Kansas Redistricting

Ricky Cook, William Harrison, and Jackson Linson

Executive Summary:

As the 2020 census data collection has closed and is in the process of being finalized, the time has come to assign congressional districts according to adjusted population. In this report, we will lay out how states, particularly the state of Kansas, could go about revising their districts to accommodate the migration of people in the state. In order to create a districting plan that met all of Kansas's and Washington D.C. 's criteria, we paid special attention to certain restrictions imposed on the districts to give the legislature a reasonable and viable starting point for their redistricting plans. The requirements we considered are tailored toward the needs of the state, imposing compactness, county-preserving boundaries, and low population deviation. Compactness is the hardest one to follow by far, as it is more of a subjective measure than the others. However, by using the total distance along district boundaries as our objective value, we were able to ensure that this criteria was satisfied to the highest possible degree by grouping together counties with the shortest shared perimeter between separated counties.

After the code was run, we could look at the given solution to determine if we had adequately modeled the problem. In the end, the program found each of the four districts to be contiguous. They are also divided along county lines, which preserves the political subdivisions (another subjective criteria usually considered). Our final solution gave a population deviation of 0.45% which is less than the required 1%. Finally, we can see that the map of the districts appears visually reasonable, with a district representing different regions of the state without odd borders. The Kansas City metro, Eastern Kansas, the Wichita area, and Western Kansas all have their own district in our model's solution. We found this to be a logical distribution of the representatives in the state.

Since the solver created districts that conform to the given requirements, we are confident to say that our program can be used with the upcoming finalized census data to create a possible model for the state to adopt. This program could also be used in other states if new information is provided to the system. Hopefully this information could be used to help the officials in Kansas to complete a model that would be a useful and feasible beginning to their redistricting efforts.

Introduction:

Every ten years, a census is taken to gather an accurate population for every state. With this, the voting districts have to change so each district represents a fairly equal number of citizens. Solving this by hand could be hard for many states, especially states who allow counties to be split and require communities of interest to stay intact. Another issue with hand-solving the redistricting for each state is that the optimal solution may not be found. For our project, we used skills that we learned in Python and Gurobi, to produce a redistricting map for the state of Kansas based on the requirements we were able to impose.

Criteria:

For the state of Kansas, the state legislature requires that the districts are: contiguous, preserve political subdivisions, preserve communities of interest, and preserve cores of prior districts. While the Constitution says nothing on the topic of redistricting, legislation and court decisions have shaped the requirements over the years. One of these few requirements given is that each district must represent equal populations with very little leeway. Oftentimes, even a 1% deviation from one district to another could result in the overruling of the plan in court. The Supreme Court interpreted the equal protection clause of the Constitution to mean that racial groups should be given equal representation as well. Besides these, Washington D.C. has very little impact on the redistricting of the States.

IP Model:

The goal of our project is to minimize the distance of the cut edges for each district to ensure compactness. We will be keeping the counties whole and have the IP model as follows:

Sets:

C is the set of counties (0,1,2…,104)

Indices:

i is a county (in Kansas)

u and v are counties (in Kansas) that are being checked for contiguity

j is a district (in Kansas)

Parameters:

$population_i$ = the population of county i

$total\_population_j$ = the population of district j

k = the number of districts (k = 4)

L_j = Lower bound of district j for each district j

U_j = Upper bound of district j for each district j

Variables:

X_{ij} = {1 county i is a part of district j

{0 Otherwise

Y_{uv} = {1 there is a district boundary (cut edge) between county u and county v

{0 Otherwise

r_{uv} = {1 Contiguity variable stating whether county u and county v are adjacent

{0 Otherwise

f_{uv} = {1County u and county v are part of district j

{0 Otherwise

Objective:

Minimize the shared perimeter of district j

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}

$total\_population_j >= L$

$total\_population_j <= U$

$X_{ij}, Y_{uv}, r_{uv}, f_{uv} >= 0.$

Python/Gurobi Code:

Process 1

1. Importing Packages

To start off, we imported the packages we needed to solve the redistricting.

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

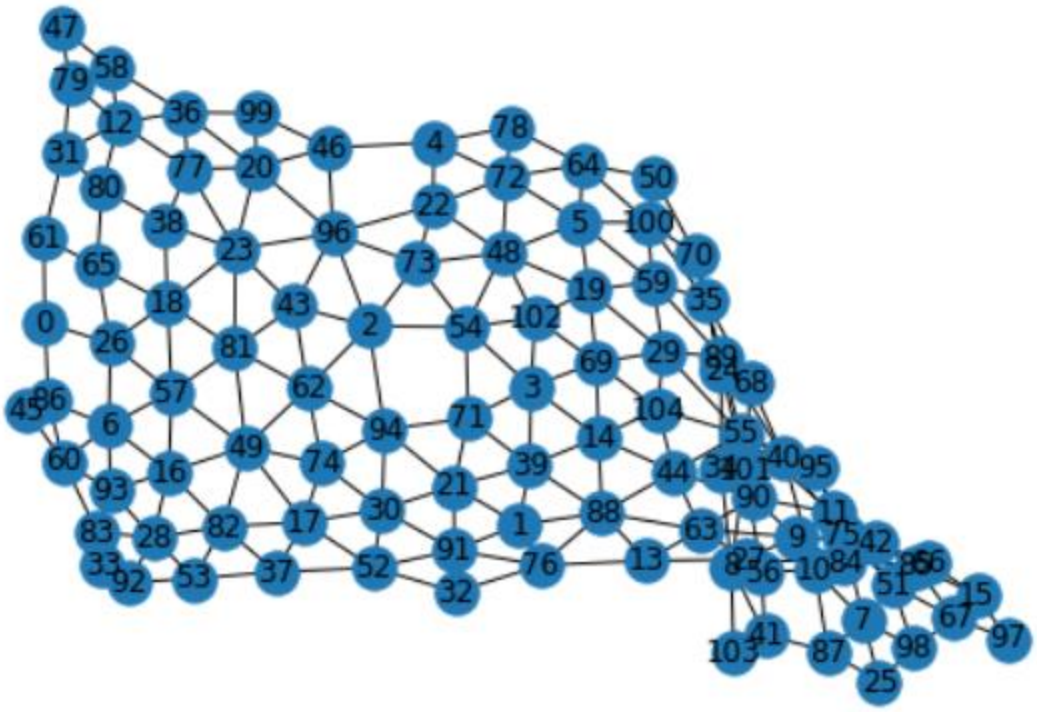
2. Finding Contiguity

After importing the packages, we called the KS.graph file which provided which counties are connected and which were not.

```
In [2]: filepath = 'C:/Users/wharr/OneDrive/Desktop/OR project/'
        filename = 'KS.graph.txt'

        G = nx.read_edgelist(filepath + filename,nodetype=int)
        print("The Kansas graph has",G.number_of_nodes(),"nodes")
        print("the Kansas graph has",G.number_of_edges(),"edges")
        print("Kansas graph has nodes",G.nodes )
        print("Kansas graph has edges",G.edges )
        nx.draw(G, with_labels=True)
```

This next picture shows a snippet of the outputs from this code.

```
The Kansas graph has 105 nodes
the Kansas graph has 263 edges
Kansas graph has nodes [0, 26, 61, 86, 1, 21, 39, 76, 88, 91, 2, 43, 54, 62, 73, 94, 9
6, 3, 14, 69, 71, 102, 4, 22, 46, 72, 78, 5, 19, 48, 59, 64, 100, 6, 16, 57, 60, 93, 7,
10, 25, 51, 75, 87, 98, 8, 34, 41, 56, 101, 103, 9, 11, 27, 42, 63, 84, 90, 40, 55, 85,
95, 12, 31, 36, 58, 77, 79, 80, 13, 44, 104, 15, 66, 67, 97, 28, 49, 82, 17, 30, 37, 5
2, 74, 18, 23, 38, 65, 81, 29, 20, 99, 24, 35, 68, 70, 53, 83, 92, 89, 32, 33, 45, 47,
50]
Kansas graph has edges [(0, 26), (0, 61), (0, 86), (26, 6), (26, 18), (26, 57), (26, 6
5), (61, 31), (61, 65), (86, 6), (86, 45), (86, 60), (1, 21), (1, 39), (1, 76), (1, 8
8), (1, 91), (21, 30), (21, 39), (21, 71), (21, 91), (21, 94), (39, 3), (39, 14), (39,
71), (39, 88), (76, 13), (76, 32), (76, 88), (76, 91), (88, 13), (88, 14), (88, 44), (8
8, 63), (91, 30), (91, 32), (91, 52), (2, 43), (2, 54), (2, 62), (2, 73), (2, 94), (2,
96), (43, 23), (43, 62), (43, 81), (43, 96), (54, 3), (54, 48), (54, 71), (54, 73), (5
```



From this we can find that county 0 touches counties 26, 61, and 86; county 26 touches 6, 18, 57, 65, and 0; etc. This information was crucial to build a contiguous map, that is, one where counties must touch to be in the same district. The only issue we have is that we don't know which counties the numbers represent. This leads us to our next portion of code.

3. Labeling nodes

With the KS_counties.json file we were able to interpret which nodes represented which counties. (Code and portion of output follows)

In [3]:
```
# Read Kansas county graph from the json file
filepath = 'C:/Users/wharr/OneDrive/Desktop/OR project/'
filename = 'KS_counties.json'
G = Graph.from_json(filepath+filename)
```

In [4]:
```
for node in G.nodes:
    name = G.nodes[node]['NAME10']
    population = G.nodes[node]['TOTPOP']
    print("Node", node,"represents", name,"County and its population in 2010",population)
```

```
Node 0 represents Greeley County and its population in 2010 1247
Node 1 represents Franklin County and its population in 2010 25992
Node 2 represents Phillips County and its population in 2010 5642
Node 3 represents Jackson County and its population in 2010 13462
Node 4 represents Pawnee County and its population in 2010 6973
Node 5 represents Clay County and its population in 2010 8535
Node 6 represents Bourbon County and its population in 2010 15173
Node 7 represents Republic County and its population in 2010 4980
Node 8 represents Doniphan County and its population in 2010 7945
Node 9 represents Seward County and its population in 2010 22952
Node 10 represents Ford County and its population in 2010 33848
Node 11 represents Montgomery County and its population in 2010 35471
Node 12 represents Marshall County and its population in 2010 10117
Node 13 represents Woodson County and its population in 2010 3309
```

4. Download shape of state and populations

Now that we have the counties labeled, we download the KS_counties.shp file so that we could create a map later with the actual shape of Kansas and its counties. We also imported the KS.population file to find the population of each county.

In [5]:
```
filepath = 'C:/Users/wharr/OneDrive/Desktop/OR Project/'
filename = 'KS_counties.shp'
df = gpd.read_file(filepath+filename)
```

In [6]:
```
# read the text file "KS.population" and store in the list called population
population = list()

# open the text file for reading
filepath = 'C:/Users/wharr/OneDrive/Desktop/OR project/'
filename = "KS.population"
file = open( filepath+filename ,"r")

# while the current line is not empty, read in a new county population
line = file.readline()

while line != "":
    # split the line into two "words":
    #    word[0]: the county's number
    #    word[1]: the county's population
    words = line.split()
    county_number = words[0]
    county_population = int(words[1]) # cast the string as type int

    # append to population list
    population.append(county_population)

    # read next line
    line = file.readline()

file.close()
print("population = ",population)
```

```
population =  [32787, 6373, 12660, 6497, 24132, 9656, 110826, 2234, 5724, 2597,
3977, 2695, 16512, 3853, 6970, 2519, 177934, 71115, 8601, 4437, 2882, 6091, 498
365, 6689, 4575, 1247, 25992, 5642, 13462, 6973, 8535, 15173, 4980, 7945, 2295
2, 33848, 35471, 10117, 3309, 3241, 36776, 2235, 2556, 2790, 28452, 157505, 363
11, 21603, 64511, 7053, 1891, 2756, 5799, 10178, 29180, 3107, 7829, 16295, 2160
7, 3037, 76227, 9656, 5923, 5181, 4861, 8102, 2961, 6010, 6006, 27674, 2215, 55
606, 7858, 34684, 34362, 4936, 3077, 9409, 6034, 39134, 13371, 33690, 21604, 16
924, 5671, 7900, 544179, 2690, 3858, 1916, 3001, 9533, 9984, 19126, 19754, 175
0, 65880, 2726, 1485, 3669, 2553, 4256, 10083, 3233, 3307]
```

5. Add decision variables, parameters, and constraints

In [9]:
```
deviation = 0.01

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)
```

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

We first established the population deviation to be less than 1 percent and then set the number of districts to four. We then found the total population and created variables for the lower and upper boundaries. After this, we made the model and added the x-variable which represents when a county is assigned to a district and a y-variable that represents an edge is cut.

In [11]:
```
# objective is to minimize the total perimeter of the districts
#  the boundary length between counties u and v is stored in G.edges[u,v]['shared_perim']
m.setObjective( gp.quicksum( G.edges[u,v]['shared_perim']*y[u,v] for u,v in G.edges ), GRB.MINIMIZE )
# add constraints saying that each county i is assigned to one district
m.addConstrs( gp.quicksum(x[i,j] for j in range(k)) == 1 for i in G.nodes)
# add constraints saying that each district has population at least L and at most U
m.addConstrs( gp.quicksum( G.nodes[i]['TOTPOP'] * x[i,j] for i in G.nodes) >= L for j in range(k) )
m.addConstrs( gp.quicksum( G.nodes[i]['TOTPOP'] * x[i,j] for i in G.nodes) <= U for j in range(k) )
# add constraints saying that edge {i,j} is cut if i is assigned to district v but j is not.
m.addConstrs( x[i,v] - x[j,v] <= y[i,j] for i,j in G.edges for v in range(k))
m.update()
```

In [12]:
```
# 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)
# 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
DG = nx.DiGraph(G) # directed version of G
f = m.addVars(DG.edges, vtype=GRB.CONTINUOUS)
```

In [13]:
```
M = G.number_of_nodes() - k + 1
# Each district j should have one root
m.addConstrs( gp.quicksum( r[i,j] for i in DG.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 DG.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[u,v] - f[v,u] for u in DG.neighbors(v) ) >= 1 - M * gp.quicksum( r[v,j] for j in range(k)) for v ir
# 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()
```

In [14]:
```
m.optimize()
```

Note: the rest of the cutoff line is “in G.nodes)”

We then set the objective function to minimize the total perimeter of each district to imply compact districts. We established constraints that each county must be assigned to exactly one district and that the population of the district must be within the lower and upper population bounds. Root and flow variables were then added to create the contiguity constraints in the map produced.

```
 5184116  5424     cutoff   34        10.46886   10.40985  0.56%  90.5 17571s
 5191464     0     cutoff   28        10.46886   10.46686  0.02%  90.4 17575s

Cutting planes:
  Gomory: 1
  Cover: 1
  MIR: 6
  Flow cover: 15
  RLT: 64

Explored 5191486 nodes (469580581 simplex iterations) in 17575.22 seconds

Thread count was 12 (of 12 available processors)

Solution count 10: 10.4689 10.4689 10.4689 ... 12.8717

Optimal solution found (tolerance 1.00e-04)
Best objective 1.046885786808e+01, best bound 1.046885701738e+01, gap 0.0000%
```

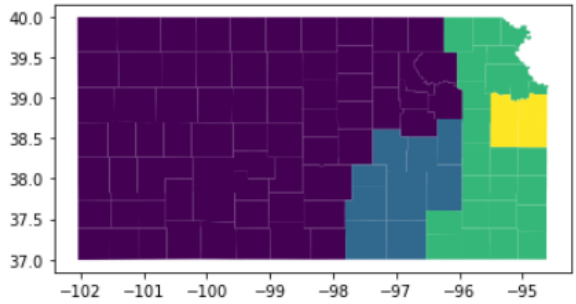
Our code found an optimal solution after 17,575 seconds.

6. Create map

```
In [15]: print("The number of cut edges is",m.objval)

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

The number of cut edges is 10.46885786808446
District 0 has population 710054 and contains counties ['Greeley', 'Phillips', 'Pawnee', 'Clay', 'Republic', 'Seward', 'Ford',
'Marshall', 'Lincoln', 'Finney', 'Stanton', 'Sheridan', 'Ellis', 'Reno', 'Wabaunsee', 'Comanche', 'Logan', 'Harper', 'Pottawato
mie', 'Norton', 'Thomas', 'Hamilton', 'Osborne', 'Hodgeman', 'Trego', 'Cloud', 'Dickinson', 'Lane', 'Cheyenne', 'Wallace', 'Kio
wa', 'Haskell', 'Rice', 'Morton', 'Rush', 'Washington', 'McPherson', 'Ness', 'Grant', 'Edwards', 'Morris', 'Rooks', 'Barber',
'Decatur', 'Sherman', 'Gray', 'Barton', 'Clark', 'Saline', 'Kingman', 'Geary', 'Scott', 'Jewell', 'Wichita', 'Stevens', 'Graha
m', 'Kearny', 'Gove', 'Smith', 'Russell', 'Rawlins', 'Riley', 'Stafford', 'Ottawa', 'Meade', 'Mitchell', 'Ellsworth', 'Pratt']
District 1 has population 715201 and contains counties ['Chase', 'Cowley', 'Lyon', 'Butler', 'Harvey', 'Sedgwick', 'Greenwood',
'Marion', 'Sumner']
District 2 has population 714079 and contains counties ['Jackson', 'Bourbon', 'Doniphan', 'Montgomery', 'Woodson', 'Wyandotte',
'Cherokee', 'Crawford', 'Allen', 'Atchison', 'Brown', 'Jefferson', 'Chautauqua', 'Nemaha', 'Osage', 'Labette', 'Leavenworth',
'Linn', 'Anderson', 'Wilson', 'Neosho', 'Shawnee', 'Coffey', 'Elk']
District 3 has population 713784 and contains counties ['Franklin', 'Johnson', 'Miami', 'Douglas']
```

First, we found the district to which each county was assigned.

In [17]:
```
# 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]["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()
```



The map was then created based on the assignments given from the previous code.

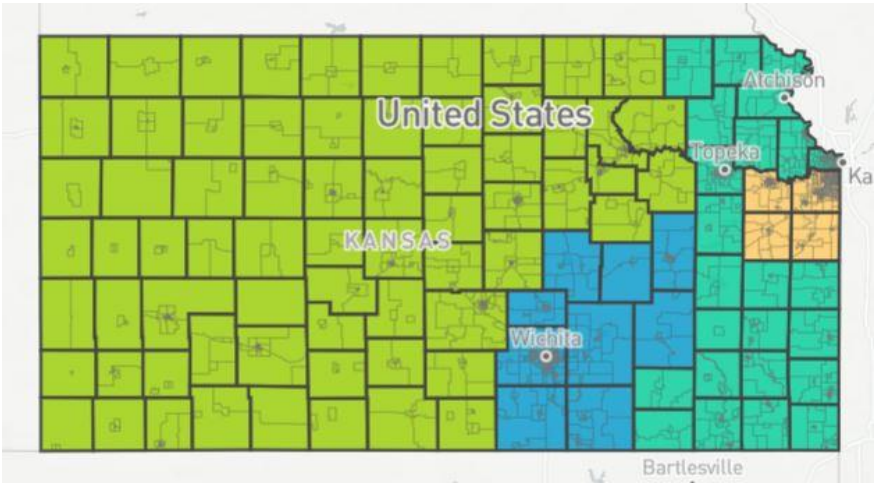
Process 2

Due to complications with the coding for this portion, we elected to stick with the first process. The second process would have used moment of inertia concepts to create districts with a majority of its population near the center of the district.

Experiments:

The program was solved on a Dell G5 15 with a RAM of 16 GB (15.8 usable), 64-bit operating system, and a processor speed of 2.20 GHz. With the Gurobi Optimizer version 9.1.1, it took 17,575 seconds to explore 5191486 nodes (469580581 simplex iterations) to reach an objective value of 1.046885786808e+01.

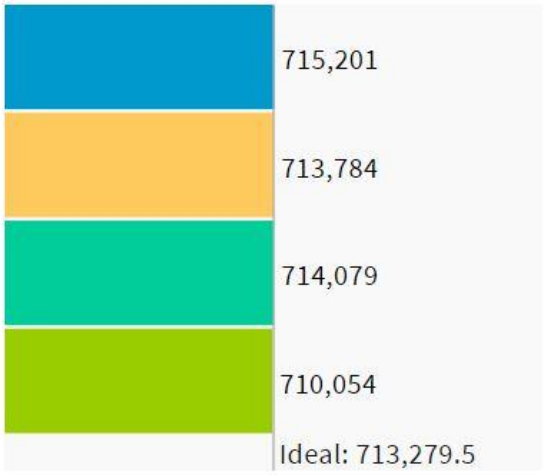
Plan and Map:



Population | Data Layers | Evaluation

▼ Population Balance

ⓘ Uses **2010 Decennial Census** data.



UNASSIGNED POPULATION:	0
MAX. POPULATION DEVIATION:	0.45%

https://districtr.org/plan/16435

Evaluation:

Our code produced a map that was contiguous, compact, and had a population deviation of only 0.45%. Due to the limitation of coding knowledge and data, we weren't able to impose other constraints like: preserving political subdivisions, preserving communities of interest, and preserving cores of period districts. Kansas also allows counties to be split but, considering it took almost 5 hours to solve with counties, it would have taken Gurobi a significant amount of time to solve without these constraints. We are overall satisfied with our proposed map's appearance and low population deviation.

Conclusion:

In conclusion, we recommend that: District 0 contains counties ['Greeley', 'Phillips', 'Pawnee', 'Clay', 'Republic', 'Seward', 'Ford', 'Marshall', 'Lincoln', 'Finney', 'Stanton', 'Sheridan', 'Ellis', 'Reno', 'Wabaunsee', 'Comanche', 'Logan', 'Harper', 'Pottawatomie', 'Norton', 'Thomas', 'Hamilton', 'Osborne', 'Hodgeman', 'Trego', 'Cloud', 'Dickinson', 'Lane', 'Cheyenne', 'Wallace', 'Kiowa', 'Haskell', 'Rice', 'Morton', 'Rush', 'Washington', 'McPherson', 'Ness', 'Grant', 'Edwards', 'Morris', 'Rooks', 'Barber', 'Decatur', 'Sherman', 'Gray', 'Barton', 'Clark', 'Saline', 'Kingman', 'Geary', 'Scott', 'Jewell', 'Wichita', 'Stevens', 'Graham', 'Kearny', 'Gove', 'Smith', 'Russell', 'Rawlins', 'Riley', 'Stafford', 'Ottawa', 'Meade', 'Mitchell', 'Ellsworth', 'Pratt'] which would give district 0 a population of 710,054, that district 1 contains counties ['Chase', 'Cowley', 'Lyon', 'Butler', 'Harvey', 'Sedgwick', 'Greenwood', 'Marion', 'Sumner'], which would give district 1 a population of 715,201, that district 2 contains counties ['Jackson', 'Bourbon', 'Doniphan', 'Montgomery', 'Woodson', 'Wyandotte', 'Cherokee', 'Crawford', 'Allen', 'Atchison', 'Brown', 'Jefferson', 'Chautauqua', 'Nemaha', 'Osage', 'Labette', 'Leavenworth', 'Linn', 'Anderson', 'Wilson', 'Neosho', 'Shawnee', 'Coffey', 'Elk'], which would give district 2 a population of 714,079, that district 3 contains counties ['Franklin', 'Johnson', 'Miami', 'Douglas'], which would give district 3 a population of 713,784. This will provide a map of four compact and contiguous districts, with a maximum population deviation of 0.45%, or 5,147 people. Our model solved this problem in 4.88 hours to give this optimal solution, which is much faster than it would have been if solved by hand. Also, when solving by hand, you run the risk of missing an optimal solution. The program we created imposes the compact and contiguous constraints, but leaves out the following constraints: preserving political subdivisions, preserving communities of interest, and

preserving cores of period districts. Without other datasets and limited coding knowledge, imposing the further constraints would have been difficult but would be needed for an accurate Kansas districting graph.

References:

https://districtr.org/plan/16435

http://people.csail.mit.edu/ddeford/COUNTY/COUNTY_20.json

https://lykhovyd.com/files/public/districting/KS/counties/graph/KS.population

https://lykhovyd.com/files/public/districting/KS/counties/maps/KS_counties.dbf

https://lykhovyd.com/files/public/districting/KS/counties/maps/KS_counties.prj

https://lykhovyd.com/files/public/districting/KS/counties/maps/KS_counties.qpj

https://lykhovyd.com/files/public/districting/KS/counties/maps/KS_counties.shp

https://lykhovyd.com/files/public/districting/KS/counties/maps/KS_counties.shx