Voting

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

```
INFO: Recompiling stale cache file /home/juser/.julia/lib/v0.5/DiffBase.ji for module DiffBase.
INFO: Recompiling stale cache file /home/juser/.julia/lib/v0.5/JuMP.ji for module JuMP.
INFO: Recompiling stale cache file /home/juser/.julia/lib/v0.5/Cbc.ji for module Cbc.
```

In [2]:
```
m = Model(solver=CbcSolver())
```

Out[2]:

$$\min \quad 0$$

Subject to

In [3]:
```
@variable(m, X[1:5,1:10], Bin)
```

Out[3]: $X_{i,j} \in \{0,1\} \quad \forall i \in \{1,2,3,4,5\}, j \in \{1,2,\dots,9,10\}$

In [5]:
```
for j = 1:10
    @constraint(m, sum(X[i,j] for i = 1:5) == 1)
end
```

In [6]:
```
Rep = [80, 60, 40, 20, 40, 40, 70, 50, 70, 70]
Dem = [34, 44, 44, 24, 114, 64, 14, 44, 54, 64]
DV_Upper_Bound = sum(Dem[i] for i = 1:10)
RV_Upper_Bound = sum(Rep[i] for i = 1:10)
Lower_Bound = 150
Upper_Bound = 250

#Voters[1,j] = number of Republicans in district j
#Voters[2,j] = number of Democrats in district j
@variable(m, Voters[1:2,1:5] >= 0, Int)

#Majority[i] = 1 if district i is a strict majority for the Democrats. 0 otherwise.
@variable(m, Majority[1:5], Bin)

for i = 1:5
    @constraint(m, Voters[1,i] == sum(X[i,j]*Rep[j] for j = 1:10))
    @constraint(m, Voters[2,i] == sum(X[i,j]*Dem[j] for j = 1:10))
    @constraint(m, Lower_Bound <= Voters[1,i] + Voters[2,i] <= Upper_Bound)
    @constraint(m, Voters[2,i] - Voters[1,i] <= DV_Upper_Bound*Majority[i])
    @constraint(m, (1 - Voters[2,i] + Voters[1,i]) <= (RV_Upper_Bound + 1)*(1 - Majority[i]))
end
```

In [7]:
```
@objective(m, Max, sum(Majority[i] for i = 1:5))
```

Out[7]: $Majority_1 + Majority_2 + Majority_3 + Majority_4 + Majority_5$

In [8]:
```
solve(m)
```

Out[8]: :Optimal

In [9]:
```
getvalue(X)
```

Out[9]:
```
5×10 Array{Float64,2}:
 0.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  1.0  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  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  1.0  1.0  0.0  0.0  0.0  1.0  0.0  0.0
```

In [10]:
```
getvalue(Majority)
```

Out[10]:
```
5-element Array{Float64,1}:
 1.0
 1.0
 0.0
 0.0
 1.0
```

In [11]:
```
getvalue(Voters)
```

Out[11]:
```
2×5 Array{Float64,2}:
 100.0   40.0  150.0  140.0  110.0
 108.0  114.0   88.0   78.0  112.0
```

Paint Production

In [135]:
```
m = Model(solver = CbcSolver());
```

In [136]:
```
@variable(m, X[1:5,1:5], Bin)
@variable(m, W[1:5,1:25], Bin)
@variable(m, CT[1:5,1:25] >= 0, Int);
```

In [137]:
```
for i = 1:5
    @constraint(m, sum(X[i,j] for j = 1:5) == 1)
    @constraint(m, sum(X[j,i] for j = 1:5) == 1)
end
```

In [138]:
```
A = [ 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 ];
```

In [139]:
```
z = 1
for i = 1:5
    j = (i < 5) ? i + 1 : 1
    for l = 1:5
        for k = 1:5
            @constraint(m, (X[l,i] + X[k,j] - 2) >= (-2)*(1 - W[i, (l - 1)*5 + k]))
            @constraint(m, (X[l,i] + X[k,j] - 1) <= W[i, (l - 1)*5 + k])
        end
    end
end
```

In [142]:
```
for i = 1:5
    j = (i < 5) ? i + 1 : 1
    for z = 1:25
        l = ceil(z / 5)
        l = convert(Int64,l)
        k = z - (l - 1)*5
        @constraint(m, CT[i,z] == A[l,k]*W[i,z])
    end
end
```

In [143]:
```
@objective(m, Min, sum(CT[i,j] for i = 1:5, j = 1:25));
```

In [144]:
```
solve(m)
```

Out[144]:
```
:Optimal
```

In [151]:
```
println("The following array holds the optimal ordering of paint jobs.")
println("If Array[i,j] = 1, then paint job j should be placed at the ith postion.")
getvalue(X)
```

```
The following array holds the optimal ordering of paint jobs.
If Array[i,j] = 1, then paint job j should be placed at the ith postion.
```

Out[151]:
```
5×5 Array{Float64,2}:
 0.0  1.0  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  1.0  0.0  0.0
 0.0  0.0  0.0  0.0  1.0
```

In [152]:
```
println("Total run time: ", getobjectivevalue(m) + 40 + 35 + 45 + 32 + 50)
```

```
Total run time: 243.0
```

The Queens problem

Part a

In [61]:
```
m = Model(solver = CbcSolver())
@variable(m, board[1:8,1:8], Bin);
```

In [62]:
```
@constraint(m, sum(board[i,j] for i = 1:8, j = 1:8) >= 8);
```

In [63]:
```
#rows/columns are attack-free
for i = 1:8
    @constraint(m, sum(board[i,j] for j = 1:8) <= 1)
    @constraint(m, sum(board[j,i] for j = 1:8) <= 1)
end
```

In [64]:
```
#diagonals are attack-free
for i = 1:8
    @constraint(m, sum(board[j, 8 - j + i] for j = i:8) <= 1)
    @constraint(m, sum(board[i - j + 1, j] for j = 1:i) <= 1)
    @constraint(m, sum(board[j, j - i + 1] for j = i:8) <= 1)
    @constraint(m, sum(board[j - i + 1, j] for j = i:8) <= 1)
end
```

In [65]:
```
@objective(m, Min, sum(board[i,j] for i = 1:8, j = 1:8));
```

In [66]:
```
solve(m)
```

Out[66]:
```
:Optimal
```

In [67]:
```
getvalue(board)
```

Out[67]:
```
8×8 Array{Float64,2}:
 0.0  1.0  0.0  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  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  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  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  1.0  0.0  0.0  0.0
```

Part b

In [68]:
```
m = Model(solver = CbcSolver())
@variable(m, board[1:8,1:8], Bin);
@constraint(m, sum(board[i,j] for i = 1:8, j = 1:8) >= 8);
```

In [69]:
```
#rows/columns are attack-free
for i = 1:8
    @constraint(m, sum(board[i,j] for j = 1:8) <= 1)
    @constraint(m, sum(board[j,i] for j = 1:8) <= 1)
end
#diagonals are attack-free
for i = 1:8
    @constraint(m, sum(board[j, 8 - j + i] for j = i:8) <= 1)
    @constraint(m, sum(board[i - j + 1, j] for j = 1:i) <= 1)
    @constraint(m, sum(board[j, j - i + 1] for j = i:8) <= 1)
    @constraint(m, sum(board[j - i + 1, j] for j = i:8) <= 1)
end
```

In [71]:
```
#point-symmetry
for i = 1:8, j = 1:8
    @constraint(m, board[i,j] == board[8 - i + 1, 8 - j + 1])
end
```

In [72]:
```
@objective(m, Min, sum(board[i,j] for i = 1:8, j = 1:8));
```

In [73]:
```
solve(m)
```

Out[73]:
```
:Optimal
```

In [74]:
```
getvalue(board)
```

Out[74]:
```
8×8 Array{Float64,2}:
 0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0
 0.0  0.0  1.0  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  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  0.0  0.0  0.0  0.0  0.0  0.0  1.0
 0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
 0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0
```

Part c

In [136]:
```
m = Model(solver = CbcSolver())
@variable(m, board[1:8,1:8], Bin)
@variable(m, act_board[1:8,1:8], Bin)
@variable(m, attack[1:8,1:8,1:4] >= 0, Int);
```

In [137]:
```
for i = 1:8,j = 1:8
    @constraint(m, attack[i,j,1] == sum(board[i, k] for k = 1:8) - board[i,j])
    @constraint(m, attack[i,j,2] == sum(board[k, j] for k = 1:8) - board[i,j])
    @constraint(m, attack[i,j,3] == sum(board[i + k,j + k] for k = 1:(8 - ((i > j) ? i : j)))
                                  + sum(board[i-k,j-k] for k = 1:(((i < j) ? i : j) - 1)))
x = 8 - j
y = i - 1
a = 8 - i
b = j - 1

    @constraint(m, attack[i,j,4] == sum(board[i-k,j+k] for k = 1:((x < y) ? x : y))
                                  + sum(board[i+k,j-k] for k = 1:((a < b) ? a : b)))
end
```

In [138]:
```
for i = 1:8,j = 1:8
    @constraint(m, 2 - sum(attack[i,j,k] for k = 1:4) + board[i,j] <= 2*act_board[i,j])
    @constraint(m, sum(attack[i,j,k] for k = 1:4) + board[i,j] >= 2)
end
```

In [139]:
```
@objective(m, Min, sum(act_board[i,j] for i = 1:8, j = 1:8));
```

In [140]:
```
solve(m)
```

Out[140]:
```
:Optimal
```

In [142]:
```
getvalue(board)
```

Out[142]:
```
8×8 Array{Float64,2}:
 1.0  1.0  1.0  0.0  0.0  0.0  1.0  1.0
 1.0  1.0  0.0  0.0  0.0  0.0  1.0  1.0
 0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0
 0.0  0.0  1.0  1.0  1.0  1.0  0.0  0.0
 0.0  0.0  1.0  1.0  1.0  1.0  0.0  0.0
 0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0
 1.0  1.0  0.0  0.0  0.0  0.0  1.0  1.0
 1.0  1.0  0.0  0.0  0.0  0.0  1.0  1.0
```

Part d

In [153]:
```
m = Model(solver = CbcSolver())
@variable(m, board[1:8,1:8], Bin)
@variable(m, act_board[1:8,1:8], Bin)
@variable(m, attack[1:8,1:8,1:4] >= 0, Int);
for i = 1:8,j = 1:8
    @constraint(m, attack[i,j,1] == sum(board[i, k] for k = 1:8) - board[i,j])
    @constraint(m, attack[i,j,2] == sum(board[k, j] for k = 1:8) - board[i,j])
    @constraint(m, attack[i,j,3] == sum(board[i + k,j + k] for k = 1:(8 - ((i > j) ? i : j)))
                                  + sum(board[i-k,j-k] for k = 1:(((i < j) ? i : j) - 1)))
x = 8 - j
y = i - 1
a = 8 - i
b = j - 1

    @constraint(m, attack[i,j,4] == sum(board[i-k,j+k] for k = 1:((x < y) ? x : y))
                                  + sum(board[i+k,j-k] for k = 1:((a < b) ? a : b)))
end
```

In [154]:
```
for i = 1:8,j = 1:8
    @constraint(m, 2 - sum(attack[i,j,k] for k = 1:4) + board[i,j] <= 2*act_board[i,j])
    @constraint(m, sum(attack[i,j,k] for k = 1:4) + board[i,j] >= 2)
end
```

In [155]:
```
#point-symmetry
for i = 1:8, j = 1:8
    @constraint(m, board[i,j] == board[8 - i + 1, 8 - j + 1])
end
```

In [156]:
```
@objective(m, Min, sum(act_board[i,j] for i = 1:8, j = 1:8));
```

In [157]:
```
solve(m)
```

Out[157]:
```
:Optimal
```

In [158]:
```
getvalue(board)
```

Out[158]:
```
8×8 Array{Float64,2}:
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
 0.0  1.0  0.0  0.0  0.0  0.0  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  1.0  1.0  0.0  0.0  0.0
 0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0
 0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0
 0.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
```

In [ ]: