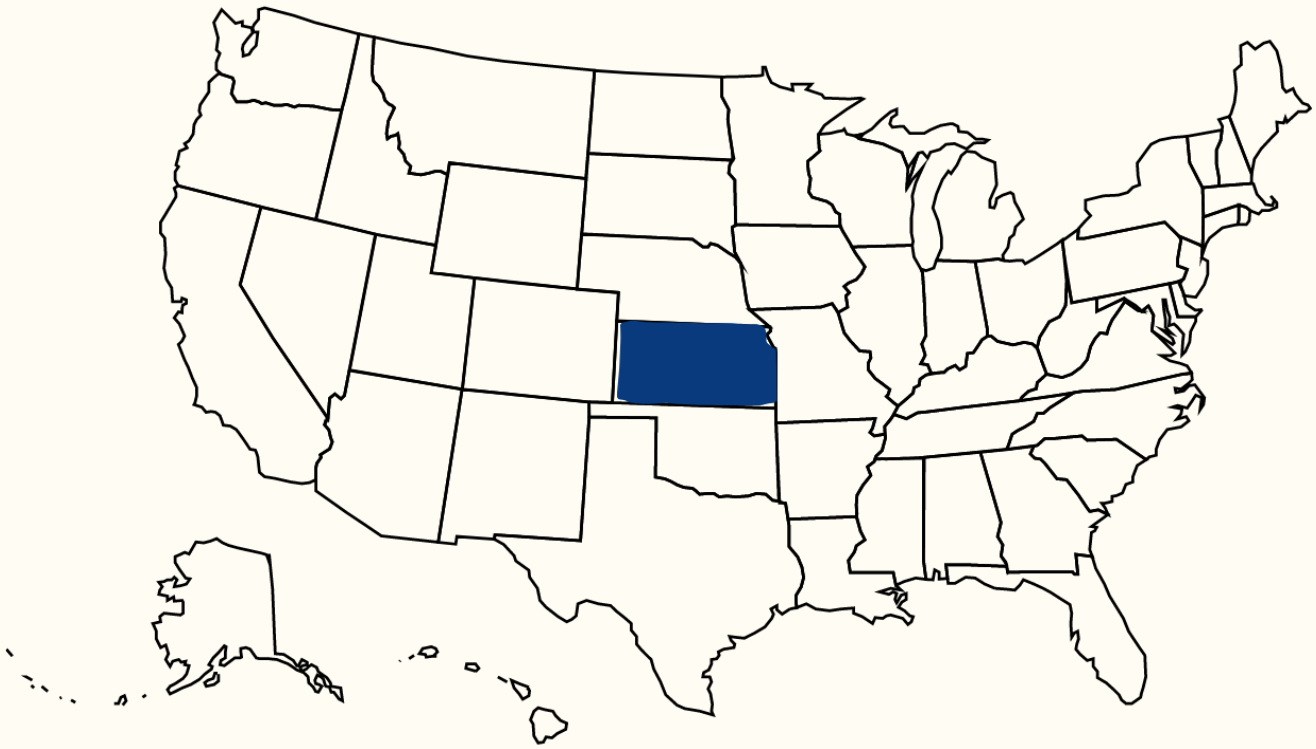
IEM 4013 Semester Project
Project
Kansas Redistricting



THE THREE AMIGOS
Andrew Caraway, Hope Goodwin,
& Kayla Hilger

Table of Contents

Executive Summary Letter

When redistricting in the United States, state legislatures and/or redistricting commissions are presented with a census conducted by the United States Government every ten years to provide each state with an accurate representation of its population. Specific guidelines have been set on the state and federal level that, in order for the districts to remain constitutional, must be followed. Federal guidelines state that redistricting must follow equal population requirements stated in The Apportionment Clause of Article I, Section 2 of the U.S. Constitution and minority representation requirements stated in Section 2 of the Voting Rights Act of 1965. Redistricting requirements, adopted by most states, must also maintain compactness, contiguity, preserve communities of interest, preserve cores of prior districts, preserve political subdivisions, and avoid pairing incumbents. Similar to most states, Kansas must preserve political subdivisions, communities of interest, and the cores of prior districts. These constraints also need to be met in congruence with the DRA, Dave's Redistricting App, meaning all precincts must be assigned to the district, all counties in a district must be connected, there can be no districts plotted in another district, and all districts must have a roughly equal population.

Based on the data collected in the 2020 Census and criteria for redistricting for the state of Kansas this report will propose a potential redistricting map. We have built an Integer Programming Model to redistrict Kansas's congressional lines while following the state and federal criteria, based on its population distribution. Our map was able to follow the DRA guidelines and follows all federal and state constraints. Through our research, we were able to apply constraints to the model that produced a map that is contiguous, compact, and follows the federal guidelines.

Introduction

Redistricting is a process of adjusting and redrawing the lines for voting districts in order to reflect the ever-changing population. This could mean having to redraw congressional district lines every ten years, according to the census, for states with a changing population. In order to properly represent a state's population, there are important requirements that must be followed regarding the protection of voters' rights and population equality.

In this report, we aim to ensure a population deviation of less than one present for the redistricting to be accepted. We will also focus on minimizing the total number of miles from the district's center to the other counties within the district. In order to maintain a neat and organized congressional district map, we have imposed compactness and contiguity constraints. Minimizing population deviation and maintaining compactness are recognized methods for establishing fairness and consistency when redrawing congressional districts. Based on our state's criteria we chose the Hess model (moment-of-inertia), as it best fits the contiguity and compactness constraints.

Criteria for Congressional Redistricting

Federal

Required - for every state

- Equal population requirements - The Apportionment Clause of Article I, Section 2, of the U.S. Constitution requires that all districts have nearly equal populations. Deviations should not exceed plus or minus 0.5 percent of the ideal population. Some states have their own deviations they must follow.
- Minority representation - Section 2 of the Voting Rights Act of 1965 prohibits the sectioning of districts that intentionally or inadvertently discriminate on the basis of race. This is to ensure that minority voters are not diluted.

Allowed (Adopted by most States)

- Avoiding pairing incumbents - avoid districts that would create contests between incumbents.
- Compactness - Having the distance between all parts of a district be minimum. Typically a circle, square, or hexagon is the most compact.
- Contiguity - All parts of the district are connected to the rest of the district.
- Preserve communities of interest - Places where residents have common political interests, like neighborhoods of a city, that do not coincide with the boundaries of political subdivisions.
- Preserve cores of prior districts - Maintaining the districts that were previously drawn, to the extent possible. This leads to continuity of representation.
- Preserve political subdivisions - This means that when drawing districts the city, town, or county boundaries are not crossed.

State

Required

The Kansas Constitution does not specifically mention anything about congressional redistricting, but the Kansas Legislative Reapportionment Committees have adopted the 2012 guidelines. These guidelines require:

- Required federal - Like all states, Kansas must comply with the constitutional equal population requirement and they must abide by the Voting Rights Act for minority representation (both of which are listed above under federally required criteria). All other criteria are subject to federal guidelines.
- Compactness - “Districts should be as compact as possible…”.

- Contiguous - Districts should be contiguous as possible while subjected to the constitutional equal population requirement. Water will act as land for contiguity.
- Preserve political subdivisions - “Whole counties should be in the same congressional district to the extent possible while still meeting guideline No. 2 above. Country loans are meaningful in Kansas and Kansas counties historically have been significant political units. Many officials are elected on a countywide basis, and political parties have been organized in county unites. Election of the Kansas member of Congress is a political process requiring political organizations which in Kansas are developed in county units. To a considerable degree most counties in Kansas are economic, social, and cultural units, or parts of a larger socioeconomic unit. These communities of interest should be considered during the creation of congressional districts.”
- Preserve communities of interest - “There should be recognition of communities of interest. Social, cultural, racial, ethnic, and economic interests to the population of the area, which are probable subjects of legislation should be considered.”
- Preserve the cores of prior districts - “The core of existing congressional districts should be preserved when considering the communities of interest to the extent possible.”

Problem Statement

Congressional districts are drawn to properly reflect the population and demographics of a state and its counties. As populations shift and demographics change it is important that redistricting represents these changes. When politicians take advantage of this process in order to redraw state lines in favor of their party and themselves it is called gerrymandering. Not only is gerrymandering unethical it is also unconstitutional. A model for redistricting must solve this problem of adaptation to changing populations while remaining unbiased and constitutional in its configuration.

In our constructed Integer Programming Model, we will redraw Kansas's congressional districts based on the demographics of its population and all criteria for redistricting on the federal and state level.

OR Model

Indices:

i is a county in Kansas

j is a county in Kansas assigned to the center of a district

Parameters:

L is Low population bound (L = 730798)

U is High population bound (U = 738143)

K is the number of districts (k = 4)

n is |V| is the number of land parcels

P_i is the population of county i

W_{ij} is Penalty for the moment-of-inertia ($w_{ij} = p_i d_{ij}^2$)

D_{ij} is the distance from centroid j

Sets:

V is the set of land parcels

N(i) is the set of neighboring counties

E is the set of edges

Variables:

X_{ij} = 1 if vertex i is assigned to vertex j, 0 otherwise

f^v_{ij} = the amount of flow, originating at district center v, that is sent across edge {i,j} (from I to j).

Objective:

Minimizes total distance traveled from county i to county j

Model:

$$\min \quad \Sigma_{i \in V} \Sigma_{j \in V} w_{ij} x_{ij} \qquad \text{(a)}$$

$$\text{st.} \quad \Sigma_{j \in V} x_{ij} = 1 \qquad \forall\ i \in V$$

$$\text{(b)} \quad \Sigma_{j \in V} x_{ij} = k$$

(c)

$$Lx_{jj} \leq \Sigma_{i \in V} p_i x_{ij} \leq U x_{jj} \qquad \forall\ j \in V$$

(d)

$$x_{ij} \leq x_{jj} \qquad \forall\ i,\ j \in V$$

$$\text{(e)} \quad \Sigma_{u \in N(i)} (f^j_{ui} - f^j_{iu}) = x_{ij} \qquad \forall\ i \in V \setminus \{j\},\ \forall\ j \in V$$

(f)

$$\Sigma_{u \in N(i)} f^j_{ui} \leq (n-1) x_{ij} \qquad \forall\ i \in V \setminus \{j\},\ \forall\ j \in V$$

(g)

$$\Sigma_{u \in N(i)} f^j_{ui} = 0 \qquad \forall\ j \in V$$

(h)

$$f^v_{ij},\ f^v_{ji} \geq 0 \qquad \forall\ \{i,j\} \in E,\ \forall\ v \in V$$

(i)

$$x_{ij} \in \{0,1\} \qquad \forall I,\ j \in V. \qquad \text{(j)}$$

(a): Minimizes total distance traveled from county i to county j

(b): Ensures that each vertex is assigned to one district

(c): Ensures that k districts are chosen

(d): Ensures that the population of each district lies between L and U

(e): Ensures if I is assigned to j, then j is the center

(f): Ensures that if vertex i is assigned to center j, then i consumes one unit of the flow of type j; otherwise, it consumes none.

(g): Ensures that vertex i can receive the flow of type j only if i is assigned to center j

(h): Prevents flow circulations

(i): Ensure non-negative flow

(j): Ensures x_{ij} is binary and an integer

Python/Gurobi Code

1. The first thing we did was import the majority of the packages we would use throughout the code.

```
#Import all the things

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

2. Next, we read the json file that stored the edges and nodes of the graph.

```
#Read the KS county graph. This is where all the edges and nodes are stored in the json file

filepath = 'C:\\Users\\14053\\Documents\\OPERATIONS RESEARCH\\OR Project Data KS\\'
filename = 'KS_county.json'

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

3. After extracting all the nodes and edges from the json file, we printed them all out to ensure the file was read correctly and everything was there.

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

4. The next step was naming some attributes of the data that will be used later in the code. Then printing those attributes to ensure the data is outputting correctly.

```
#For each node print the node #, the county name,its population, and Lat-Long coordinates

for node in G.nodes:
    county_name = G.nodes[node]['NAME20']
    county_population = G.nodes[node]['P0010001']
    G.nodes[node]['TOTPOP'] = county_population

    G.nodes[node]['C_X'] = G.nodes[node]['INTPTLON20']
    G.nodes[node]['C_Y'] = G.nodes[node]['INTPTLAT20']

    print("Node", node, "represents", county_name, "County, which had a population of", county_population, "and is centered at ("
```

```
Node 0 represents Montgomery County, which had a population of 31486 and is centered at ( -095.7424028 , +37.1895369 )
Node 1 represents Finney County, which had a population of 38470 and is centered at ( -100.7399674 , +38.0498075 )
Node 2 represents Kearny County, which had a population of 3983 and is centered at ( -101.3081363 , +37.9944614 )
Node 3 represents Ford County, which had a population of 34287 and is centered at ( -099.8847484 , +37.6884159 )
Node 4 represents Chase County, which had a population of 2572 and is centered at ( -096.5940289 , +38.2994749 )
Node 5 represents Lyon County, which had a population of 32179 and is centered at ( -096.1616407 , +38.4554034 )
Node 6 represents Atchison County, which had a population of 16348 and is centered at ( -095.3133975 , +39.5325441 )
Node 7 represents Barton County, which had a population of 25493 and is centered at ( -098.7678374 , +38.4812394 )
Node 8 represents Ellis County, which had a population of 28934 and is centered at ( -099.3173133 , +38.9145967 )
Node 9 represents Reno County, which had a population of 61898 and is centered at ( -098.0783463 , +37.9481849 )
Node 10 represents Leavenworth County, which had a population of 81881 and is centered at ( -095.0389770 , +39.1895108 )
Node 11 represents Johnson County, which had a population of 609863 and is centered at ( -094.8223295 , +38.8839065 )
Node 12 represents Wyandotte County, which had a population of 169245 and is centered at ( -094.7630866 , +39.1153842 )
Node 13 represents Miami County, which had a population of 34191 and is centered at ( -094.8329626 , +38.5667718 )
Node 14 represents Linn County, which had a population of 9591 and is centered at ( -094.8449321 , +38.2165494 )
Node 15 represents Douglas County, which had a population of 118785 and is centered at ( -095.2909475 , +38.8964168 )
Node 16 represents Seward County, which had a population of 21964 and is centered at ( -100.8552566 , +37.1809599 )
Node 17 represents Riley County, which had a population of 71959 and is centered at ( -096.7274889 , +39.2912114 )
Node 18 represents Pottawatomie County, which had a population of 25348 and is centered at ( -096.3371126 , +39.3821868 )
Node 19 represents Geary County, which had a population of 36739 and is centered at ( -096.7680996 , +39.0021387 )
Node 20 represents McPherson County, which had a population of 30223 and is centered at ( -097.6474893 , +38.3958120 )
Node 21 represents Franklin County, which had a population of 25996 and is centered at ( -095.2789618 , +38.5580187 )
stricting Problem.ipynb#  ts Labette County, which had a population of 20184 and is centered at ( -095.2974732 , +37.1914676 )
```

5. Next, we used the geopy.distance function to create a dictionary of the distances between nodes.

```
# getting and storing distances
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
```

6. We then implemented a 1% deviation on the lower and upper bounds that will be used in optimizing. We then printed the upper and lower bound to ensure the calculations were correct.

```
#Impose a 1% deviation
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 = 730798 and U = 738142 and k = 4
```

7. Next we created the model and variable that will be used to optimize the distance between nodes.

```
# creating the model
m = gp.Model()

# creating the 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
```

8. We then set the objective function to minimize the distance between nodes.

```
# The objective is to minimize the the 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 )
```

9. We then added constraints to ensure that each node is assigned to a district, that there are only four districts, that the district populations are between the upper and lower bound, and that if a district is chosen that it would be the center.

```
# adding a constraint that ensures 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)

# adding a constraint that ensures there should be 4 districts
m.addConstr( gp.quicksum( x[j,j] for j in G.nodes ) == k )

# adding a constraint that ensures the 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 )

# adding a coupling constraint stating 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()
```

10. We then added the continuity constraints to ensure that if a node is added to a district then it is connected in some capacity to another node in that district.

```
# adding contiguity constraints
DG = nx.DiGraph(G)

# adding 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 )

# Add constraints saying that 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()
```

11. We then ensured that the gap is equal to 0 and solved the model.

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

```
Set parameter MIPGap to value 0
Gurobi Optimizer version 9.5.0 build v9.5.0rc5 (win64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 33286 rows, 66255 columns and 241498 nonzeros
Model fingerprint: 0xe97be69c
Variable types: 55230 continuous, 11025 integer (11025 binary)
Coefficient statistics:
  Matrix range     [1e+00, 7e+05]
  Objective range  [8e+05, 1e+11]
  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 392 rows and 713 columns
Presolve time: 1.59s
Presolved: 32894 rows, 65542 columns, 240054 nonzeros
Variable types: 54524 continuous, 11018 integer (11018 binary)

Deterministic concurrent LP optimizer: primal and dual simplex
Showing first log only...


Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
   13568    4.5223115e+10   0.000000e+00   4.030077e+13      5s
Concurrent spin time: 0.00s

Solved with dual simplex

Root relaxation: objective 8.851388e+09, 5451 iterations, 4.12 seconds (2.46 work units)
```

12. Next, we printed the objective function and retrieved the districts, the counties in the districts, and the population of the districts. After retrieving the data, we then printed it.

```
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]["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])
```

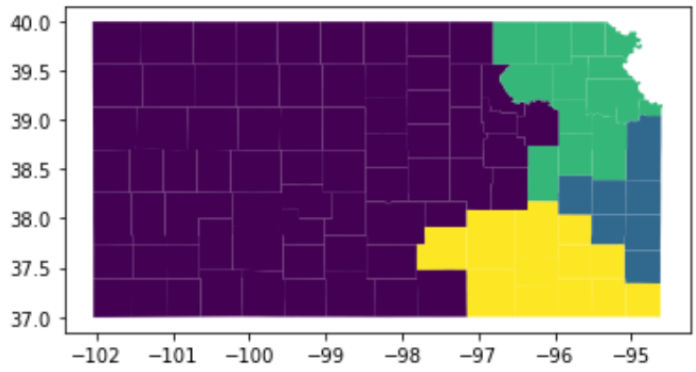
```
The moment of inertia objective is 9669715457.580788
District 0 has population 732489 and contains counties ['Finney', 'Kearny', 'Ford', 'Chase', 'Barton', 'Ellis', 'Reno', 'Sewar
d', 'Riley', 'Geary', 'McPherson', 'Ottawa', 'Saline', 'Wabaunsee', 'Harvey', 'Sumner', 'Clay', 'Lincoln', 'Smith', 'Morris',
'Barber', 'Logan', 'Rawlins', 'Thomas', 'Rice', 'Ness', 'Osborne', 'Clark', 'Haskell', 'Kingman', 'Stafford', 'Dickinson', 'Edw
ards', 'Sheridan', 'Kiowa', 'Harper', 'Washington', 'Norton', 'Ellsworth', 'Hodgeman', 'Meade', 'Wichita', 'Grant', 'Rooks', 'G
ray', 'Republic', 'Decatur', 'Phillips', 'Hamilton', 'Wallace', 'Pratt', 'Lane', 'Trego', 'Greeley', 'Marion', 'Rush', 'Stanto
n', 'Sherman', 'Cheyenne', 'Cloud', 'Russell', 'Jewell', 'Mitchell', 'Scott', 'Stevens', 'Comanche', 'Pawnee', 'Graham', 'Morto
n', 'Gove']
District 1 has population 735699 and contains counties ['Johnson', 'Miami', 'Linn', 'Crawford', 'Bourbon', 'Coffey', 'Anderso
n', 'Allen']
District 2 has population 733386 and contains counties ['Lyon', 'Atchison', 'Leavenworth', 'Wyandotte', 'Douglas', 'Pottawatomi
```

13. Next, to get an accurate map of the districts we just solved, we needed to read the map's shapefile.

```
# Read Kansas county shapefile
filepath = 'C:\\Users\\14053\\Documents\\OPERATIONS RESEARCH\\OR Project Data KS\\'
filename = 'Ks_county.shp'

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

14. The final step was to assign the nodes to districts based on the optimized objective. To do this we needed to retrieve the GEOID and use that in the assignment. After that, we created our model's Kansas map.

```
# Which 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 this district
    for i in districts[j]:

        # What is its GEOID?
        geoID = G.nodes[i]["GEOID20"]

        # Need to find this GEOID in the dataframe
        for u in G.nodes:
            if geoID == df['GEOID20'][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()
```



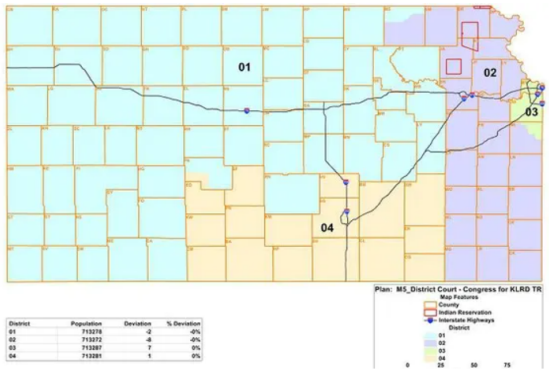
Experiments

The program was solved using an MSI computer with a RAM of 16.0 GB (15.8 usable), a 64-bit operating system, and a processor speed of 2.6 GHz. We used Gurobipy Optimizer version 9.5.0 v9.5.0rc5. The code ran for 52.13 seconds to run 19489 simplex iterations. The code's best objective value ended up being 9.669715457581e+09.

Plans and Maps

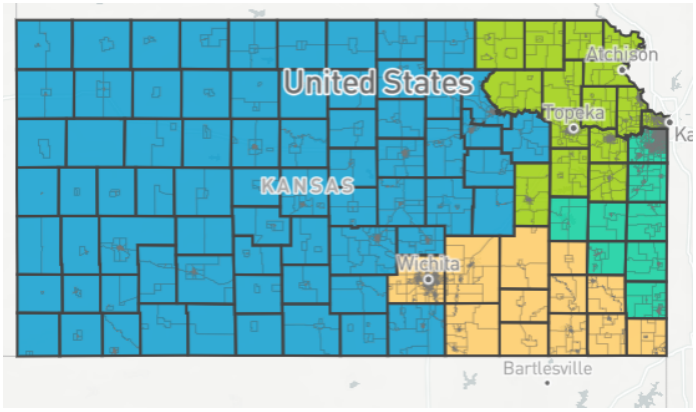
To better understand our redistricting plan we used district.org to recreate our plan and get the exact population distribution between the districts. There is the link to the plan: https://districtr.org/plan/126375.

2010:

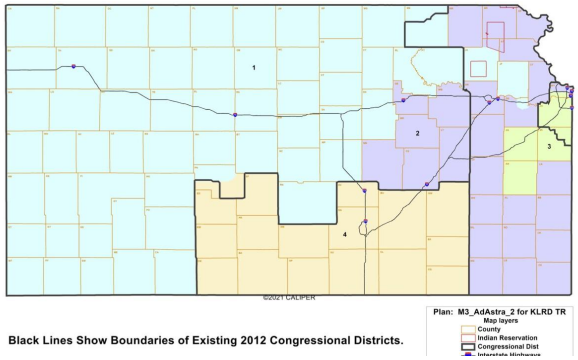


The map for Kansas' congressional districts as drawn by the courts during the 2012 redistricting process. *Courtesy Of Kansas Legislative Research Department*

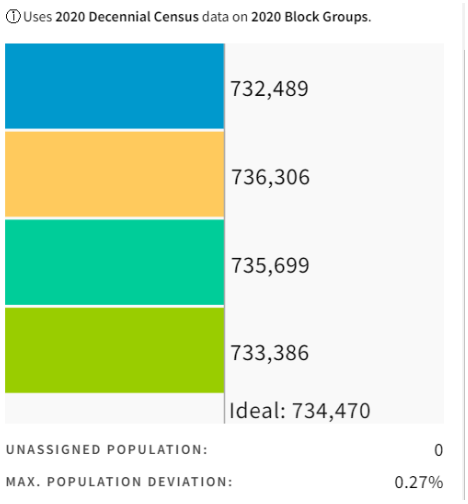
Our Proposed Plan:



2020:



Black Lines Show Boundaries of Existing 2012 Congressional Districts.



Evaluation of Plans

When observing our new districting plan, we covered the required federal criteria including compact and contiguous state criteria. We were able to construct a plan of 4 districts with a population deviation of 0.27% which falls within the allowable 1% deviation. It also covers the required state criteria that it is to be compact, contiguous, and preserve the cores of prior districts. Below is a small table showing the required criteria we were able to ensure our new plan covers. The limitation of our plan is the "preserve communities of interest" criteria. Though we did not intentionally code to ensure our plan met the "preserve communities of interest" criteria, our plan does meet that criteria.

Federal Required Criteria		State Required Criteria	
Equal population requirements	Yes	Compact	Yes
Minority Representation	Yes	Contiguous	Yes
		Preserve Political Subdivisions	Yes
		Preserve Communities of Interest	Yes
		Preserve the cores of prior districts	Yes

Conclusion

To conclude this report, our proposed plan meets both federal and state criteria of being contiguous, compact, and having equal populations in all districts and only deviating by less than 1%. We were able to follow all federally required criteria of equal population requirements and minority representation. We were also able to follow all state-required criteria for preserving political subdivisions, preserving communities of interest, and preserving cores of prior districts. The written code followed Hess' model to ensure we were meeting all federal and most of the state criteria. All of the state criteria was not ensured by the code, but our plan does meet all of the state criteria.

Our redistricting plan follows all federal and state criteria while maintaining compactness and contiguity restraints. Our team would recommend this redistricting proposal due to the plan meeting all federal and state requirements.

Resources

https://gerrymander.princeton.edu/reforms/KS

http://www.kslegislature.org/li_2012/b2011_12/committees/misc/ctte_h_redist_1_20120109_01_other.pdf

https://districtr.org