HW9

April 21, 2018

1 Homework 9: More integer programs

Name: Zihao Qiu
Email: zqiu34@wisc.edu
Campus ID: 9079810942

1.1 1. Voting

In [23]:
```
using JuMP, Cbc

info = [[80 34];
        [60 44];
        [40 44];
        [20 24];
        [40 114];
        [40 64];
        [70 14];
        [50 44];
        [70 54];
        [70 64]]

m = Model(solver=CbcSolver())

# x[i][j]=1 means city i belongs to district j
@variable(m, x[1:10, 1:5], Bin)

# z[i]=1 means district i has a Democratic majority
@variable(m, z[1:5], Bin)

# sum of each line equals to 1
for i in 1:10
    @constraint(m, sum(x[i,:]) == 1)
end

# each district contains 150k~250k voters
for j in 1:5
    @constraint(m, 150 <= sum{(info[i,1]+info[i,2])*x[i,j],i=1:10} <= 250)
```

```
    end

    # connection between x and z
    # if district i has a Democratic majority, then z[i]=1
    # if z[i]=1, D[i]-R[i]>=0    ==>    D[i]-R[i] >= m(1-z[i])
    for j in 1:5
        @constraint(m, sum{(info[i,2]-info[i,1])*x[i,j],i=1:10} >= -250*(1-z[j]))
    end

    @objective(m, Max, sum(z))

    status = solve(m)

    println(status)

    for j in 1:5
        println("District: ", j)
        if (getvalue(z[j]) == 1)
            println("Demo majority")
        else
            println("Repu majority")
        end

        println("It contains:")

        for i in 1:10
            if (getvalue(x[i,j]) >= 0.99)
                println("city: ", i)
            end
        end
        println("")
    end

    println("objective = ", getobjectivevalue(m))
```

```
Optimal
District: 1
Repu majority
It contains:
city: 1
city: 2

District: 2
Repu majority
It contains:
city: 7
city: 10
```

```
District: 3
Demo majority
It contains:
city: 3
city: 4
city: 8

District: 4
Demo majority
It contains:
city: 6
city: 9

District: 5
Demo majority
It contains:
city: 5

objective = 3.0
```

1.2 2. The Queens problem

```
In [70]: # HELPER function, print the chess board
         function print_chessboard(mat_x)
             println("+---+---+---+---+---+---+---+---+")
             for i in 1:8
                 for j in 1:8
                     print("|")
                     if (mat_x[i,j]==1.0)
                         print(" * ")
                     else
                         print("   ")
                     end
                 end
                 println("|")
                 println("+---+---+---+---+---+---+---+---+")
             end
         end
         ;
```

1.2.1 (a)

```
In [71]: using JuMP, Cbc

         m = Model(solver=CbcSolver())

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

```
# the sum each row and each column should be 1
for i in 1:8
    @constraint(m, sum{x[i,j], j=1:8} == 1)
    @constraint(m, sum{x[j,i], j=1:8} == 1)
end

# diagonal-direction 1
for diag_sum in 2:16
    if (diag_sum <= 9)
        @constraint(m, sum{x[i,diag_sum-i], i=1:(diag_sum-1)} <= 1)
    else
        @constraint(m, sum{x[i,diag_sum-i], i=(diag_sum-8):8} <= 1)
    end
end

# diagonal-direction 2
for diag_diff in -7:7
    if (diag_diff >= 0)
        @constraint(m, sum{x[i,diag_diff+i], i=1:(8-abs(diag_diff))} <= 1)
    else
        @constraint(m, sum{x[i,diag_diff+i], i=(-diag_diff+1):8} <= 1)
    end
end

@objective(m, Min, sum(x))

status = solve(m)

mat_x = getvalue(x)

println(status)
print_chessboard(mat_x)
```

```
Optimal
+---+---+---+---+---+---+---+---+
|   |   |   |   | * |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | * |
+---+---+---+---+---+---+---+---+
| * |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   | * |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | * |   |
+---+---+---+---+---+---+---+---+
|   |   | * |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

```
|   |   |   |   |   |   | * |   |   |
+---+---+---+---+---+---+---+---+
|   |   | * |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

1.2.2 (b)

In [75]:
```
using JuMP, Cbc

m = Model(solver=CbcSolver())

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

# the sum each row and each column should be 1
for i in 1:8
    @constraint(m, sum{x[i,j], j=1:8} == 1)
    @constraint(m, sum{x[j,i], j=1:8} == 1)
end

# diagonal-direction 1
for diag_sum in 2:16
    if (diag_sum <= 9)
        @constraint(m, sum{x[i,diag_sum-i], i=1:(diag_sum-1)} <= 1)
    else
        @constraint(m, sum{x[i,diag_sum-i], i=(diag_sum-8):8} <= 1)
    end
end

# diagonal-direction 2
for diag_diff in -7:7
    if (diag_diff >= 0)
        @constraint(m, sum{x[i,diag_diff+i], i=1:(8-abs(diag_diff))} <= 1)
    else
        @constraint(m, sum{x[i,diag_diff+i], i=(-diag_diff+1):8} <= 1)
    end
end

# has point symmetry
for i in 1:8
    for j in 1:8
        @constraint(m, x[i,j] == x[9-i,9-j])
    end
end

@objective(m, Min, sum(x))

status = solve(m)
```

```
println(status)
print_chessboard(getvalue(x))
```

```
Optimal
+---+---+---+---+---+---+---+---+
|   |   |   |   | * |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   | * |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
| * |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | * |   |
+---+---+---+---+---+---+---+---+
|   | * |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | * |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   | * |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   | * |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

1.2.3 (c)

```
In [78]: using JuMP, Cbc

         m = Model(solver=CbcSolver())

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

         # for each cell on chess board
         # there exist a queen on its row or column or diagnoal line
         for i in 1:8
             for j in 1:8
                 @constraint(m, sum{x[i,k], k=1:8}+sum{x[k,j],k=1:8}
                     +sum{x[k,(i+j)-k],k=max(1,(i+j)-8):min(8,(i+j)-1)}
                     +sum{x[k,(j-i)+k],k=max(1,-(j-i)+1):min(8,8-(j-i))} >= 1)
             end
         end

         @objective(m, Min, sum(x))

         status = solve(m)

         println(status)
         println("The smallest number of queens: ", getobjectivevalue(m))
```

```
print_chessboard(getvalue(x))
```

```
Optimal
The smallest number of queens: 5.0
+---+---+---+---+---+---+---+---+
|   |   |   |   | * |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   | * |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   | * |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | * |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   | * |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

1.2.4 (d)

In [79]:
```
using JuMP, Cbc

m = Model(solver=CbcSolver())

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

# for each cell on chess board
# there exist a queen on its row or column or diagnoal line
for i in 1:8
    for j in 1:8
        @constraint(m, sum{x[i,k], k=1:8}+sum{x[k,j],k=1:8}
            +sum{x[k,(i+j)-k],k=max(1,(i+j)-8):min(8,(i+j)-1)}
            +sum{x[k,(j-i)+k],k=max(1,-(j-i)+1):min(8,8-(j-i))} >= 1)
    end
end

# has point symmetry
for i in 1:8
    for j in 1:8
        @constraint(m, x[i,j] == x[9-i,9-j])
    end
end
```

```
        @objective(m, Min, sum(x))

        status = solve(m)

        println(status)
        println("The smallest number of queens: ", getobjectivevalue(m))
        print_chessboard(getvalue(x))
```

```
Optimal
The smallest number of queens: 6.0
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   | * |   |
+---+---+---+---+---+---+---+---+
| * |   |   |   |   | * |   |   |
+---+---+---+---+---+---+---+---+
|   |   | * |   |   |   |   | * |
+---+---+---+---+---+---+---+---+
|   | * |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

1.3 3. Paint production

The sum of all blending times is a constant. So we only need to find out the minimum cleaning time. It can be modeled as a TSP.

Sum of blending time = 40+35+45+32+50 = 202 min

```
In [80]: 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 [90]: N = size(A,1);

In [91]: using JuMP, Cbc

         m = Model(solver=CbcSolver())

         # x[i,j]=1 means i->j
         @variable(m, x[1:5,1:5], Bin)
```

```
# one in-edge
for j = 1:5
    @constraint(m, sum{x[k, j], k=1:5} == 1)
end

# one out-edge
for i = 1:5
    @constraint(m, sum{x[i, k], k=1:5} == 1)
end

# no self-loop
for i = 1:5
    @constraint(m, x[i,i] == 0)
end

@objective(m, Min, sum{x[i,j]*A[i,j], i in 1:5, j in 1:5})

# MTZ variables and constraint
@variable(m, u[1:5])

for i in 1:5
    for j in 2:5
        @constraint(m, u[i]-u[j]+N*x[i,j] <= N-1)
    end
end

status = solve(m)

println(status)
println(getvalue(x))
println(getobjectivevalue(m))
```

```
Optimal
[0.0 0.0 0.0 1.0 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 1.0 0.0 0.0
 0.0 1.0 0.0 0.0 0.0]
41.0
```

So the minimum cleaning time is 41 min. And one possible order is: 2->1->4->3->5.
The minimum total time is 243 min.