Oklahoma and Oregon Redistricting Project Code_revA

December 12, 2023

1 Final Project: Redistricing Project of Oklahoma and Oregon.

1.0.1 Class: IE 5318 Principles of Operations Reasearch

1.0.2 Team Members: Natalie Ventura, Jessica Dedeaux, Damilola Fasheru

1.0.3 Date: Fall 2023; 12-12-2023

[1]:
```
# GitHub Project file: https://github.com/Jdedeaux/Districts-Counties/tree/main
# Credit to:
#     Dr. Hamidreza Validi transportation, short path codes, and assigment␣
↪codes: IE 5318 Principles of Operations Reasearch at Texas Tech University
#     Dr. Austin Buchanan districting example starter code: https://github.com/
↪AustinLBuchanan/Districting-Examples/tree/main
#     Gurobi Optimization Tutorial via the primary website: https://www.gurobi.
↪com/resources/lp-chapter-6-modeling-and-solving-linear-programming-problems/
#     Dr. Gabrielle Taylor: https://github.com/gabrielletay96
#     shapefiles: https://www.youtube.com/watch?v=xxg4Vm-Xg9g
#     for loops: https://www.youtube.com/watch?v=KWgYha0clzw&t=172s
#     Data: http://people.csail.mit.edu/ddeford/dual_graphs
#     Data: https://lykhovyd.com/files/public/districting/2020/
#     nested for loops: https://www.youtube.com/watch?v=APWy6Pc83gE
#     Networknx Package: https://networkx.org/documentation/stable/
↪auto_examples/drawing/plot_labels_and_colors.html
```

1.1 OKLAHOMA: Code Verification

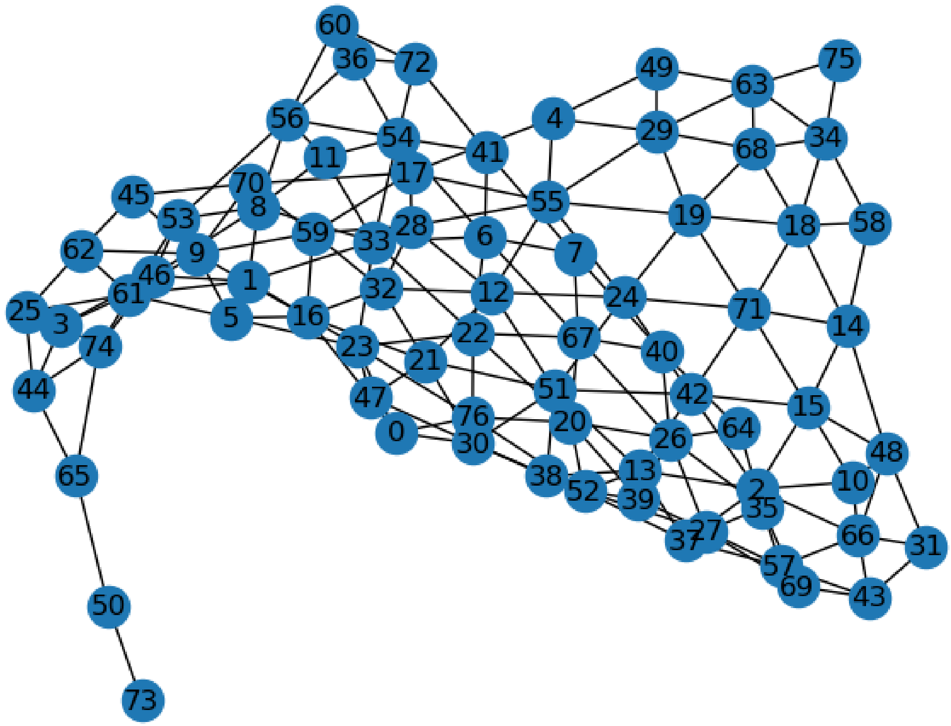
[2]:
```
# Import the following packages to run various cells
import gurobipy as gp
from gurobipy import GRB
import networkx as nx
import geopandas as gpd
import math
from gerrychain import Graph
import pandas as pd
import matplotlib.pyplot as plt
```

1.1.1 Step 1: Define the edges and nodes of the model

[3]:
```python
# Create the intial node and edges model using the edges list for Oklahoma
# Remove the non-integers from the text file and ensure it is a plain text file.
filepath = '/Users/jessi/Gurobi-Python Playground/OK.graph'

# reading from the data set
G = nx.read_edgelist(filepath,nodetype=int)

print(type(G))

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

# Current model description
print("The Oklahoma graph has",G.number_of_nodes(),"nodes")
print("The Oklahoma graph has",G.number_of_edges(),"edges")
print("Oklahoma graph has nodes",G.nodes )
print("Oklahoma graph has edges",G.edges )
```

```
<class 'networkx.classes.graph.Graph'>
The Oklahoma graph has 77 nodes
The Oklahoma graph has 195 edges
Oklahoma graph has nodes [0, 23, 30, 47, 76, 1, 5, 8, 33, 46, 61, 2, 10, 13, 15,
37, 42, 57, 66, 3, 25, 44, 74, 4, 17, 29, 49, 55, 9, 16, 6, 7, 22, 41, 54, 67,
40, 11, 53, 56, 45, 59, 62, 70, 48, 12, 21, 24, 28, 32, 51, 38, 52, 14, 18, 58,
71, 19, 34, 68, 20, 26, 39, 27, 35, 64, 69, 63, 31, 43, 75, 36, 60, 72, 65, 50,
73]
Oklahoma graph has edges [(0, 23), (0, 30), (0, 47), (0, 76), (23, 1), (23, 5),
(23, 22), (23, 33), (23, 47), (23, 76), (30, 21), (30, 38), (30, 47), (30, 51),
(30, 52), (30, 76), (47, 16), (47, 21), (76, 20), (76, 22), (76, 52), (1, 5),
(1, 8), (1, 33), (1, 46), (1, 61), (5, 9), (5, 16), (5, 61), (8, 11), (8, 33),
(8, 46), (8, 53), (8, 56), (33, 6), (33, 11), (33, 22), (33, 54), (46, 3), (46,
53), (46, 61), (46, 74), (61, 3), (61, 9), (61, 25), (61, 62), (2, 10), (2, 13),
(2, 15), (2, 37), (2, 42), (2, 57), (2, 66), (10, 15), (10, 48), (10, 66), (13,
37), (13, 38), (13, 42), (13, 51), (13, 52), (15, 14), (15, 42), (15, 48), (15,
71), (37, 27), (37, 52), (37, 57), (42, 24), (42, 51), (42, 71), (57, 27), (57,
43), (57, 66), (57, 69), (66, 31), (66, 43), (66, 48), (3, 25), (3, 44), (3,
74), (25, 44), (25, 62), (44, 65), (44, 74), (74, 53), (74, 65), (4, 17), (4,
29), (4, 49), (4, 55), (17, 28), (17, 55), (17, 59), (17, 70), (29, 19), (29,
49), (29, 55), (29, 63), (29, 68), (49, 63), (55, 12), (55, 19), (55, 24), (55,
28), (9, 16), (9, 45), (9, 59), (9, 62), (9, 70), (16, 21), (16, 32), (16, 59),
(6, 7), (6, 22), (6, 41), (6, 54), (6, 67), (7, 40), (7, 41), (7, 67), (22, 20),
(22, 67), (41, 54), (41, 72), (54, 11), (54, 36), (54, 56), (54, 72), (67, 20),
(67, 26), (67, 40), (40, 26), (40, 64), (11, 56), (53, 56), (56, 36), (56, 60),
(45, 62), (45, 70), (59, 28), (59, 32), (59, 70), (48, 14), (48, 31), (12, 21),
(12, 24), (12, 28), (12, 32), (12, 51), (21, 32), (21, 51), (24, 19), (24, 51),
(24, 71), (28, 32), (51, 38), (38, 52), (52, 20), (52, 27), (52, 39), (14, 18),
```

```
(14, 58), (14, 71), (18, 19), (18, 34), (18, 58), (18, 68), (18, 71), (58, 34),
(71, 19), (19, 68), (34, 63), (34, 68), (34, 75), (68, 63), (20, 26), (20, 39),
(26, 27), (26, 35), (26, 39), (26, 64), (39, 27), (27, 35), (27, 69), (35, 64),
(35, 69), (69, 43), (63, 75), (31, 43), (36, 60), (36, 72), (60, 72), (65, 50),
(50, 73)]
```



1.1.2 Step 2: List the county populations

[4]:
```
# Integrate the populations by county to the model

# create population list
population = list()

# read text file into software system using files on local drive
filepath = "/Users/jessi/Gurobi-Python Playground/OK-2.population"
file = open( filepath,"r")

# Create List for each county population
Listpop = []
for each in file:
```

```
    #split the list so the populations in each range
    each = each.split(' ')
    #print((each[1])
    Listpop.append(int(each[1]))

# displays the list of integers
print("Oklahoma's county populations = ", Listpop)

# the sums can be taken by list position
total_population = sum(Listpop)

#print the total population of OK
print("The total_population of Oklahoma = ", total_population)
population = Listpop #label for code clarity
n =len(population)

#print the county population as a sum districts of OK
print("Oklahoma's number of counties = ",n)
print("Population data is", type(population))
```

```
Oklahoma's county populations =  [255755, 11943, 45837, 20081, 50976, 15034,
124098, 6193, 27469, 60580, 11154, 11629, 69967, 14003, 42391, 12769, 41848,
47472, 46987, 73085, 27576, 34273, 52431, 115541, 40069, 8878, 47557, 10957,
16577, 86905, 69442, 33151, 77350, 29600, 41487, 15840, 6239, 5925, 25482,
13488, 6472, 7992, 20252, 15205, 3685, 4527, 4810, 718633, 50384, 10536, 20640,
12191, 37492, 3647, 9446, 603403, 22119, 14182, 22683, 11561, 2922, 7527, 5642,
15029, 9423, 5636, 11572, 45048, 41259, 42416, 46562, 70990, 26446, 2475, 4151,
31848, 34506]
The total_population of Oklahoma =  3751351
Oklahoma's number of counties =  77
Population data is <class 'list'>
```

1.1.3 Step 3: Population Deviation Model

[5]:
```
# If model is feasible, use the population deviation operations research model.
# create model
m = gp.Model()
#number of districts per the 2020 census for OK
k = 5

# create variables adapted from districting starter models
x = m.addVars(n,k, vtype=GRB.BINARY) # x_ij equals one when county i is␣
↪assigned to district j
y = m.addVar()                     # the population of least-populated district
z = m.addVar()                     # the population of most-populated district

# objective is to minimize absolute population deviation
```

```
m.setObjective(z-y, GRB.MINIMIZE)

# add constraints saying that each county i is assigned to one district
m.addConstrs( sum(x[i,j] for j in range(k)) == 1 for i in range(n) )

# add constraints saying that each district has population at least y
m.addConstrs(y <= sum(population[i] * x[i,j] for i in range(n)) for j in␣
 ↪range(k) )

# add constraints saying that each district has population at most z
m.addConstrs( sum( population[i] * x[i,j] for i in range(n)) <= z for j in␣
 ↪range(k) )

# solve IP model
m.optimize()
```

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 87 rows, 387 columns and 1165 nonzeros
Model fingerprint: 0x235259e8
Variable types: 2 continuous, 385 integer (385 binary)
Coefficient statistics:
  Matrix range     [1e+00, 7e+05]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 1149841.0000
Presolve time: 0.00s
Presolved: 87 rows, 387 columns, 1165 nonzeros
Variable types: 0 continuous, 387 integer (385 binary)

Root relaxation: objective 0.000000e+00, 125 iterations, 0.00 seconds (0.00 work
units)

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

     0     0    0.00000    0    7 1149841.00    0.00000   100%     -    0s
H    0     0                    339690.00000    0.00000   100%     -    0s
H    0     0                    190691.00000    0.00000   100%     -    0s
H    0     0                    29846.000000    0.00000   100%     -    0s
H    0     0                    29061.000000    0.00000   100%     -    0s
     0     0    0.00000    0   12 29061.0000    0.00000   100%     -    0s
H    0     0                    18448.000000    0.00000   100%     -    0s
```

```
     0     0    0.00000    0   12 18448.0000    0.00000   100%     -    0s
H    0     0                    13568.000000    0.00000   100%     -    0s
     0     0    0.00000    0    9 13568.0000    0.00000   100%     -    0s
H    0     0                    13270.000000    0.00000   100%     -    0s
     0     0    0.00000    0   11 13270.0000    0.00000   100%     -    0s
H    0     0                    9740.0000000    0.00000   100%     -    0s
H    0     0                    7343.0000000    0.00000   100%     -    0s
     0     0    0.00000    0    7 7343.00000    0.00000   100%     -    0s
H    0     0                    4835.0000000    0.00000   100%     -    0s
     0     0    0.00000    0    7 4835.00000    0.00000   100%     -    0s
H    0     0                    2400.0000000    0.00000   100%     -    0s
     0     0    0.00000    0    9 2400.00000    0.00000   100%     -    0s
     0     0    0.00000    0    9 2400.00000    0.00000   100%     -    0s
H    0     0                    1655.0000000    0.00000   100%     -    0s
H    0     0                    1349.0000000    0.00000   100%     -    0s
H    0     0                    1059.0000000    0.00000   100%     -    0s
     0     2    1.00000    0    7 1059.00000    1.00000   100%     -    0s
H    8     7                     963.0000000    1.00000   100%   9.0    0s
H   10     7                     828.0000000    1.00000   100%   9.9    0s
H   32    21                     265.0000000    1.00000   100%   9.9    0s
H   84    70                     207.0000000    1.00000   100%   6.4    0s
H  151   296                     133.0000000    1.00000  99.2%   5.0    0s
H 2618  1940                      68.0000000    1.00000  98.5%   4.1    0s
H 2698  1905                      48.0000000    1.00000  97.9%   4.8    0s
*14689  7127              95      46.0000000    1.00000  97.8%   4.9    2s
H14723  6759                      40.0000000    1.00000  97.5%   4.9    3s
H14736  6428                      33.0000000    1.00000  97.0%   4.9    3s
H14772  6137                      25.0000000    1.00000  96.0%   5.0    3s
H16044  6517                      24.0000000    1.00000  95.8%   5.1    3s
 22064  7847 infeasible  130        24.00000    1.00000  95.8%   5.4    5s
*27388  7146             116      14.0000000    1.00000  92.9%   5.6    5s
H29273  6390                      12.0000000    1.00000  91.7%   5.7    5s
 43956  6339 infeasible  112        12.00000    1.00000  91.7%   5.7   10s
H50017  6328                      10.0000000    1.00000  90.0%   5.7   10s
H70833  6330                       9.0000000    1.00000  88.9%   5.9   12s
*73086  5777             107       8.0000000    1.00000  87.5%   5.9   13s
 83148  5995 infeasible  106         8.00000    1.00000  87.5%   5.9   15s
*83297  5907             104       5.0000000    1.00000  80.0%   5.9   15s
H111759  6572                       3.0000000    1.00000  66.7%   5.7   18s
H120568  6515                       2.0000000    1.00000  50.0%   5.6   19s
 120814  6754 infeasible   53         2.00000    1.00000  50.0%   5.5   20s
 159469  4916    1.00000  143    5    2.00000    1.00000  50.0%   5.5   25s
H186669   959                       1.0000000    1.00000  0.00%   5.4   28s

Cutting planes:
  Gomory: 1
  Cover: 113
  MIR: 144
```

```
  StrongCG: 150
  Flow cover: 121
  Inf proof: 2
  Zero half: 1
  RLT: 3

Explored 188295 nodes (1027011 simplex iterations) in 28.94 seconds (16.15 work
units)
Thread count was 8 (of 8 available processors)

Solution count 10: 1 2 3 … 24

Optimal solution found (tolerance 1.00e-04)
Best objective 1.000000000000e+00, best bound 1.000000000000e+00, gap 0.0000%
```

1.1.4 Step 4: Current model status for population

[6]:
```python
# print the absolute population deviation
print("The absolute population deviation is",m.objval,"person(s).")
print("Oklahoma has districts =",k)
```

```
The absolute population deviation is 1.0 person(s).
Oklahoma has districts = 5
```

1.1.5 Step 5: create model to access the minimization of cut edges using the county to district populations

[7]:
```python
filepath = '/Users/jessi/Gurobi-Python Playground/OK_county.json'
G = Graph.from_json(filepath)
print(G.nodes)

m = gp.Model()
#check the list of nodes and matches are align with the mapping data
print(G.nodes)
print(G.edges)

for node in G.nodes:
    # using the columns, located the population
    population = G.nodes[node]['TOTPOP']

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

# Impose a 3% population deviation (+/- 0.5%) for the Operations Research model
deviation = 0.03
#number of counties
n
# number of districts
```

```
k

# Lower and Upper bound populations to ensure the populations of the districts␣
 ↪do not exceed the bounds
L = math.ceil((1-deviation/2)*total_population/k)
U = math.floor((1+deviation/2)*total_population/k)

# create variables adapted from redistricitng starter:
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

# objective is to minimize cut edges
m.setObjective( gp.quicksum( 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()

# solve IP model
m.optimize()

# print the optimal minization of cut edges
print("Oklahoma: The number of cut edges is",m.objval)
print("Oklahoma: Using Lower Bound of Population =",L,"and Upper Bound of␣
 ↪Population =",U,"and districts =",k)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 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]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 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]
[(0, 66), (0, 57), (0, 61), (0, 73), (0, 27), (1, 51), (1, 22), (1, 45), (1,
56), (1, 68), (1, 53), (1, 46), (2, 26), (2, 63), (2, 36), (2, 31), (2, 43), (3,
32), (3, 37), (3, 71), (3, 72), (4, 40), (4, 28), (4, 22), (4, 24), (5, 72), (5,
25), (5, 20), (5, 47), (5, 58), (6, 20), (6, 58), (6, 75), (6, 19), (7, 71), (7,
72), (7, 65), (7, 20), (8, 32), (8, 52), (8, 71), (8, 35), (9, 52), (9, 40), (9,
51), (9, 22), (10, 47), (10, 58), (10, 60), (10, 19), (10, 39), (10, 69), (11,
37), (11, 76), (11, 25), (11, 34), (12, 42), (12, 48), (12, 70), (12, 33), (13,
54), (13, 30), (13, 38), (14, 53), (14, 21), (14, 46), (14, 42), (14, 29), (14,
48), (15, 67), (15, 75), (15, 68), (15, 66), (15, 61), (16, 76), (16, 34), (16,
55), (16, 60), (16, 59), (17, 74), (17, 49), (17, 31), (18, 41), (18, 33), (19,
58), (19, 75), (19, 69), (19, 66), (19, 57), (20, 72), (20, 65), (20, 58), (20,
44), (20, 75), (21, 24), (21, 53), (21, 42), (22, 40), (22, 51), (22, 24), (22,
56), (23, 69), (23, 57), (23, 64), (23, 74), (23, 73), (23, 62), (23, 49), (24,
28), (24, 56), (24, 53), (25, 37), (25, 72), (25, 34), (25, 47), (26, 59), (26,
63), (27, 61), (27, 73), (27, 54), (27, 62), (27, 38), (28, 40), (29, 46), (29,
50), (29, 54), (29, 48), (29, 70), (29, 30), (30, 54), (30, 70), (31, 36), (31,
74), (31, 43), (32, 71), (33, 42), (34, 76), (34, 55), (34, 47), (35, 52), (35,
71), (35, 51), (35, 65), (35, 45), (36, 63), (36, 64), (36, 74), (37, 72), (38,
54), (38, 62), (39, 60), (39, 59), (39, 69), (39, 63), (39, 64), (42, 48), (44,
65), (44, 45), (44, 67), (44, 75), (44, 68), (45, 51), (45, 65), (45, 68), (46,
68), (46, 53), (46, 50), (47, 55), (47, 58), (47, 60), (48, 70), (49, 74), (49,
62), (50, 68), (50, 61), (50, 54), (51, 52), (53, 56), (54, 61), (55, 60), (57,
69), (57, 66), (57, 73), (59, 60), (59, 63), (61, 66), (62, 73), (63, 64), (64,
69), (64, 74), (65, 71), (66, 75), (67, 75), (67, 68), (71, 72)]
3751351
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 1062 rows, 580 columns and 4080 nonzeros
Model fingerprint: 0x7b871e34
Variable types: 0 continuous, 580 integer (580 binary)
Coefficient statistics:
  Matrix range     [1e+00, 7e+05]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 8e+05]
Presolve time: 0.01s
Presolved: 1062 rows, 580 columns, 4080 nonzeros
Variable types: 0 continuous, 580 integer (580 binary)
Found heuristic solution: objective 102.0000000

Root relaxation: objective 0.000000e+00, 683 iterations, 0.06 seconds (0.03 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
```

	Expl	Unexpl	Obj	Depth	IntInf	Incumbent	BestBd	Gap	It/Node	Time
	0	0	0.00000	0	385	102.00000	0.00000	100%	-	0s
H	0	0				100.0000000	0.00000	100%	-	0s
	0	0	3.54115	0	394	100.00000	3.54115	96.5%	-	0s
	0	0	9.75216	0	322	100.00000	9.75216	90.2%	-	0s
H	0	0				94.0000000	9.75216	89.6%	-	0s
	0	0	9.77816	0	322	94.00000	9.77816	89.6%	-	0s
H	0	0				80.0000000	9.77816	87.8%	-	0s
	0	0	10.25155	0	325	80.00000	10.25155	87.2%	-	0s
	0	0	10.31184	0	310	80.00000	10.31184	87.1%	-	1s
	0	0	10.32740	0	310	80.00000	10.32740	87.1%	-	1s
	0	0	10.32740	0	310	80.00000	10.32740	87.1%	-	1s
	0	0	10.48686	0	373	80.00000	10.48686	86.9%	-	1s
	0	0	10.57637	0	316	80.00000	10.57637	86.8%	-	1s
	0	0	10.72201	0	309	80.00000	10.72201	86.6%	-	1s
	0	0	10.73880	0	362	80.00000	10.73880	86.6%	-	1s
	0	0	10.74733	0	377	80.00000	10.74733	86.6%	-	1s
	0	0	10.74776	0	381	80.00000	10.74776	86.6%	-	1s
	0	0	10.81927	0	333	80.00000	10.81927	86.5%	-	1s
	0	0	10.83029	0	312	80.00000	10.83029	86.5%	-	1s
	0	0	10.83029	0	304	80.00000	10.83029	86.5%	-	1s
	0	0	10.83029	0	304	80.00000	10.83029	86.5%	-	1s
	0	0	10.83029	0	304	80.00000	10.83029	86.5%	-	1s
	0	2	10.83029	0	304	80.00000	10.83029	86.5%	-	1s
H	36	34				78.0000000	17.12159	78.0%	352	2s
H	64	70				74.0000000	17.12159	76.9%	303	2s
H	66	70				73.0000000	17.12159	76.5%	298	2s
H	70	70				61.0000000	17.12159	71.9%	295	2s
H	103	110				59.0000000	17.12159	71.0%	257	3s
H	104	110				57.0000000	17.12159	70.0%	256	3s
H	108	110				55.0000000	17.12159	68.9%	251	3s
H	149	142				51.0000000	17.12159	66.4%	225	3s
H	194	168				47.0000000	17.12159	63.6%	219	3s
H	196	168				46.0000000	17.12159	62.8%	219	3s
H	196	168				45.0000000	17.12159	62.0%	219	3s
H	277	236				42.0000000	17.12159	59.2%	202	3s
	509	374	39.92087	23	135	42.00000	17.12159	59.2%	182	5s
H	597	414				39.0000000	18.06854	53.7%	189	6s
	1273	732	32.65139	15	272	39.00000	19.83573	49.1%	199	10s
	1493	813	21.63256	15	281	39.00000	21.63256	44.5%	201	15s
	1937	929	30.08872	25	376	39.00000	21.63256	44.5%	212	20s
	2582	1061	cutoff	37		39.00000	24.30628	37.7%	207	25s
	3347	1243	cutoff	43		39.00000	25.82863	33.8%	201	30s
	4361	1478	34.36045	23	332	39.00000	27.14976	30.4%	193	35s
	5563	2021	35.46161	27	315	39.00000	28.00574	28.2%	186	40s
	6376	2332	32.66073	21	317	39.00000	28.39526	27.2%	182	45s
	7899	2871	30.59452	28	361	39.00000	29.04111	25.5%	173	50s

```
  9302  3242   32.01120   24  341   39.00000   29.49253  24.4%   170   56s
 10454  3613   36.11452   26  269   39.00000   29.75403  23.7%   166   60s
 11861  3965   34.24716   32  359   39.00000   30.12514  22.8%   164   65s
 13467  4408   32.81638   25  284   39.00000   30.51311  21.8%   161   71s
 14148  4592   36.98568   31  299   39.00000   30.67587  21.3%   160   76s
 15541  4844   37.58558   32  249   39.00000   31.00584  20.5%   157   80s
 17461  5413   33.46229   26  295   39.00000   31.32097  19.7%   154   86s
 18819  5662 infeasible   27         39.00000   31.55234  19.1%   152   90s
 20258  5973     cutoff   36         39.00000   31.72164  18.7%   151   95s
 21936  6252     cutoff   31         39.00000   31.98454  18.0%   148  100s
 24431  6653   34.75042   26  303   39.00000   32.31155  17.1%   146  107s
 25412  6793   35.99915   26  326   39.00000   32.42132  16.9%   144  110s
 27094  7021   33.79341   26  179   39.00000   32.59847  16.4%   143  115s
 29648  7358   36.94286   26  309   39.00000   32.86213  15.7%   141  143s
 30728  7391   37.44637   30  212   39.00000   32.97760  15.4%   140  145s
 32025  7418   36.74339   28  286   39.00000   33.13178  15.0%   140  150s
 33604  7480   37.67916   33  311   39.00000   33.28150  14.7%   140  155s
 35171  7500   37.46878   47  200   39.00000   33.45606  14.2%   140  160s
 36719  7455   35.48846   27  395   39.00000   33.63329  13.8%   139  165s
 38214  7413   36.83750   33  160   39.00000   33.78725  13.4%   140  170s
 39377  7407   37.01480   31  147   39.00000   33.87441  13.1%   139  177s
 40074  7373     cutoff   30         39.00000   33.94368  13.0%   140  180s
 41718  7249   37.27453   30  231   39.00000   34.12849  12.5%   139  185s
 43176  7085   37.88204   30  194   39.00000   34.31174  12.0%   140  191s
 44925  6953   37.45237   26  364   39.00000   34.48929  11.6%   140  196s
*45981  5723              52      38.0000000   34.57873  9.00%   139  199s
 46669  5433     cutoff   32        38.00000   34.69002  8.71%   139  202s
H46678  2871                      37.0000000   34.69002  6.24%   139  202s
 47256  2457     cutoff   31        37.00000   34.79256  5.97%   139  205s
 49344   384     cutoff   45        37.00000   35.63649  3.69%   138  210s

Cutting planes:
  Gomory: 10
  Cover: 3
  Implied bound: 30
  MIR: 2
  Flow cover: 12
  Zero half: 4
  RLT: 159

Explored 50328 nodes (6849996 simplex iterations) in 210.94 seconds (148.30 work
units)
Thread count was 8 (of 8 available processors)

Solution count 10: 37 38 39 … 57

Optimal solution found (tolerance 1.00e-04)
Best objective 3.700000000000e+01, best bound 3.700000000000e+01, gap 0.0000%
```

```
Oklahoma: The number of cut edges is 37.0
Oklahoma: Using Lower Bound of Population = 739017 and Upper Bound of Population
= 761524 and districts = 5
```

1.1.6 Step 6: Retrieve the districts and their populations

[8]:
```python
districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
district_populations = [ sum(G.nodes[i]["TOTPOP"] for i in districts[j]) for j␣
 ↪in range(k) ]

# print district status
for j in range(k):
    print("Oklahoma: District",j,"has population",district_populations[j],"and␣
 ↪contains counties",districts[j])
```

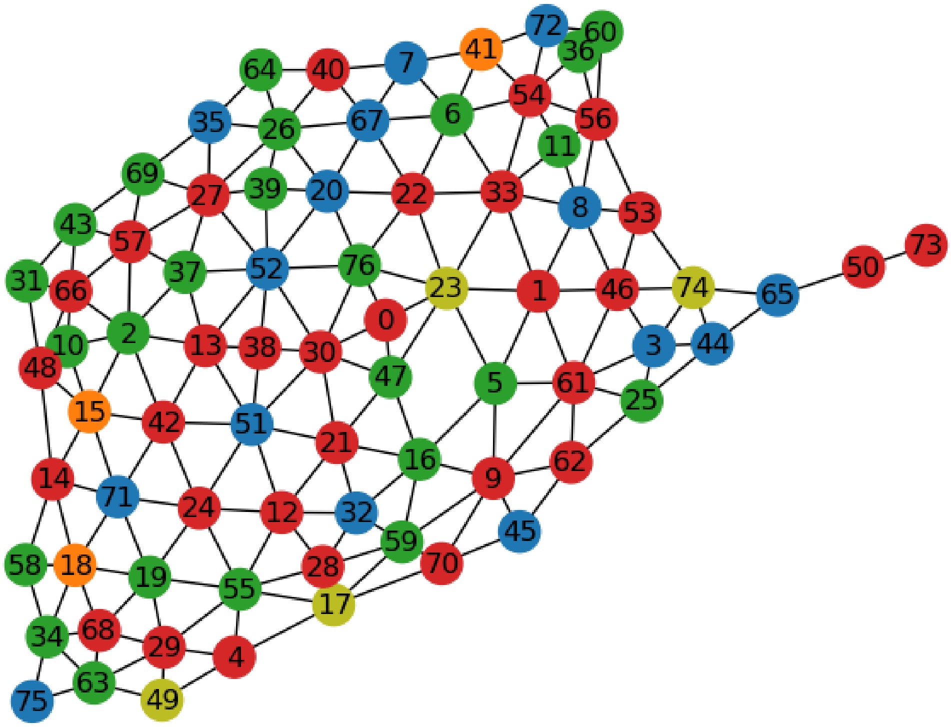
```
Oklahoma: District 0 has population 761154 and contains counties [0, 1, 4, 9,
12, 13, 14, 21, 22, 24, 27, 28, 29, 30, 33, 38, 40, 42, 46, 48, 50, 53, 54, 56,
57, 61, 62, 66, 68, 70, 73]
Oklahoma: District 1 has population 741748 and contains counties [15, 18, 41]
Oklahoma: District 2 has population 751820 and contains counties [17, 23, 49,
74]
Oklahoma: District 3 has population 740351 and contains counties [2, 5, 6, 10,
11, 16, 19, 25, 26, 31, 34, 36, 37, 39, 43, 47, 55, 58, 59, 60, 63, 64, 69, 76]
Oklahoma: District 4 has population 756278 and contains counties [3, 7, 8, 20,
32, 35, 44, 45, 51, 52, 65, 67, 71, 72, 75]
```

1.1.7 Step 7: Node Mapping the counties and districts

[9]:
```python
filepath = '/Users/jessi/Gurobi-Python Playground/OK.graph'
G = nx.read_edgelist(filepath,nodetype=int)

print(type(G))

# List of districts
districts = ["District 0","District 1","District 2","District 3","District 4"]

# County to District Dictionary
counties_district_dict = {
"District 0": [0, 1, 4, 9, 12, 13, 14, 21, 22, 24, 27, 28, 29, 30, 33, 38, 40,␣
 ↪42, 46, 48, 50, 53, 54, 56, 57, 61, 62, 66, 68, 70, 73],
"District 1":  [15, 18, 41],
"District 2":  [17, 23, 49, 74],
"District 3":  [2, 5, 6, 10, 11, 16, 19, 25, 26, 31, 34, 36, 37, 39, 43, 47,␣
 ↪55, 58, 59, 60, 63, 64, 69, 76],
"District 4":  [3, 7, 8, 20, 32, 35, 44, 45, 51, 52, 65, 67, 71, 72, 75]
}

# Color dictionary
```

```python
counties_district_colors_dict = {
    "District 0": "tab:red",
    "District 1": "tab:orange",
    "District 2": "tab:olive",
    "District 3": "tab:green",
    "District 4": "tab:blue"
}

# Defining the position
pos = nx.spring_layout(G)

# Assigning the colors
color_list = []
for node in G.nodes:   # iteration over nodes
    if node in counties_district_dict['District 0']:
        color_list.append(counties_district_colors_dict['District 0'])

    elif node in counties_district_dict['District 1']:
        color_list.append(counties_district_colors_dict['District 1'])

    elif node in counties_district_dict['District 2']:
        color_list.append(counties_district_colors_dict['District 2'])

    elif node in counties_district_dict['District 3']:
        color_list.append(counties_district_colors_dict['District 3'])
    else:
        color_list.append(counties_district_colors_dict['District 4'])

nx.draw(G, pos, node_color=color_list,with_labels=True)
plt.show()
```

```
<class 'networkx.classes.graph.Graph'>
```

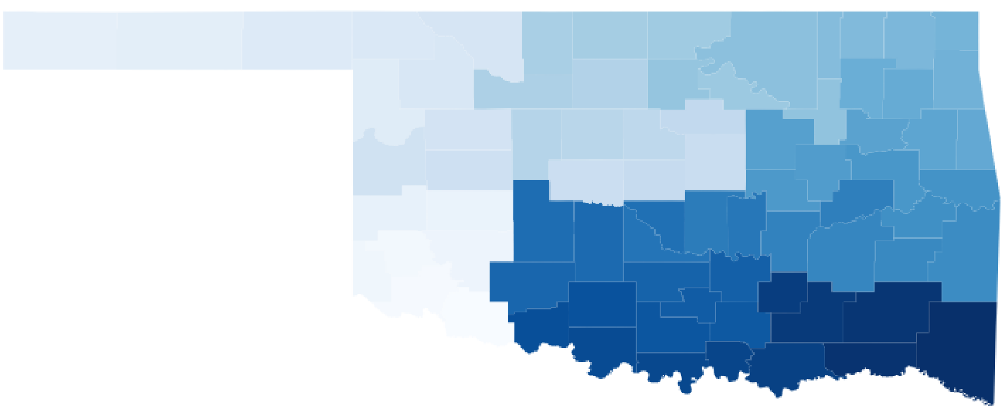


1.1.8 Step 8: State shape Mapping the counties

[10]:
```
# Read Oklahoma county shapefile and distance files
file1 = '/Users/jessi/Gurobi-Python Playground/OK_distances.csv'
file2 = '/Users/jessi/Gurobi-Python Playground/OK_counties.shx'
file = '/Users/jessi/Gurobi-Python Playground/OK_counties.shp'
#create the data frame
df = gpd.read_file(file)

#checker
print((df))
print(type(df))

#assign the districts to the counties
districts = [[i for i in range(n) if x[i,j].x > 0.5] for j in range(k)]
print("Oklahoma's counties are in the following districts", districts)

# Mapping
fig,ax = plt.subplots(figsize=(10,10))
df.plot(ax=ax, column = "geometry", cmap ='Blues')
```

```python
ax.axis("off")
print("Oklahoma's counties are shown by varied blues in the diagram")
```

```
                                              geometry
0   POLYGON ((-99.36446 35.30577, -99.36446 35.306…
1   POLYGON ((-99.42320 34.72502, -99.42314 34.725…
2   POLYGON ((-98.53366 36.39787, -98.53369 36.404…
3   POLYGON ((-95.00543 36.40142, -95.00543 36.405…
4   POLYGON ((-99.14619 35.81240, -99.14280 35.812…
..                                                ...
72  POLYGON ((-96.62036 35.95931, -96.62051 35.963…
73  POLYGON ((-97.14206 35.14521, -97.14208 35.146…
74  POLYGON ((-99.18527 35.11660, -99.18520 35.116…
75  POLYGON ((-97.24410 33.90427, -97.24505 33.903…
76  POLYGON ((-97.46132 36.43373, -97.46132 36.434…

[77 rows x 1 columns]
<class 'geopandas.geodataframe.GeoDataFrame'>
Oklahoma's counties are in the following districts [[0, 1, 4, 9, 12, 13, 14, 21,
22, 24, 27, 28, 29, 30, 33, 38, 40, 42, 46, 48, 50, 53, 54, 56, 57, 61, 62, 66,
68, 70, 73], [15, 18, 41], [17, 23, 49, 74], [2, 5, 6, 10, 11, 16, 19, 25, 26,
31, 34, 36, 37, 39, 43, 47, 55, 58, 59, 60, 63, 64, 69, 76], [3, 7, 8, 20, 32,
35, 44, 45, 51, 52, 65, 67, 71, 72, 75]]
Oklahoma's counties are shown by varied blues in the diagram
```

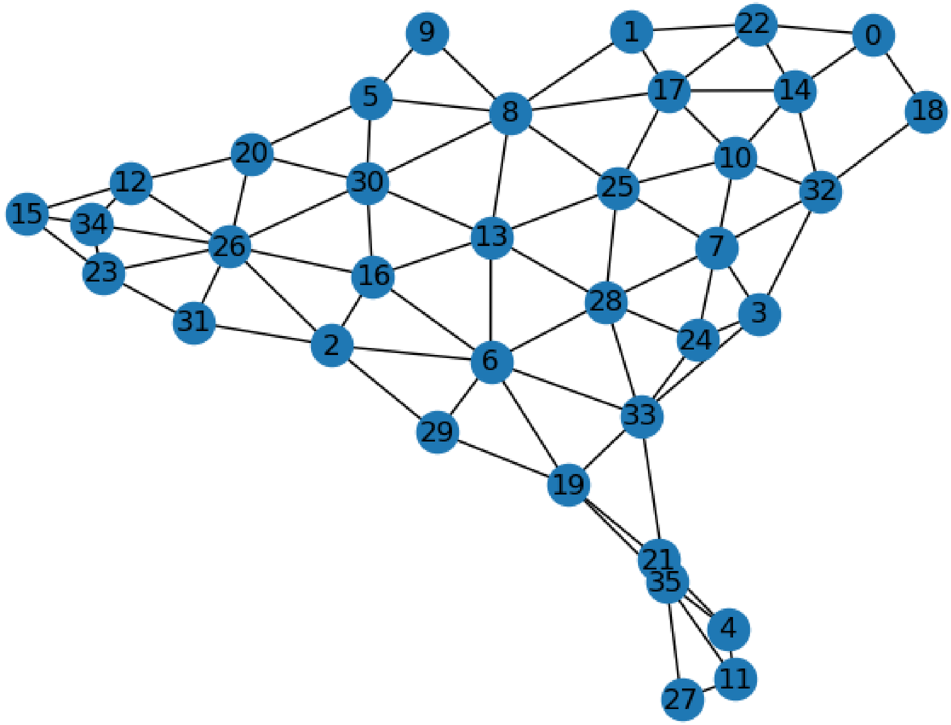


1.2 OREGON

1.2.1 Step 1: Define the edges and nodes of the model

[11]:
```
# Create the intial node and edges model using the edges list for Oregon
# Remove the non-integers from the text file and ensure it is a plain text file.
filepath = '/Users/jessi/Gurobi-Python Playground/OR.dimacs-mod.txt'

# reading from the data set
G = nx.read_edgelist(filepath,nodetype=int)

print(type(G))

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

# Current model description
print("The Oregon graph has",G.number_of_nodes(),"nodes")
print("The Oregon graph has",G.number_of_edges(),"edges")
print("Oregon graph has nodes",G.nodes )
print("Oregon graph has edges",G.edges )
```

```
<class 'networkx.classes.graph.Graph'>
The Oregon graph has 36 nodes
The Oregon graph has 81 edges
Oregon graph has nodes [0, 14, 18, 22, 1, 8, 17, 2, 6, 16, 26, 29, 31, 3, 7, 24,
32, 33, 4, 11, 21, 35, 5, 9, 20, 30, 13, 19, 28, 10, 25, 27, 12, 15, 34, 23]
Oregon graph has edges [(0, 14), (0, 18), (0, 22), (14, 10), (14, 17), (14, 22),
(14, 32), (18, 32), (22, 1), (22, 17), (1, 8), (1, 17), (8, 5), (8, 9), (8, 13),
(8, 17), (8, 25), (8, 30), (17, 10), (17, 25), (2, 6), (2, 16), (2, 26), (2,
29), (2, 31), (6, 13), (6, 16), (6, 19), (6, 28), (6, 29), (6, 33), (16, 13),
(16, 26), (16, 30), (26, 12), (26, 20), (26, 23), (26, 30), (26, 31), (26, 34),
(29, 19), (31, 23), (3, 7), (3, 24), (3, 32), (3, 33), (7, 10), (7, 24), (7,
25), (7, 28), (7, 32), (24, 28), (24, 33), (32, 10), (33, 19), (33, 28), (33,
35), (4, 11), (4, 21), (4, 35), (11, 27), (11, 35), (21, 19), (21, 35), (35,
19), (35, 27), (5, 9), (5, 20), (5, 30), (20, 12), (20, 30), (30, 13), (13, 25),
(13, 28), (28, 25), (10, 25), (12, 15), (12, 34), (15, 23), (15, 34), (34, 23)]
```



1.2.2 Step 2: List the county populations

[12]:
```
# Integrate the populations by county to the model

# create population list
population = list()

# read text file into software system using files on local drive
filepath = "/Users/jessi/Gurobi-Python Playground/OR.population"
file = open( filepath,"r")

#remove the header to only read the integers
next(file)

# Create List for each county population
Listpop = []
for each in file:
    #split the list so the populations in each range
    each = each.split(' ')
    #print((each[1])
```

```
    Listpop.append(int(each[1]))

# displays the list of integers
print("Oregon's county populations = ", Listpop)

# the sums can be taken by list position
total_population = sum(Listpop)

#print the total population of OR
print("The total_population of Oregon = ", total_population)
population = Listpop #label for code clarity
n =len(population)

#print the county population as a sum districts of OR
print("Oregon's number of counties = ",n)
print("Population data is", type(population))

districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
```

```
Oregon's county populations =  [52589, 23977, 7495, 50395, 88090, 1995, 198253,
87433, 26670, 1870, 107722, 23446, 80075, 24502, 600372, 7391, 24738, 421401,
41072, 69413, 12186, 223259, 815428, 16668, 95184, 345920, 7233, 64929, 128610,
8160, 1451, 31571, 27390, 382971, 26196, 111201]
The total_population of Oregon =  4237256
Oregon's number of counties =  36
Population data is <class 'list'>
```

1.2.3 Step 3: Population Deviation Model

[13]:
```
# If model is feasible, use the population deviation operations research model.
# create model
m = gp.Model()
#number of districts per the 2020 census for OR
k = 6

# create variables adapted from districting starter models
x = m.addVars(n,k, vtype=GRB.BINARY) # x_ij equals one when county i is␣
 ↪assigned to district j
y = m.addVar()                      # the population of least-populated district
z = m.addVar()                      # the population of most-populated district

# objective is to minimize absolute population deviation
m.setObjective(z-y, GRB.MINIMIZE)

# add constraints saying that each county i is assigned to one district
m.addConstrs( sum(x[i,j] for j in range(k)) == 1 for i in range(n) )
```

```python
# add constraints saying that each district has population at least y
m.addConstrs(y <= sum(population[i] * x[i,j] for i in range(n)) for j in␣
↪range(k) )

# add constraints saying that each district has population at most z
m.addConstrs( sum( population[i] * x[i,j] for i in range(n)) <= z for j in␣
↪range(k) )

# solve IP model
m.optimize()
```

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 48 rows, 218 columns and 660 nonzeros
Model fingerprint: 0x00001389
Variable types: 2 continuous, 216 integer (216 binary)
Coefficient statistics:
  Matrix range     [1e+00, 8e+05]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 957825.00000
Presolve time: 0.00s
Presolved: 48 rows, 218 columns, 660 nonzeros
Variable types: 0 continuous, 218 integer (216 binary)

Root relaxation: objective 0.000000e+00, 52 iterations, 0.00 seconds (0.00 work
units)

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

     0     0    0.00000    0    9 957825.000    0.00000   100%     -    0s
H    0     0                    611097.00000    0.00000   100%     -    0s
H    0     0                    447504.00000    0.00000   100%     -    0s
     0     0    0.33333    0   14 447504.000    0.33333   100%     -    0s
H    0     0                    213397.00000    0.33333   100%     -    0s
H    0     0                    198044.00000 61593.0000  68.9%     -    0s
     0     0 61593.0000    0   13 198044.000 61593.0000  68.9%     -    0s
H    0     0                    196344.00000 61593.0000  68.6%     -    0s
H    0     0                    139288.00000 61593.0000  55.8%     -    0s
H    0     0                    132428.00000 61593.0000  53.5%     -    0s
     0     0 61593.0000    0   10 132428.000 61593.0000  53.5%     -    0s
     0     0 109219.000    0   13 132428.000 109219.000  17.5%     -    0s
```

```
      0     0 109219.000    0   16 132428.000 109219.000  17.5%     -    0s
H     0     0                    132183.00000 109219.000  17.4%     -    0s
      0     0 109219.000    0   22 132183.000 109219.000  17.4%     -    0s
H     0     0                    132140.00000 109219.000  17.3%     -    0s
      0     0 109219.000    0   24 132140.000 109219.000  17.3%     -    0s
      0     0 109219.000    0   12 132140.000 109219.000  17.3%     -    0s
      0     0 109219.000    0   12 132140.000 109219.000  17.3%     -    0s
H     0     0                    132029.00000 109219.000  17.3%     -    0s
      0     0 109219.000    0   17 132029.000 109219.000  17.3%     -    0s
      0     0 109219.000    0   11 132029.000 109219.000  17.3%     -    0s
H     0     2                    131960.00000 109219.000  17.2%     -    0s
      0     2 109219.000    0   10 131960.000 109219.000  17.2%     -    0s
H    31    19                    131825.00000 131063.000  0.58%  10.1    0s
H    78    85                    131525.00000 131063.000  0.35%   7.2    0s
H   183   236                    131347.00000 131063.000  0.22%   4.7    0s
H   604   757                    131323.00000 131063.000  0.20%   3.7    0s
H   742   757                    131175.00000 131063.000  0.09%   3.7    0s
* 9935  3740              70    131116.00000 131063.000  0.04%   1.2    1s
*15801  2908              74    131080.00000 131063.000  0.01%   1.5    1s
*20150  2872              80    131069.00000 131063.000  0.00%   1.6    1s

Cutting planes:
  Cover: 2
  Flow cover: 3

Explored 20804 nodes (56687 simplex iterations) in 1.54 seconds (0.46 work
units)
Thread count was 8 (of 8 available processors)

Solution count 10: 131069 131080 131116 … 132029

Optimal solution found (tolerance 1.00e-04)
Best objective 1.310690000000e+05, best bound 1.310630000000e+05, gap 0.0046%
```

1.2.4 Step 4: Current model status for population

[14]:
```python
# print the absolute population deviation
print("The absolute population deviation is",m.objval,"person(s).")
print("Oregon has districts =",k)

# Current district status per the population deviation, since the model is␣
↪infeasible
districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]

# print district status
for j in range(k):
    print("Oregon: District",j,"and contains counties",districts[j])
```

```
The absolute population deviation is 131069.0 person(s).
Oregon has districts = 6
Oregon: District 0 and contains counties [0, 16, 7, 32, 33, 20, 30, 19, 34]
Oregon: District 1 and contains counties [22]
Oregon: District 2 and contains counties [14, 26, 3, 9, 13]
Oregon: District 3 and contains counties [17, 31, 11, 35, 12, 23]
Oregon: District 4 and contains counties [8, 2, 29, 24, 28, 25, 27, 15]
Oregon: District 5 and contains counties [18, 1, 6, 4, 21, 5, 10]
```

1.2.5 Step 5: Create model to access the minimization of cut edges using the county to district populations

[ ]:

```python
# data from the county 2010 Census
filepath = '/Users/jessi/Gurobi-Python Playground/OR_county.json'
G = Graph.from_json(filepath)
print(G.nodes)

m = gp.Model()
#check the list of nodes and matches are align with the mapping data
print(G.nodes)
print(G.edges)

for node in G.nodes:
    # using the columns, located the population
    population = G.nodes[node]['TOTPOP']

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

# Impose a 3% population deviation (+/- 0.5%) for the Operations Research model
deviation = 0.03
#number of counties
n
# number of districts
k

# Lower and Upper bound populations to ensure the populations of the districts␣
 ↪do not exceed the bounds
L = math.ceil((1-deviation/2)*total_population/k)
U = math.floor((1+deviation/2)*total_population/k)

# create variables adapted from redistricitng starter:
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
```

```
# objective is to minimize cut edges
m.setObjective( gp.quicksum( 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()

# solve IP model
m.optimize()

# print the optimal minization of cut edges
print("Oregon: The number of cut edges is",m.objval)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 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, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132,
133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148,
149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,
165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180,
181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196,
197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 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, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132,
133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148,
149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,
165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180,
```

181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211]
[(0, 2), (0, 154), (0, 62), (1, 196), (1, 175), (1, 195), (1, 155), (1, 64), (1, 2), (1, 63), (2, 155), (2, 154), (2, 63), (2, 62), (3, 6), (3, 4), (3, 5), (4, 84), (4, 6), (4, 99), (4, 94), (4, 5), (5, 94), (6, 84), (7, 130), (7, 10), (7, 11), (7, 131), (7, 9), (7, 8), (8, 131), (8, 9), (8, 19), (9, 11), (9, 56), (9, 19), (9, 18), (10, 22), (10, 130), (10, 11), (11, 31), (11, 59), (11, 56), (12, 13), (12, 100), (12, 14), (13, 161), (13, 16), (13, 100), (13, 15), (13, 14), (14, 15), (15, 16), (15, 128), (16, 127), (16, 161), (16, 128), (17, 19), (17, 18), (17, 20), (17, 21), (17, 49), (17, 52), (18, 56), (18, 57), (18, 19), (18, 54), (18, 21), (19, 131), (19, 178), (19, 184), (19, 20), (20, 184), (20, 49), (21, 54), (21, 85), (21, 52), (22, 27), (22, 23), (22, 31), (22, 24), (22, 130), (23, 36), (23, 27), (23, 31), (24, 27), (24, 30), (24, 132), (24, 130), (25, 121), (25, 116), (25, 118), (25, 33), (25, 117), (25, 26), (25, 36), (25, 28), (26, 36), (26, 28), (26, 34), (26, 27), (27, 36), (27, 28), (27, 34), (27, 29), (27, 30), (28, 33), (28, 34), (28, 29), (29, 33), (29, 199), (29, 30), (29, 133), (30, 132), (30, 133), (31, 122), (31, 117), (31, 36), (31, 35), (31, 32), (31, 59), (32, 35), (32, 59), (33, 188), (33, 121), (33, 199), (35, 122), (36, 117), (37, 153), (37, 38), (37, 151), (37, 39), (38, 153), (38, 41), (38, 40), (38, 39), (39, 40), (39, 151), (39, 61), (40, 203), (40, 41), (41, 203), (41, 153), (42, 43), (42, 109), (42, 44), (42, 107), (42, 177), (42, 176), (43, 148), (43, 105), (43, 146), (43, 44), (44, 146), (44, 199), (44, 45), (44, 177), (45, 199), (45, 197), (45, 177), (45, 47), (45, 46), (45, 196), (46, 47), (46, 140), (46, 196), (47, 197), (47, 198), (47, 140), (48, 52), (48, 50), (48, 83), (48, 51), (48, 169), (48, 86), (48, 170), (49, 184), (49, 52), (49, 183), (49, 50), (49, 51), (50, 52), (50, 51), (51, 183), (51, 180), (51, 166), (51, 169), (51, 168), (52, 85), (52, 83), (53, 59), (53, 58), (53, 57), (53, 55), (54, 57), (54, 55), (54, 85), (55, 57), (56, 59), (56, 58), (56, 57), (57, 58), (58, 59), (60, 65), (60, 63), (60, 67), (60, 61), (60, 66), (61, 64), (61, 151), (61, 67), (61, 66), (62, 63), (63, 64), (63, 65), (64, 176), (64, 175), (64, 149), (64, 152), (64, 65), (64, 151), (64, 67), (65, 67), (68, 69), (68, 70), (68, 210), (68, 72), (68, 93), (68, 124), (69, 71), (69, 209), (69, 211), (69, 70), (69, 210), (70, 72), (71, 209), (72, 93), (72, 87), (72, 91), (73, 81), (73, 80), (73, 76), (74, 80), (74, 76), (74, 189), (74, 75), (74, 79), (74, 124), (74, 188), (75, 76), (75, 78), (75, 79), (76, 81), (76, 80), (76, 78), (77, 81), (77, 207), (77, 78), (77, 79), (77, 124), (78, 81), (78, 79), (79, 124), (80, 81), (80, 189), (81, 209), (81, 208), (81, 207), (82, 86), (82, 170), (82, 84), (83, 85), (83, 86), (84, 170), (86, 170), (87, 91), (87, 89), (88, 89), (88, 90), (88, 92), (89, 91), (89, 90), (90, 91), (90, 119), (90, 92), (90, 122), (91, 93), (91, 125), (91, 119), (92, 122), (93, 124), (93, 125), (94, 97), (94, 99), (94, 95), (94, 96), (95, 97), (95, 96), (97, 98), (97, 99), (98, 102), (98, 172), (98, 99), (99, 170), (99, 172), (100, 161), (100, 165), (100, 104), (101, 163), (101, 103), (101, 104), (101, 102), (102, 103), (102, 173), (102, 172), (103, 163), (103, 168), (103, 173), (104, 163), (104, 165), (105, 109), (105, 106), (106, 109), (106, 113), (107, 109), (107, 115), (107, 114), (107, 112), (107, 108), (107, 205), (107, 176), (107, 204), (108, 112), (109, 113), (109, 115), (109, 114), (110, 111), (110, 114), (110, 112), (111, 113), (111, 114), (112, 114), (114, 115), (116, 123), (116, 121), (116, 119), (116, 118), (117, 118), (117,

```
122), (118, 119), (118, 122), (119, 125), (119, 123), (119, 122), (120, 124),
(120, 125), (120, 123), (120, 121), (121, 124), (121, 188), (121, 123), (123,
125), (124, 207), (124, 210), (124, 125), (124, 188), (126, 140), (126, 195),
(126, 129), (126, 155), (126, 138), (126, 127), (127, 129), (127, 159), (127,
138), (127, 161), (127, 128), (127, 139), (129, 144), (129, 181), (129, 159),
(130, 132), (130, 131), (131, 132), (131, 134), (131, 136), (131, 178), (132,
137), (132, 133), (132, 134), (133, 199), (133, 137), (133, 134), (133, 142),
(133, 136), (133, 135), (133, 144), (133, 181), (134, 137), (134, 136), (135,
136), (135, 178), (135, 181), (135, 179), (136, 178), (138, 155), (138, 139),
(139, 155), (139, 154), (140, 198), (140, 141), (140, 196), (140, 144), (140,
143), (140, 195), (141, 198), (141, 142), (141, 144), (141, 143), (142, 199),
(142, 200), (142, 144), (143, 144), (144, 181), (145, 191), (145, 148), (145,
147), (146, 148), (146, 187), (146, 147), (146, 188), (146, 199), (147, 191),
(147, 148), (147, 187), (149, 176), (149, 206), (149, 153), (149, 152), (150,
153), (150, 152), (150, 151), (151, 153), (151, 152), (152, 153), (153, 202),
(153, 201), (153, 206), (153, 203), (154, 155), (155, 195), (156, 159), (156,
162), (156, 157), (156, 164), (156, 163), (157, 167), (157, 162), (157, 166),
(157, 163), (158, 160), (158, 159), (158, 162), (159, 181), (159, 160), (159,
162), (159, 161), (159, 164), (159, 163), (159, 165), (160, 181), (160, 185),
(160, 167), (160, 162), (161, 165), (162, 167), (163, 166), (163, 164), (163,
165), (163, 168), (166, 167), (166, 180), (166, 168), (167, 182), (167, 186),
(167, 180), (168, 169), (168, 171), (168, 173), (168, 172), (169, 171), (169,
170), (170, 171), (170, 172), (171, 172), (172, 173), (174, 177), (174, 176),
(174, 196), (174, 175), (175, 176), (175, 196), (176, 177), (176, 204), (176,
206), (177, 196), (178, 179), (178, 184), (179, 181), (179, 185), (179, 184),
(180, 186), (180, 183), (181, 185), (182, 185), (182, 184), (182, 186), (183,
184), (183, 186), (184, 185), (184, 186), (187, 190), (187, 194), (187, 191),
(187, 189), (187, 188), (188, 189), (188, 199), (189, 193), (189, 190), (190,
193), (190, 194), (191, 192), (191, 193), (191, 194), (192, 193), (193, 194),
(195, 196), (197, 199), (197, 200), (197, 198), (198, 200), (199, 200), (201,
205), (201, 204), (201, 202), (201, 206), (202, 205), (202, 203), (204, 205),
(204, 206), (207, 208), (207, 211), (207, 210), (208, 209), (208, 211), (209,
211), (210, 211)]
3831074
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[arm])

CPU model: Apple M2
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 3572 rows, 1830 columns and 13860 nonzeros
Model fingerprint: 0xb7d1b581
Variable types: 0 continuous, 1830 integer (1830 binary)
Coefficient statistics:
  Matrix range     [1e+00, 5e+05]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 6e+05]
Presolve time: 0.03s
```

```
Presolved: 3572 rows, 1830 columns, 13860 nonzeros
Variable types: 0 continuous, 1830 integer (1830 binary)

Root relaxation: objective 0.000000e+00, 2181 iterations, 0.38 seconds (0.21
work units)

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

     0     0    0.00000    0 1272          -    0.00000      -     -    1s
H    0     0                     408.0000000    0.00000   100%     -    1s
H    0     0                     378.0000000    0.00000   100%     -    1s
     0     0    2.53618    0 1278  378.00000    2.53618  99.3%     -    3s
H    0     0                     373.0000000    2.53618  99.3%     -    3s
     0     0    5.01059    0 1275  373.00000    5.01059  98.7%     -    4s
     0     0    5.10652    0 1275  373.00000    5.10652  98.6%     -    5s
H    0     0                     370.0000000    5.10652  98.6%     -    5s
H    0     0                     364.0000000    5.10652  98.6%     -    5s
     0     0    5.10933    0 1275  364.00000    5.10933  98.6%     -    5s
     0     0    5.30570    0 1276  364.00000    5.30570  98.5%     -    6s
H    0     0                     332.0000000    5.30570  98.4%     -    6s
     0     0    5.33643    0 1273  332.00000    5.33643  98.4%     -    6s
     0     0    5.38135    0 1274  332.00000    5.38135  98.4%     -    7s
H    0     0                     213.0000000    5.38135  97.5%     -    8s
     0     0    5.61900    0 1273  213.00000    5.61900  97.4%     -    8s
     0     0    5.61900    0 1274  213.00000    5.61900  97.4%     -    8s
     0     0    5.61900    0 1274  213.00000    5.61900  97.4%     -    9s
     0     0    5.64039    0 1274  213.00000    5.64039  97.4%     -    9s
     0     0    5.64962    0 1275  213.00000    5.64962  97.3%     -    9s
     0     0    5.64962    0 1275  213.00000    5.64962  97.3%     -    9s
     0     0    5.66493    0 1273  213.00000    5.66493  97.3%     -   10s
H    0     0                     198.0000000    5.66493  97.1%     -   10s
     0     0    5.66493    0 1275  198.00000    5.66493  97.1%     -   10s
     0     0    5.66493    0 1275  198.00000    5.66493  97.1%     -   10s
H    0     0                     191.0000000    5.66493  97.0%     -   10s
     0     0    5.66493    0 1275  191.00000    5.66493  97.0%     -   11s
H    0     0                     190.0000000    5.66493  97.0%     -   11s
     0     0    5.66493    0 1276  190.00000    5.66493  97.0%     -   11s
     0     0    5.66493    0 1275  190.00000    5.66493  97.0%     -   12s
H    0     0                     173.0000000    5.66493  96.7%     -   12s
     0     0    5.66493    0 1276  173.00000    5.66493  96.7%     -   12s
     0     0    5.66493    0 1274  173.00000    5.66493  96.7%     -   13s
     0     0    5.66493    0 1274  173.00000    5.66493  96.7%     -   13s
H    0     0                     172.0000000    5.66493  96.7%     -   14s
     0     2    5.66493    0 1274  172.00000    5.66493  96.7%     -   14s
     1     4    5.70991    1 1273  172.00000    5.70991  96.7%   377   15s
    19    22   12.19095    5 1292  172.00000   10.97884  93.6%   958   20s
H   27    30                     170.0000000   10.98100  93.5%  1315   22s
```

H	28	30				164.0000000	10.98100	93.3%	1325	22s
H	30	30				160.0000000	10.98100	93.1%	1340	22s
	53	54	17.17788	8	1084	160.00000	10.98100	93.1%	1459	25s
H	61	62				153.0000000	10.98100	92.8%	1401	26s
H	67	62				147.0000000	10.98100	92.5%	1431	26s
H	100	99				142.0000000	10.98100	92.3%	1358	30s
H	103	99				131.0000000	10.98100	91.6%	1361	30s
H	139	133				123.0000000	10.98100	91.1%	1285	34s
H	143	133				119.0000000	10.98100	90.8%	1260	34s
	146	144	26.20348	13	964	119.00000	10.98100	90.8%	1258	35s
H	148	144				118.0000000	10.98100	90.7%	1253	35s
H	150	144				114.0000000	10.98100	90.4%	1264	35s
H	175	178				112.0000000	10.98100	90.2%	1174	37s
H	180	178				110.0000000	10.98100	90.0%	1163	37s
H	199	191				109.0000000	10.98100	89.9%	1115	38s
	226	227	26.19656	15	967	109.00000	10.98100	89.9%	1058	47s
H	227	227				105.0000000	10.98100	89.5%	1053	47s
H	238	227				102.0000000	10.98100	89.2%	1031	47s
H	269	285				101.0000000	10.98100	89.1%	987	50s
H	288	285				100.0000000	10.98100	89.0%	957	50s
H	346	347				99.0000000	10.98100	88.9%	880	52s
	406	401	31.46402	20	875	99.00000	10.98100	88.9%	811	55s
H	421	401				95.0000000	10.98100	88.4%	805	55s
H	431	401				89.0000000	10.98100	87.7%	802	55s
H	486	466				88.0000000	10.98100	87.5%	767	57s
H	501	466				87.0000000	10.98100	87.4%	756	57s
	544	495	41.08958	25	917	87.00000	10.98100	87.4%	753	62s
H	549	484				84.0000000	10.98100	86.9%	755	62s
H	607	525				78.0000000	10.98100	85.9%	739	67s
	674	555	51.28433	30	1027	78.00000	10.98100	85.9%	725	71s
	755	652	58.50274	39	986	78.00000	10.98100	85.9%	735	76s
	885	760	71.85941	55	884	78.00000	11.65128	85.1%	711	81s
	984	813	17.74359	6	1286	78.00000	11.65128	85.1%	698	86s
	1060	888	22.45636	10	1271	78.00000	11.65128	85.1%	698	90s
	1208	1041	27.67607	14	1085	78.00000	11.65128	85.1%	690	97s
H	1210	1035				77.0000000	11.65128	84.9%	690	97s
	1289	1095	26.36601	15	1014	77.00000	11.65128	84.9%	674	100s
	1460	1171	60.67746	51	1266	77.00000	11.65128	84.9%	649	106s
	1463	1173	28.42284	11	1273	77.00000	11.65128	84.9%	648	110s
	1468	1177	35.70242	25	1271	77.00000	11.65128	84.9%	646	116s
	1474	1181	55.56659	20	1290	77.00000	11.65128	84.9%	643	120s
	1478	1186	12.00619	14	1287	77.00000	11.65128	84.9%	655	125s
	1484	1196	24.68984	16	1098	77.00000	11.65128	84.9%	664	131s
	1500	1207	28.70150	17	1078	77.00000	12.03636	84.4%	682	135s
	1524	1223	29.15261	19	1244	77.00000	12.34477	84.0%	703	140s
	1562	1251	32.46566	21	1275	77.00000	12.34477	84.0%	719	145s
	1597	1278	38.21335	23	999	77.00000	12.34477	84.0%	728	150s
	1687	1348	44.79227	26	833	77.00000	12.34477	84.0%	723	155s

```
  1767  1375   48.10591   29   807    77.00000    12.34477  84.0%   709  162s
H 1778  1311                         75.0000000    12.34477  83.5%   707  162s
H 1815  1290                         74.0000000    12.34477  83.3%   701  164s
  1837  1329   49.16924   33   589    74.00000    12.34477  83.3%   698  165s
H 1904  1268                         73.0000000    12.34477  83.1%   686  168s
  1957  1335   60.86902   39   587    73.00000    12.34477  83.1%   676  171s
  2116  1389   67.62673   49   769    73.00000    12.34477  83.1%   650  175s
  2214  1416   28.25583   19  1066    73.00000    18.39214  74.8%   646  180s
  2324  1473   45.97738   26  1069    73.00000    18.39214  74.8%   639  185s
  2402  1494   60.54922   32   998    73.00000    18.39214  74.8%   636  191s
H 2403  1424                         72.0000000    18.39214  74.5%   635  191s
  2446  1473   70.37679   36   887    72.00000    19.26548  73.2%   638  196s
  2542  1547   36.44970   22  1110    72.00000    19.26548  73.2%   634  200s
  2603  1541   41.29385   25   992    72.00000    19.26548  73.2%   629  214s
  2624  1569   42.14809   26   923    72.00000    19.26548  73.2%   627  217s
  2686  1591   51.04369   32   899    72.00000    19.26548  73.2%   625  220s
  2792  1666   67.33191   54   858    72.00000    19.26548  73.2%   623  225s
  2921  1682   27.27647   22  1183    72.00000    21.30443  70.4%   618  235s
  3005  1750   38.44172   29  1043    72.00000    21.30443  70.4%   619  242s
  3053  1804   43.74192   34   972    72.00000    21.30443  70.4%   617  246s
  3220  1899   60.37663   62   793    72.00000    21.50653  70.1%   609  253s
  3320  1939   30.38479   22  1118    72.00000    21.50653  70.1%   603  256s
  3418  1913   36.95148   28  1076    72.00000    21.50653  70.1%   599  289s
  3428  1938   37.86978   29  1154    72.00000    21.50653  70.1%   599  296s
```

[24]:
```python
# Population Status
print("Oregon: Using Lower Bound of Population =",L,"and Upper Bound of Population =",U,"and districts =",k)
print("Oregon'population deviation is greater than 100; therefore, the model is assumed to be infeasible. This redistricting of OR is a more complex tasking due to counties being larger than others. The model proves the infeasibility.")
```

```
Oregon: Using Lower Bound of Population = 628935 and Upper Bound of Population =
648090 and districts = 6
Oregon'population deviation is greater than 100; therefore, the model is assumed
to be infeasible. This redistricting of OR is a more complex tasking due to
counties being larger than others. The model proves the infeasibility.
```

1.2.6 Step 6: Retrieve the districts and their populations

[25]:
```python
districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
district_populations = [ sum(G.nodes[i]["TOTPOP"] for i in districts[j]) for j in range(k) ]

# print district status
for j in range(k):
```

```
    print("Oregon: District",j,"has population",district_populations[j],"and␣
↪contains counties",districts[j])
```

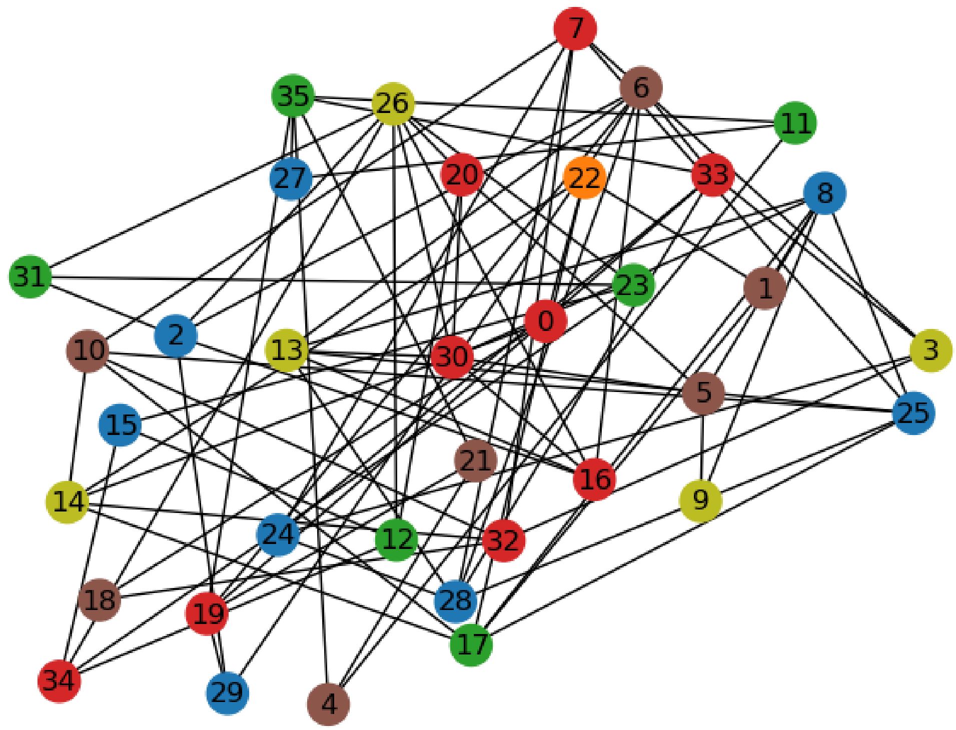
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[25], line 1
----> 1 districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
      2 district_populations = [ sum(G.nodes[i]["TOTPOP"] for i in districts[j]) ␣
↪for j in range(k) ]
      4 # print district status

Cell In[25], line 1, in <listcomp>(.0)
----> 1 districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
      2 district_populations = [ sum(G.nodes[i]["TOTPOP"] for i in districts[j]) ␣
↪for j in range(k) ]
      4 # print district status

Cell In[25], line 1, in <listcomp>(.0)
----> 1 districts = [ [i for i in G.nodes if x[i,j].x > 0.5] for j in range(k)]
      2 district_populations = [ sum(G.nodes[i]["TOTPOP"] for i in districts[j]) ␣
↪for j in range(k) ]
      4 # print district status

File src/gurobipy/var.pxi:125, in gurobipy.Var.__getattr__()

File src/gurobipy/var.pxi:153, in gurobipy.Var.getAttr()

File src/gurobipy/attrutil.pxi:100, in gurobipy.__getattr()

AttributeError: Unable to retrieve attribute 'x'
```

1.2.7 Step 7: Node Mapping the counties and districts

[16]:
```
filepath = '/Users/jessi/Gurobi-Python Playground/OR.dimacs-mod.txt'
G = nx.read_edgelist(filepath,nodetype=int)

print(type(G))

districts = ["District 0","District 1","District 2","District 3","District␣
↪4","District 5"]

# Assigning a dictionary for the districts
counties_district_dict = {
"District 0": [0, 16, 7, 32, 33, 20, 30, 19, 34],
"District 1":  [22],
"District 2":  [14, 26, 3, 9, 13],
```

```
"District 3":  [17, 31, 11, 35, 12, 23],
"District 4":  [8, 2, 29, 24, 28, 25, 27, 15],
"District 5":  [18, 1, 6, 4, 21, 5, 10]
}

# Assigning a color to each district
counties_district_colors_dict = {
    "District 0": "tab:red",
    "District 1": "tab:orange",
    "District 2": "tab:olive",
    "District 3": "tab:green",
    "District 4": "tab:blue",
    "District 5": "tab:brown",

}



# Assigning the colors
color_list = []
for node in G.nodes:   # iteration over nodes
    if node in counties_district_dict['District 0']:
        color_list.append(counties_district_colors_dict['District 0'])

    elif node in counties_district_dict['District 1']:
        color_list.append(counties_district_colors_dict['District 1'])

    elif node in counties_district_dict['District 2']:
        color_list.append(counties_district_colors_dict['District 2'])

    elif node in counties_district_dict['District 3']:
        color_list.append(counties_district_colors_dict['District 3'])

    elif node in counties_district_dict['District 4']:
        color_list.append(counties_district_colors_dict['District 4'])

    else:
        color_list.append(counties_district_colors_dict['District 5'])

# Map output
nx.draw(G, pos, node_color=color_list, with_labels=True)
plt.show()
```

```
<class 'networkx.classes.graph.Graph'>
```

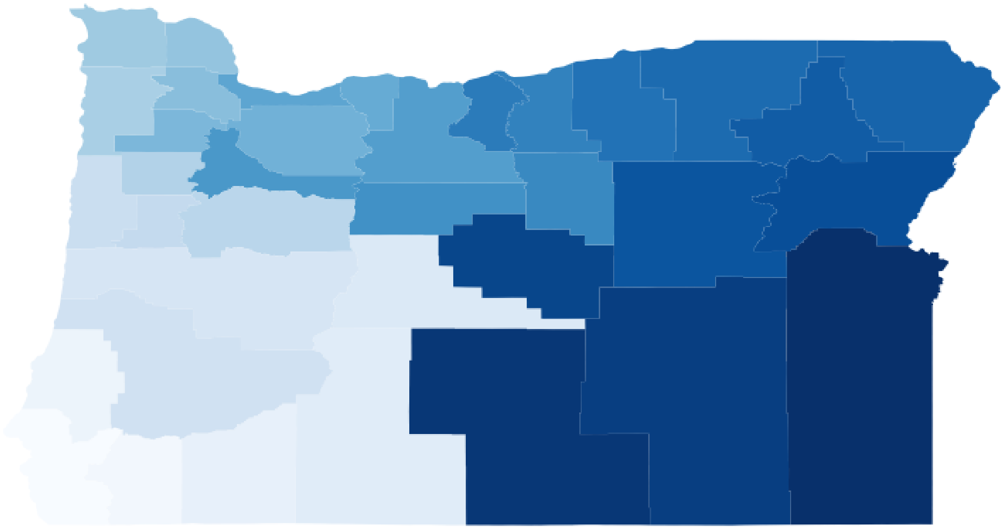


1.2.8 Step 8: State shape Mapping the counties

[17]:
```
# Read Oregon county shapefile and distance files
file1 = '/Users/jessi/Gurobi-Python Playground/OR_distances.csv'
file2 = '/Users/jessi/Gurobi-Python Playground/OR_counties.shx'
file = '/Users/jessi/Gurobi-Python Playground/OR_counties.shp'
#create the data frame
df = gpd.read_file(file)

#checker
print((df))
print(type(df))


# Mapping
fig,ax = plt.subplots(figsize=(10,10))
df.plot(ax=ax, column = "geometry", cmap ='Blues')
ax.axis("off")
print("Oregon's counties are shown by varied blues in the diagram")
```

```
geometry
```

```
0   POLYGON ((-123.35949 46.01121, -123.35959 46.0…
1   POLYGON ((-121.63812 45.70477, -121.63132 45.7…
2   POLYGON ((-119.89506 43.56813, -119.89506 43.5…
3   POLYGON ((-124.14651 44.77862, -124.14609 44.7…
4   POLYGON ((-124.00393 42.45154, -124.00387 42.4…
5   POLYGON ((-120.36083 45.50569, -120.36080 45.5…
6   POLYGON ((-121.40550 44.39315, -121.40256 44.3…
7   POLYGON ((-123.72466 44.98427, -123.72462 44.9…
8   POLYGON ((-121.71160 45.15282, -121.71222 45.1…
9   POLYGON ((-120.82708 45.51531, -120.82814 45.5…
10  POLYGON ((-123.46310 45.28019, -123.46310 45.2…
11  POLYGON ((-124.54413 42.68382, -124.55310 42.6…
12  POLYGON ((-119.14139 45.93100, -119.13504 45.9…
13  POLYGON ((-121.14964 44.39065, -121.15085 44.3…
14  POLYGON ((-123.20927 45.43371, -123.20976 45.4…
15  POLYGON ((-117.74746 45.83324, -117.74745 45.8…
16  POLYGON ((-121.01452 44.39032, -121.01311 44.3…
17  POLYGON ((-122.74374 45.33207, -122.74376 45.3…
18  POLYGON ((-124.00807 46.01053, -124.00791 46.0…
19  POLYGON ((-121.31649 41.99706, -121.31761 41.9…
20  POLYGON ((-119.99817 45.68439, -119.99815 45.6…
21  POLYGON ((-122.95119 42.76756, -122.95120 42.7…
22  POLYGON ((-122.86749 45.60623, -122.86830 45.6…
23  POLYGON ((-117.94624 45.00670, -117.94605 45.0…
24  POLYGON ((-123.18202 44.72008, -123.17725 44.7…
25  POLYGON ((-122.88251 45.25531, -122.88224 45.2…
26  POLYGON ((-119.65318 44.76270, -119.65319 44.7…
27  POLYGON ((-124.46746 43.19648, -124.46524 43.2…
28  POLYGON ((-123.25099 44.55198, -123.25240 44.5…
29  POLYGON ((-120.88050 42.42859, -120.88049 42.4…
30  POLYGON ((-120.38569 44.76118, -120.38572 44.7…
31  POLYGON ((-118.23029 43.77921, -118.22811 43.7…
32  POLYGON ((-124.01620 45.64610, -124.01619 45.6…
33  POLYGON ((-123.73473 43.94406, -123.73927 43.9…
34  POLYGON ((-118.11635 45.49884, -118.11636 45.4…
35  POLYGON ((-123.81715 43.45959, -123.81710 43.4…
<class 'geopandas.geodataframe.GeoDataFrame'>
Oregon's counties are shown by varied blues in the diagram
```



[ ]:
```
#Trial and Error
# #for loop examples and set-up
# ## x is the place holder that looks in the range
# for x in range(1, 11):
#     print(x)
# print("Range")

# #for loop counting backwards by two which is the step function
# for x in reversed (range(1, 11, 2)):
#     print(x)
# print("Backward")

# #printing within a sequence
# ## x holds current position
# Number = "1234-1976-3467"
# for x in Number:
#     print(x)
# print("Number sequence")

# #skipping a number and looping
# for x in range(1,21):
#     if x == 13:
#         continue
#     else:
#         print(x)

# #skipping a number and looping
```

```
# for x in range(1,21):
#     if x == 13:
#         break
#     else:
#         print(x)

# # Nested for loop examples
# # outer and inner loop, make sure the varible/placeholder/ counters are
 ↪different

# #call and response
# rows =int(input("Enter the # of rows: "))
# columns =int(input("Enter the # of columns: "))
# symbol = input("Enter a symbol to use: ")
# for x in range(rows): # means 3 iterations, outer loop
# # everytime you print it ends with a new character and proceeds to the next
 ↪line,
# # so if we use a string then it will be next to each other
#     for y in range(columns):
#         print (symbol, end =" ")
#     print() #exit inner loop

# # number string
# for x in range(3): # means 3 iterations, outer loop
# # everytime you print it ends with a new character and proceeds to the next
 ↪line,
# # so if we use a string then it will be next to each other
#     for y in range(1,10):
#         print (y, end ="")
#     print() #exit inner loop

####################
# If using the unmodified .dimacs file, it will need to be manipulated to only
 ↪utilize integers. Removing column 1, row 1, last row
# filepath = "/Users/jessi/Gurobi-Python Playground/MA.dimacs"
# file = open(filepath)

# #remove the header and tail to over mixing int and str
# next(file)
# last_line = None
# for line in file:
#     if last_line:
#         for line in file:
#             if line.strip():
#                 file.write("\t".join(line.split()[1:]) + "\n")
#         print (last_line)
#         last_line = line
```

```
# # for line in file:
# #         try:
# #             r = int(line)
# #             if r > 999 or line.strip() == '-0':
# #                 #filtering numbers >999 and strings with '-0'
# #                 continue
# #             file.append(r)
# #         except ValueError:
# #             pass

# # Create List of integers
# # nodes_edges = []
# # for each in file:
# #     each = each.strip()
# #     #split the list so the populations in each range
# #     each = each.split(' ')
# #     print((each[1], each[2]))
# #     nodes_edges.append(int(each[1], each[2]))
#####################

#shifts = range(1,9000000)
# for district in districts:
#     c_counties = counties_district_dict[district]
#     c_color = counties_district_dict[district]
```