  Objective range  [8e+05, 9e+10]
  Bounds range     [1e+00, 1e+00]
```

```
Explored 1 nodes (10802 simplex iterations) in 38.78 seconds (9.51 work units)
Thread count was 4 (of 4 available processors)

Solution count 1: 9.15957e+09

Optimal solution found (tolerance 0.00e+00)
Best objective 9.159574459000e+09, best bound 9.159574459000e+09, gap 0.0000%
```

9. The cell prints the objective function, and obtains the districts, counties, and populations.

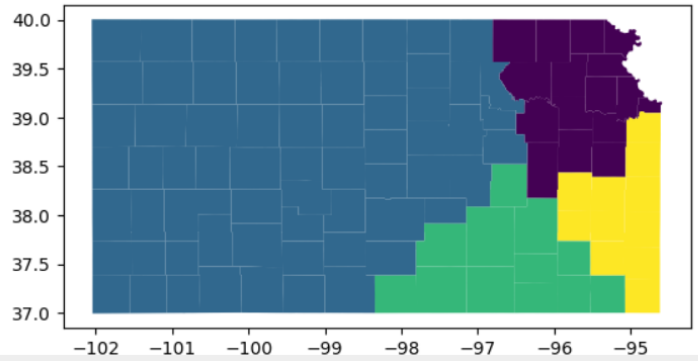
```
print("The moment of inertia objective is",m.objval)

# Retrieve the districts and their populations
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]["NAME10"] 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 information
for j in range(k):
    print("District",j,"has population",district_populations[j],"and contains counties",district_counties[j])
```

```
The moment of inertia objective is 9159574459.000452
District 0 has population 714862 and contains counties ['Franklin', 'Jackson', 'Doniphan', 'Marshall', 'Wyandotte', 'Wabaunse
e', 'Lyon', 'Pottawatomie', 'Atchison', 'Brown', 'Jefferson', 'Nemaha', 'Osage', 'Leavenworth', 'Shawnee', 'Douglas']
District 1 has population 712590 and contains counties ['Greeley', 'Phillips', 'Pawnee', 'Clay', 'Republic', 'Seward', 'Ford',
'Lincoln', 'Finney', 'Stanton', 'Sheridan', 'Ellis', 'Reno', 'Comanche', 'Logan', 'Norton', 'Thomas', 'Hamilton', 'Osborne', 'H
odgeman', 'Trego', 'Cloud', 'Dickinson', 'Lane', 'Cheyenne', 'Wallace', 'Kiowa', 'Haskell', 'Rice', 'Morton', 'Rush', 'Washingt
on', 'McPherson', 'Ness', 'Grant', 'Edwards', 'Morris', 'Rooks', 'Barber', 'Decatur', 'Sherman', 'Gray', 'Barton', 'Clark', 'Sa
line', 'Kingman', 'Harvey', 'Geary', 'Scott', 'Jewell', 'Wichita', 'Stevens', 'Graham', 'Kearny', 'Gove', 'Smith', 'Russell',
'Rawlins', 'Riley', 'Stafford', 'Ottawa', 'Meade', 'Mitchell', 'Marion', 'Ellsworth', 'Pratt']
District 2 has population 713239 and contains counties ['Montgomery', 'Chase', 'Cowley', 'Harper', 'Butler', 'Chautauqua', 'Lab
ette', 'Wilson', 'Elk', 'Sedgwick', 'Greenwood', 'Sumner']
District 3 has population 712427 and contains counties ['Bourbon', 'Woodson', 'Cherokee', 'Crawford', 'Allen', 'Johnson', 'Lin
n', 'Anderson', 'Neosho', 'Coffey', 'Miami']
```

10. Create a map of the final redistricting plan.

```
# Read Kansas county shapefile
filepath = 'C:\\Users\\Brayden\\DOR Files\\'
filename = 'KS_counties.shp'

# Read geopandas dataframe from file
df = gpd.read_file( filepath + filename )
```

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

# for each district j
for j in range(len(districts)):

    # for each node i in given district
    for i in districts[j]:

        # What is its GEOID?
        geoID = G.nodes[i]["GEOID10"]

        # Need to find this GEOID in the dataframe
        for u in G.nodes:
            if geoID == df['GEOID10'][u]: # Found it
                assignment[u] = j # Node u from the dataframe should be assigned to district j

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



Experiments

Used Dell computer with 8 GB of RAM and a 64-bit operating system

Processor: AMD Ryzen 3 2200U, 2500Mhz, 2 cores, 4 Logical processors

Optimization Solver: Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (win64)

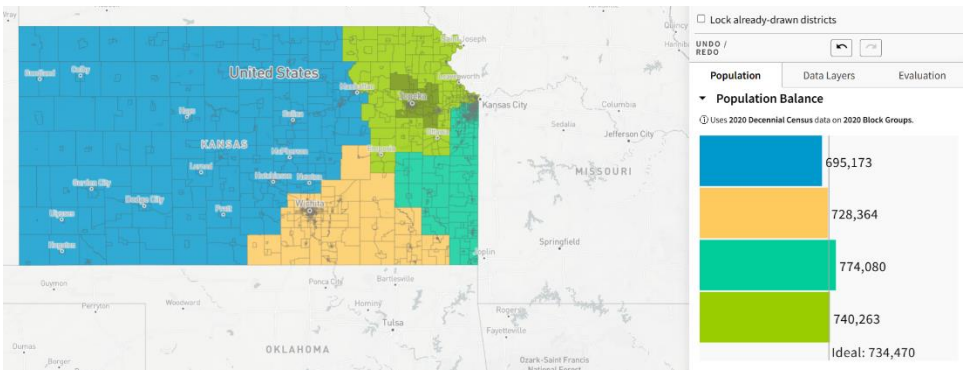
Time Required to Solve: 38.78 seconds to run 10802 simplex iterations

Objective Value: 9.159574459000e+09

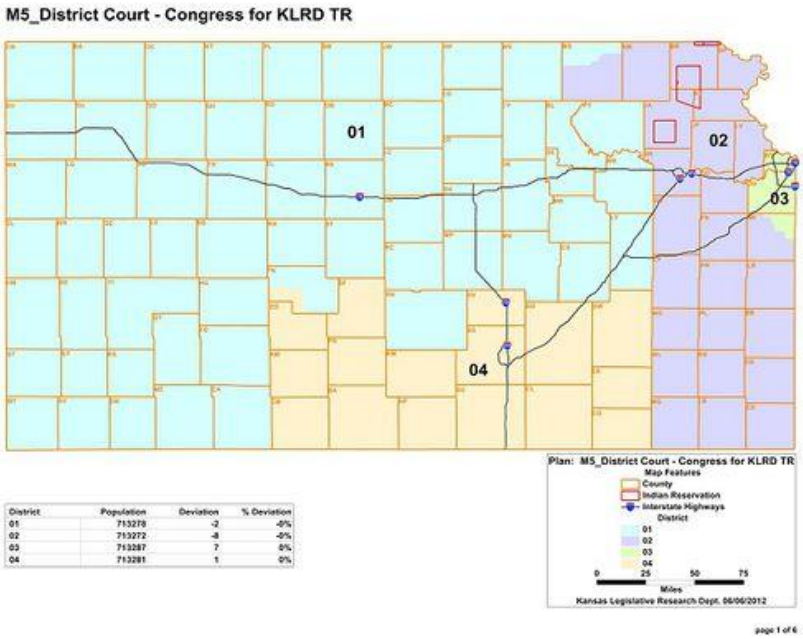
Plans and Maps

These are previous maps from past years our plan inserted into districtr.org. This shows the population of each district and the deviation in between each district. This allows us to compare our new redistricting plan to old plans and see if they are more optimal.

Our plan

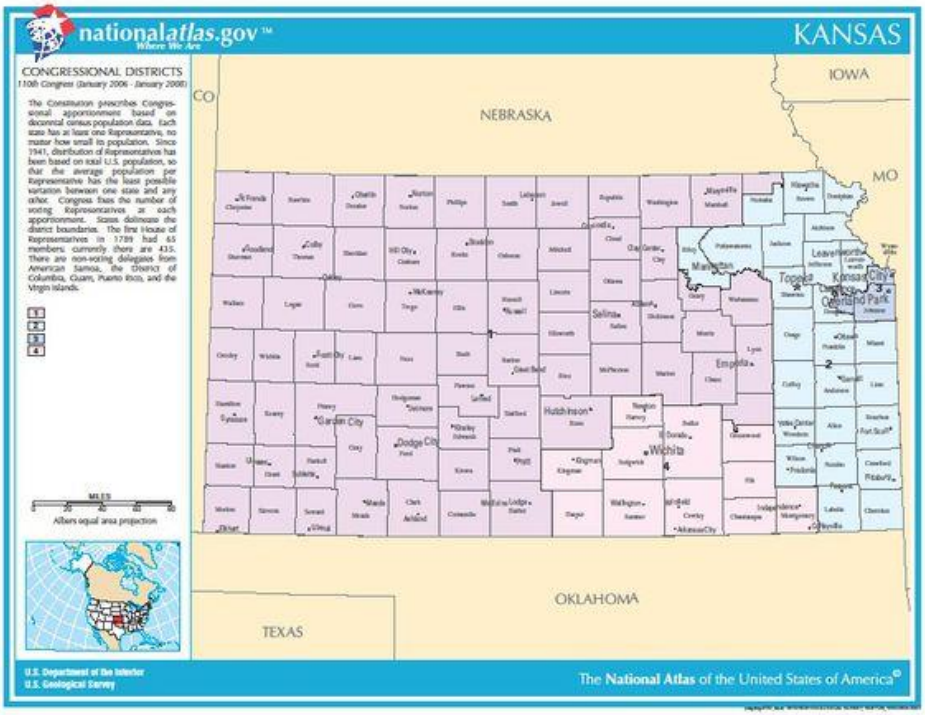


2010 Census



District	Population	Deviation	% Deviation
01	713278	-2	-0%
02	713272	-8	-0%
03	713287	7	0%
04	713281	1	0%

2000 Census



Evaluation

Our coding efforts successfully generated a contiguous and tightly knit map, showcasing a minuscule population deviation that fell within the allowable 1% deviation. However, our limitations in coding expertise and available data prevented us from incorporating crucial constraints such as preserving political subdivisions, communities of interest, and period district cores. While Kansas permits county splits, integrating this complexity took a considerable amount of time to resolve. Without these constraints, utilizing Gurobi for a solution would have demanded a significant more amount of time. Despite these limitations, we find contentment in the visual appeal and admirable population balance of our proposed map.

Conclusion

In summary, our proposed strategy fulfills both federal and state mandates by achieving contiguity, compactness, and equitable populations in districts, with deviations of less than 1%. We've successfully adhered to federal guidelines concerning population equality and minority representation. Additionally, we've met most state requirements, including the preservation of political subdivisions, communities of interest, and prior district cores. Our model also solved this problem much faster compared to doing it by hand without running the risk of having a simple mistake when working it out by hand.

Our plan effectively satisfies all state mandates while prioritizing adherence to both federal and state criteria, emphasizing compactness and contiguity. With confidence in meeting all necessary benchmarks, our team strongly recommends this redistricting proposal.