Homework 8 Question 1: Voting

Governor Blue of the state of Berry is attempting to get the state legislator to gerrymander Berry's congressional districts. The state consists of ten cities, and the numbers of registered Republicans and Democrats (in thousands) in each city are shown below

City	Republicans	Democrats
1	80	34
2	60	44
3	40	44
4	20	24
5	40	114
6	40	64
7	70	14
8	50	44
9	70	54
10	70	64

Berry has five congressional representatives. To form the five congressional districts, cities must be grouped together according to the following restrictions:

- Districts cannot subdivide cities; all voters in a city must be in the same district.
- Each district must contain between 150,000 and 250,000 voters (there are no independent voters).

Governor Blue is a Democrat. Assume 100% voter turnout and that each voter always votes according to their registered party. Formulate and solve an optimization problem to help Governor Blue maximize the number of congressional districts that have a Democratic majority.

Problem Data

In [1]:
```
cities = [:one, :two, :three, :four, :five, :six, :seven, :eight, :nine, :ten]
raw_repub = [80, 60, 40, 20, 40, 40, 70, 50, 70, 70]'
republicans = Dict(zip(cities, raw_repub))
raw_demo = [34, 44, 44, 24, 114, 64, 14, 44, 54, 64]'
democrats = Dict(zip(cities, raw_demo))
n_cities = length(cities)
n_districts = 5;
```

Problem Model

In [2]:
```
using JuMP, Cbc

m = Model(solver=CbcSolver())

@variable(m, x[1:n_cities, 1:n_districts], Bin)
@variable(m, majority[1:n_districts], Bin)

@constraint(m, cityToOneDistrict[i=1:n_cities], sum(x[i,j] for j in 1:n_districts) == 1)

@constraint(m, votersMin[j=1:n_districts], sum(x[i,j]*
                    (democrats[cities[i]] + republicans[cities[i]])
                    for i in 1:n_cities) >= 150)
@constraint(m, votersMax[j=1:n_districts], sum(x[i,j]*
                                (democrats[cities[i]] + republicans[cities[i]])
                    for i in 1:n_cities) <= 250)

# If There is majority then # of democrats are more than # of republicans
@constraint(m, majorityConstr[j=1:n_districts], sum(x[i,j]*
                                            ( republicans[cities[i]] - democrats[cities[i]])
                    for i in 1:n_cities) <= 250(1-majority[j]))
@objective(m, Max, sum(majority))
m
```

Out[2]:

$$
\begin{aligned}
\max \quad & majority_1 + majority_2 + majority_3 + majority_4 + majority_5 \\
\text{Subject to} \quad & x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} + x_{1,5} = 1 \\
& x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} + x_{2,5} = 1 \\
& x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} + x_{3,5} = 1 \\
& x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} + x_{4,5} = 1 \\
& x_{5,1} + x_{5,2} + x_{5,3} + x_{5,4} + x_{5,5} = 1 \\
& x_{6,1} + x_{6,2} + x_{6,3} + x_{6,4} + x_{6,5} = 1 \\
& x_{7,1} + x_{7,2} + x_{7,3} + x_{7,4} + x_{7,5} = 1 \\
& x_{8,1} + x_{8,2} + x_{8,3} + x_{8,4} + x_{8,5} = 1 \\
& x_{9,1} + x_{9,2} + x_{9,3} + x_{9,4} + x_{9,5} = 1 \\
& x_{10,1} + x_{10,2} + x_{10,3} + x_{10,4} + x_{10,5} = 1 \\
& 114x_{1,1} + 104x_{2,1} + 84x_{3,1} + 44x_{4,1} + 154x_{5,1} + 104x_{6,1} + 84x_{7,1} + 94x_{8,1} + 124x_{9,1} + 134x_{10,1} \geq 150 \\
& 114x_{1,2} + 104x_{2,2} + 84x_{3,2} + 44x_{4,2} + 154x_{5,2} + 104x_{6,2} + 84x_{7,2} + 94x_{8,2} + 124x_{9,2} + 134x_{10,2} \geq 150 \\
& 114x_{1,3} + 104x_{2,3} + 84x_{3,3} + 44x_{4,3} + 154x_{5,3} + 104x_{6,3} + 84x_{7,3} + 94x_{8,3} + 124x_{9,3} + 134x_{10,3} \geq 150 \\
& 114x_{1,4} + 104x_{2,4} + 84x_{3,4} + 44x_{4,4} + 154x_{5,4} + 104x_{6,4} + 84x_{7,4} + 94x_{8,4} + 124x_{9,4} + 134x_{10,4} \geq 150 \\
& 114x_{1,5} + 104x_{2,5} + 84x_{3,5} + 44x_{4,5} + 154x_{5,5} + 104x_{6,5} + 84x_{7,5} + 94x_{8,5} + 124x_{9,5} + 134x_{10,5} \geq 150 \\
& 114x_{1,1} + 104x_{2,1} + 84x_{3,1} + 44x_{4,1} + 154x_{5,1} + 104x_{6,1} + 84x_{7,1} + 94x_{8,1} + 124x_{9,1} + 134x_{10,1} \leq 250 \\
& 114x_{1,2} + 104x_{2,2} + 84x_{3,2} + 44x_{4,2} + 154x_{5,2} + 104x_{6,2} + 84x_{7,2} + 94x_{8,2} + 124x_{9,2} + 134x_{10,2} \leq 250 \\
& 114x_{1,3} + 104x_{2,3} + 84x_{3,3} + 44x_{4,3} + 154x_{5,3} + 104x_{6,3} + 84x_{7,3} + 94x_{8,3} + 124x_{9,3} + 134x_{10,3} \leq 250 \\
& 114x_{1,4} + 104x_{2,4} + 84x_{3,4} + 44x_{4,4} + 154x_{5,4} + 104x_{6,4} + 84x_{7,4} + 94x_{8,4} + 124x_{9,4} + 134x_{10,4} \leq 250 \\
& 114x_{1,5} + 104x_{2,5} + 84x_{3,5} + 44x_{4,5} + 154x_{5,5} + 104x_{6,5} + 84x_{7,5} + 94x_{8,5} + 124x_{9,5} + 134x_{10,5} \leq 250 \\
& 46x_{1,1} + 16x_{2,1} - 4x_{3,1} - 4x_{4,1} - 74x_{5,1} - 24x_{6,1} + 56x_{7,1} + 6x_{8,1} + 16x_{9,1} + 6x_{10,1} + 250majority_1 \leq 250 \\
& 46x_{1,2} + 16x_{2,2} - 4x_{3,2} - 4x_{4,2} - 74x_{5,2} - 24x_{6,2} + 56x_{7,2} + 6x_{8,2} + 16x_{9,2} + 6x_{10,2} + 250majority_2 \leq 250 \\
& 46x_{1,3} + 16x_{2,3} - 4x_{3,3} - 4x_{4,3} - 74x_{5,3} - 24x_{6,3} + 56x_{7,3} + 6x_{8,3} + 16x_{9,3} + 6x_{10,3} + 250majority_3 \leq 250 \\
& 46x_{1,4} + 16x_{2,4} - 4x_{3,4} - 4x_{4,4} - 74x_{5,4} - 24x_{6,4} + 56x_{7,4} + 6x_{8,4} + 16x_{9,4} + 6x_{10,4} + 250majority_4 \leq 250 \\
& 46x_{1,5} + 16x_{2,5} - 4x_{3,5} - 4x_{4,5} - 74x_{5,5} - 24x_{6,5} + 56x_{7,5} + 6x_{8,5} + 16x_{9,5} + 6x_{10,5} + 250majority_5 \leq 250 \\
& x_{i,j} \in \{0,1\} \quad \forall i \in \{1,2,\dots,9,10\}, j \in \{1,2,3,4,5\} \\
& majority_i \in \{0,1\} \quad \forall i \in \{1,2,3,4,5\}
\end{aligned}
$$

In [3]:
```
status = solve(m)
println("Status: ",status)
println("Majority in districts ", getvalue(majority))
assignment = getvalue(x)
println("Assignment of cities to districts")
[println(assignment[:,j]) for j in 1:n_districts]
println("# of Voters per district, # of Democrats, # of Republicans")
[println(raw_demo*assignment[:,j] + raw_repub*assignment[:,j],"\t",
        (raw_demo*assignment[:,j])[1],"\t",
        (raw_repub*assignment[:,j])[1])
    for j in 1:n_districts];
```

```
Status: Optimal
Majority in districts [1.0,0.0,0.0,1.0,1.0]
Assignment of cities to districts
[0.0,0.0,1.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0]
[1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0]
[0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0]
[0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0]
[0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0]
# of Voters per district, # of Democrats, # of Republicans
[222.0] 112.0   110.0
[238.0] 88.0    150.0
[188.0] 58.0    130.0
[154.0] 114.0   40.0
[238.0] 128.0   110.0
```

Homework 8 Question 2: Paint production.

As part of its weekly production, a paint company produces five batches of paints, always the same, for some big clients who have a stable demand. Every paint batch is produced in a single production process, all in the same blender that needs to be cleaned between each batch. The durations of blending paint batches 1 to 5 are 40, 35, 45, 32 and 50 minutes respectively. The cleaning times depend of the colors and the paint types. For example, a long cleaning period is required if an oil-based paint is produced after a water-based paint, or to produce white paint after a dark color. The times are given in minutes in the following matrix A where A_{ij} denotes the cleaning time after batch i if it is followed by batch j.

$$A = \begin{bmatrix} 0 & 11 & 7 & 13 & 11 \\ 5 & 0 & 13 & 15 & 15 \\ 13 & 15 & 0 & 23 & 11 \\ 9 & 13 & 5 & 0 & 3 \\ 3 & 7 & 7 & 7 & 0 \end{bmatrix}$$

Since the company has other activities, it wishes to deal with this weekly production in the shortest possible time (blending and cleaning). What is the corresponding order of paint batches? The order will be applied every week, so the cleaning time between the last batch of one week and the first of the following week needs to be accounted for in the total duration of cleaning.

Problem Data

In [4]:
```
using NamedArrays
batches = [:one, :two, :three, :four, :five]
blendTimes = Dict(zip(batches,[40, 35, 45, 32 ,50]))
cleaningTimes = [ 0 11  7 13 11 ;
                  5  0 13 15 15 ;
                 13 15  0 23 11 ;
                  9 13  5  0  3 ;
                  3  7  7  7  0 ]
c = NamedArray(cleaningTimes,(batches,batches))
N = size(cleaningTimes,1);
```

Problem Model

In [10]:
```
using JuMP, Cbc

m = Model(solver = CbcSolver())
@variable(m, x[batches,batches], Bin)                                      # must formulate as IP this time
@constraint(m, c1[j in batches], sum( x[i,j] for i in batches ) == 1)      # one out-edge
@constraint(m, c2[i in batches], sum( x[i,j] for j in batches ) == 1)      # one in-edge
@constraint(m, c3[i in batches], x[i,i] == 0 )                            # no self-loops
@objective(m, Min, sum( x[i,j]*(c[i,j] + blendTimes[j]) for i in batches, j in batches ))   # minimize total cos

# MTZ variables and constraints
@variable(m, u[batches])
@constraint(m, c4[i in batches, j in batches[2:end]], u[i] - u[j] + N*x[i,j] <= N-1 );
```

In [11]:
```
status = solve(m)
println("Status: ", status)
xx = getvalue(x)
subtours = getAllSubtours(xx)
println("Total length of cleaning and blending: ", getobjectivevalue(m))
println(subtours)
# pretty print the solution
sol = NamedArray(zeros(Int,N,N),(batches,batches))
for i in batches
    for j in batches
        sol[i,j] = Int(xx[i,j])
    end
end
println(sol)
```

```
Status: Optimal
Total length of cleaning and blending: 243.0
Any[Symbol[:one,:four,:three,:five,:two,:one]]
5×5 Named Array{Int64,2}
A ╲ B │   one    two  three   four   five
──────┼──────────────────────────────────
one   │     0      0      0      1      0
two   │     1      0      0      0      0
three │     0      0      0      0      1
four  │     0      0      1      0      0
five  │     0      1      0      0      0
```

In [1]:
```
# HELPER FUNCTION: returns the cycle containing the city START.
function getSubtour(x,start)
    subtour = [start]
    while true
        j = subtour[end]
        for k in batches
            if x[j,k] == 1
                push!(subtour,k)
                break
            end
        end
        if subtour[end] == start
            break
        end
    end
    return subtour
end

# HELPER FUNCTION: returns a list of all cycles
function getAllSubtours(x)
    nodesRemaining = batches
    subtours = []
    while length(nodesRemaining) > 0
        subtour = getSubtour(x,nodesRemaining[1])
        push!(subtours, subtour)
        nodesRemaining = setdiff(nodesRemaining,subtour)
    end
    return subtours
end;
```

Homework 8 Question 3: The Queens problem.

You are given a standard 8×8 chess board. The following problems involve placing queens on the board such that certain constraints are satisfied. For each of the following problems, model the optimization task as an integer program, solve it, and show what an optimal placement of queens on the board looks like.

a) Find a way to place 8 queens on the board so that no two queens threaten each other. We say that two queens threaten each other if they occupy the same row, column, or diagonal. Show what this placement looks like.

In [1]:

```
# helper function to print a chess board
function printChessBoard(arr)
  u = 0
  println("+---+---+---+---+---+---+---+---+")
    for r in 1:8
        for c in 1:8
            if arr[r,c] == 1
                print("| ",round(Int,arr[r,c])," ")
            else
                print("|  "," ")
            end
        end
        println("|")
        println("+---+---+---+---+---+---+---+---+")
    end
end
;
```

In [2]:

```
# Helper function to generate neihbours
# Operations allowed to get neighbor of a cell.
# First column element is for row and second column for column operation
ops = [ 1  1;
       -1  1;
        1 -1;
       -1 -1]

function get_diag_nbr(i, j, rows, columns)
    retval = []
    for o in 1:length(ops[:,1])
        for mul in 1:8
            if i + ops[o,1]*mul <= rows && i + ops[o,1]*mul >= 1
                if j + ops[o,2]*mul <= columns && j + ops[o,2]*mul >= 1
                    push!(retval,(i + ops[o,1]*mul, j + ops[o,2]*mul))
                end
            end
        end
    end
    return retval
end
# Helper function to generate neihbours
# Operations allowed to get neighbor of a cell.
# First column element is for row and second column for column operation
all_ops = [ 1  1;
       -1  1;
        1 -1;
       -1 -1;
        0  1;
        0 -1;
        1  0;
       -1  0]

function get_all_nbr(i, j, rows, columns)
    retval = []
    for o in 1:length(all_ops[:,1])
        for mul in 1:8
            if i + all_ops[o,1]*mul <= rows && i + all_ops[o,1]*mul >= 1
                if j + all_ops[o,2]*mul <= columns && j + all_ops[o,2]*mul >= 1
                    push!(retval,(i + all_ops[o,1]*mul, j + all_ops[o,2]*mul))
                end
            end
        end
    end
    return retval
end;
```

In [3]:
```
println("Diagonal Neighbours for (4,4) as example")
[println("(",i,",",j,"): ",get_diag_nbr(i,j, 8,8)) for i in 4 for j in 4]
println("All Neighbours for (4,4) as example")
[println("(",i,",",j,"): ",get_all_nbr(i,j, 8,8)) for i in 4 for j in 4];;
```

```
Diagonal Neighbours for (4,4) as example
(4,4): Any[(5,5),(6,6),(7,7),(8,8),(3,5),(2,6),(1,7),(5,3),(6,2),(7,1),(3,3),(2,2),(1,1)]
All Neighbours for (4,4) as example
(4,4): Any[(5,5),(6,6),(7,7),(8,8),(3,5),(2,6),(1,7),(5,3),(6,2),(7,1),(3,3),(2,2),(1,1),(4,5),(4,6),(4,7),
(4,8),(4,3),(4,2),(4,1),(5,4),(6,4),(7,4),(8,4),(3,4),(2,4),(1,4)]
```

In [4]:
```
using JuMP, Cbc

m = Model(solver = CbcSolver())

@variable(m, x[1:8,1:8], Bin)

# exactly one of each number per row
@constraint(m, rowC[i=1:8], sum(x[i,j] for j in 1:8) == 1)

# exactly one of each number per column
@constraint(m, colC[j=1:8], sum(x[i,j] for i in 1:8) == 1)

# if the sum of queens in the diagonals is 1 then this cell cannot contain queen
@constraint(m, diagC[i=1:8,j=1:8], sum(x[i_n,j_n] for (i_n,j_n) in get_diag_nbr(i,j,8,8)) - 1
                                    <= 1 - x[i,j] - x[i,j]);
```

In [5]:
```
status = solve(m)
println("Status: ", status)
board = getvalue(x)
printChessBoard(board);
```

```
Status: Optimal
+---+---+---+---+---+---+---+---+
|   |   |   |   |   | 1 |   |   |
+---+---+---+---+---+---+---+---+
|   |   | 1 |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | 1 |   |
+---+---+---+---+---+---+---+---+
|   | 1 |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 1 |
+---+---+---+---+---+---+---+---+
|   |   |   |   | 1 |   |   |   |
+---+---+---+---+---+---+---+---+
| 1 |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   | 1 |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

b) Repeat part (a) but this time find a placement of the 8 queens that has point symmetry. In other words, find a placement that looks the same if you rotate the board 180◦.

In [6]:
```
m = Model(solver = CbcSolver())

@variable(m, x[1:8,1:8], Bin)

# exactly one of each number per row
@constraint(m, rowC[i=1:8], sum(x[i,j] for j in 1:8) == 1)

# exactly one of each number per column
@constraint(m, colC[j=1:8], sum(x[i,j] for i in 1:8) == 1)

# if the sum of queens in the diagonals is 1 then this cell cannot contain queen
@constraint(m, diagC[i=1:8,j=1:8], sum(x[i_n,j_n] for (i_n,j_n) in get_diag_nbr(i,j,8,8)) - 1
                                    <= 1 - x[i,j] - x[i,j])

# symmetry constraint
@constraint(m, symC[i=1:8,j=1:8], x[i,j] == x[8-i+1,8-j+1]);
```

In [7]:
```
status = solve(m)
println("Status: ", status)
board = getvalue(x)
printChessBoard(board);
```

```
Status: Optimal
+---+---+---+---+---+---+---+---+
|   |   |   |   | 1 |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   | 1 |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
| 1 |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | 1 |   |
+---+---+---+---+---+---+---+---+
|   | 1 |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 1 |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   | 1 |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   | 1 |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

c) What is the smallest number of queens that we can place on the board so that each empty cell is threatened by at least one queen? Show a possible optimal placement.

In [8]:
```
m = Model(solver = CbcSolver())

@variable(m, x[1:8,1:8], Bin)
# exactly one of each number per row
@constraint(m, rowC[i=1:8], sum(x[i,j] for j in 1:8) <= 1)

# exactly one of each number per column
@constraint(m, colC[j=1:8], sum(x[i,j] for i in 1:8) <= 1)

# diagonal constraint
@constraint(m, diagC[i=1:8,j=1:8], sum(x[i_n,j_n] for (i_n,j_n) in get_diag_nbr(i,j,8,8)) - 1
                                   <= 1 - x[i,j] - x[i,j])
# attacking each cell at least once
@constraint(m, attackC[i=1:8,j=1:8], x[i,j] + sum(x[i_n,j_n] for (i_n,j_n) in get_all_nbr(i,j,8,8)) >= 1)
@objective(m, Min, sum(x))
;
```

In [9]:
```
status = solve(m)
println("Status: ", status)
println("Minimum number of queens required: ", getobjectivevalue(m))
board = getvalue(x)
printChessBoard(board);
```

```
Status: Optimal
Minimum number of queens required: 5.0
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 1 |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   | 1 |   |   |   |
+---+---+---+---+---+---+---+---+
| 1 |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   | 1 |   |   |
+---+---+---+---+---+---+---+---+
|   | 1 |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

d) Repeat part (c) but this time find a placement of the queens that also has point symmetry. Does the minimum number of queens required change? Show a possible optimal placement.

In [10]:
```
m = Model(solver = CbcSolver())

@variable(m, x[1:8,1:8], Bin)
# atmost one of each number per row
@constraint(m, rowC[i=1:8], sum(x[i,j] for j in 1:8) <= 1)

# atmost one of each number per column
@constraint(m, colC[j=1:8], sum(x[i,j] for i in 1:8) <= 1)
# diagonal constraint
@constraint(m, diagC[i=1:8,j=1:8], sum(x[i_n,j_n] for (i_n,j_n) in get_diag_nbr(i,j,8,8)) - 1
                                    <= 1 - x[i,j] - x[i,j])
# attacking constraint
@constraint(m, attackC[i=1:8,j=1:8], x[i,j] + sum(x[i_n,j_n]
                                            for (i_n,j_n) in get_all_nbr(i,j,8,8)) >= 1)
# symmetry constraint
@constraint(m, symC[i=1:8,j=1:8], x[i,j] == x[8-i+1,8-j+1])
@objective(m, Min, sum(x))
;
```

In [11]:
```
status = solve(m)
println("Status: ", status)
println("Minimum number of queens required: ", getobjectivevalue(m))
board = getvalue(x)
printChessBoard(board);
```

```
Status: Optimal
Minimum number of queens required: 6.0
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | 1 |   |
+---+---+---+---+---+---+---+---+
|   |   | 1 |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 1 |
+---+---+---+---+---+---+---+---+
| 1 |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   | 1 |   |   |
+---+---+---+---+---+---+---+---+
|   | 1 |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```