# HW3

February 16, 2018

### 0.1 Homework 3: Minimax and network flows

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

### 0.2 Problem 1 (Doodle scheduling)

I use a matrix x(15*13) as the variable. Each row vector of this matrix represents the possible meeting time of one person. For example, [0, 1 , 0, ...] means that this person will have meeting at 10:20.
Based on the information in this question, the matrix should have the follow 3 constraints: first, the elements in this matrix should be non-negative; second, the sum of each column should be 1; third the sum of 7th column is 3 and sums of each other columns is 1.
The object of this LP problem is to maximize the element-wise production between matrix 'availability' and my variable matrix. So if there exists possible solution, the max value of this production will be 15.

```
In [1]: using JuMP, NamedArrays

        availability =
          [ 0 0 1 1 0 0 0 1 1 0 0 0 0
            0 1 1 0 0 0 0 0 1 1 0 0 0
            0 0 0 1 1 0 1 1 0 1 1 1 1
            0 0 0 1 1 1 1 1 1 1 1 1 0
            0 0 0 0 0 0 1 1 1 0 0 0 0
            0 1 1 0 0 0 0 0 1 1 0 0 0
            0 0 0 1 1 1 1 0 0 0 0 0 0
            1 1 0 0 0 0 0 0 0 0 1 1 1
            1 1 1 0 0 0 0 0 0 1 1 0 0
            0 0 0 0 0 0 0 1 1 0 0 0 0
            0 0 0 0 0 0 1 1 1 0 0 0 0
            1 1 0 0 0 1 1 1 1 0 0 1 1
            1 1 1 0 1 1 0 0 0 0 0 1 1
            0 1 1 1 0 0 0 0 0 0 0 0 0
            1 1 0 0 1 1 0 0 0 0 0 0 0 ]

        TIMES = ["10:00","10:20","10:40","11:00","11:20","11:40","lunch","1:00",
                 "1:20","1:40","2:00","2:20","2:40"]
        NAMES = [:Manuel,:Luca,:Jule,:Michael,:Malte,:Chris,:Spyros,:Mirjam,:Matt,
                 :Florian,:Josep,:Joel,:Tom,:Daniel,:Anne ]
        times = NamedArray( availability, (NAMES,TIMES), ("NAME","TIME"));
```

```
In [19]: using JuMP, Clp

         m = Model(solver = ClpSolver())

         @variable(m, 0 <= x[1:15,1:13] <= 1)

         for i in 1:15
             @constraint(m, sum(x[i,:]) == 1)
         end

         for i in 1:6
             @constraint(m, sum(x[:,i]) == 1)
         end
         @constraint(m, sum(x[:,7]) == 3)
         for i in 8:13
             @constraint(m, sum(x[:,i]) == 1)
         end

         @objective(m, Max, sum(x[i,j]*times[i,j] for i in 1:15, j in 1:13))

         status = solve(m)
         println(status)

         for i in 1:15
             println(getvalue(x[i,:]))
         end
         println("objective value: ", getobjectivevalue(m))
```

```
Optimal
[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, 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, 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, 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, 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, 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]
[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, 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, 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, 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, 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, 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, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
objective value: 15.0
```

I use this following program to help generate the content of markdown table.

```
In [25]: for i in 1:15
             line = '|' * string(NAMES[i]) * '|'
             for j in 1:13
                 line = line * string(getvalue(x[i,j])) * '|'
             end
             println(line)
         end
```

```
|Manuel|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|
|Luca|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|
|Jule|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|
|Michael|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|
|Malte|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|
|Chris|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|
|Spyros|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|
|Mirjam|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|
|Matt|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|
|Florian|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|
|Josep|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|
|Joel|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|
|Tom|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|
|Daniel|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|
|Anne|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|
```

So a feasible interview schedule is as follows (I use * to mark the feasible time slot of each person):

| Name | 10:00 | 10:20 | 10:40 | 11:00 | 11:20 | 11:40 | lunch | 1:00 | 1:20 | 1:40 | 2:00 | 2:20 | 2:40 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Manuel | 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 |
| Luca | 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 |
| Jule | 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 |
| Michael | 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 |
| Malte | 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 |
| Chris | 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 |
| Spyros | 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 |
| Mirjam | *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 |
| Matt | 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 |
| Florian | 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 |
| Josep | 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 |
| Joel | 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 |
| Tom | 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 |

| Name | 10:00 | 10:20 | 10:40 | 11:00 | 11:20 | 11:40 | lunch | 1:00 | 1:20 | 1:40 | 2:00 | 2:20 | 2:40 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Daniel | 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 |
| Anne | 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 |

## 0.3 Problem 2 (Car rental)

I use a matrix x(4*6) as the variable. Rows of this matrix represants agency 2,5,8,9 which have more cars then required. Columns of this matrix represants agency 1,3,4,6,7,10 which have less cars then required. The item x[i,j] represants the number of cars that should be moved from agency i to agency j.
Because agency 2 has 7 extra cars, agency 5 has 3 extra cars, agency 8 has 4 extra cars and agency 9 has 6 extra cars, the sums of row 1,2,3,4 are 7,3,4,6.
Because agency 1 needs 2 cars, agency 3 needs 4 cars, agency 4 needs 3 cars, agency 6 needs 5 cars, agency 7 needs 1 car and agency 10 needs 5 cars, the sums of column 1,3,4,6,7,10 are 2,4,3,5,1,5.
I define a function 'Cost', it can take the indexes of two agencies and the number of cars and return the cost you need.
The object of this problem is to minimize the total cost which is easy to evaluate.
My code is as follows:

```
In [8]: using JuMP, Clp

        location =
            [0 20 18 30 35 33  5  5 11  2
             0 20 10 12  0 25 27 10  0 15]

        function Cost(agen1, agen2, num, loc=location)
            x1 = loc[1, agen1]
            y1 = loc[2, agen1]
            x2 = loc[1, agen2]
            y2 = loc[2, agen2]
            dis = sqrt((x1-x2)^2 + (y1-y2)^2) * 1.3
            cost = 0.5 * dis * num
            return cost
        end

        m = Model(solver = ClpSolver())

        row_idx = [2 5 8 9]
        col_idx = [1 3 4 6 7 10]

        @variable(m, x[1:4,1:6] >= 0)

        @constraint(m, sum(x[1,:]) == 7) # location 2
        @constraint(m, sum(x[2,:]) == 3) # location 5
        @constraint(m, sum(x[3,:]) == 4) # location 8
        @constraint(m, sum(x[4,:]) == 6) # location 9
        @constraint(m, sum(x[:,1]) == 2) # location 1
        @constraint(m, sum(x[:,2]) == 4) # location 3
        @constraint(m, sum(x[:,3]) == 3) # location 4
        @constraint(m, sum(x[:,4]) == 5) # location 6
        @constraint(m, sum(x[:,5]) == 1) # location 7
        @constraint(m, sum(x[:,6]) == 5) # location 10

        @objective(m, Min, sum(Cost(row_idx[i], col_idx[j], x[i,j]) for i in 1:4, j in 1:6))

        status = solve(m)
        println(status)

        for i in 1:4
            println(getvalue(x[i,:]))
        end
        println("objective value: ", getobjectivevalue(m))
```

```
Optimal
[0.0, 1.0, 0.0, 5.0, 1.0, 0.0]
[0.0, 0.0, 3.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 4.0]
[2.0, 3.0, 0.0, 0.0, 0.0, 1.0]
objective value: 152.63901632295628
```

The movement is as follows. The meaning of this table is agency 2 should move 1 car to agency 3, 5 cars to agency 6 and 1 car to agency 7 etc.

| movement | agency 1 | agency 3 | agency 4 | agency 6 | agency 7 |
|---|---|---|---|---|---|
| agnecy 2 | 0 | 1 | 0 | 5 | 1 |
| agnecy 5 | 0 | 0 | 3 | 0 | 0 |
| agnecy 8 | 0 | 0 | 0 | 0 | 0 |
| agnecy 9 | 2 | 3 | 0 | 0 | 0 |

The minimum of total cost is $152.639.

## 0.4 Problem 3 (Building a stadium)

### 0.4.1 (a)

To get the earliest date of completion for the construction, I set variable tstart, which represants the start time of each task.
I add the constraints that if task i is the predecessor of task j, then tstart[j]≥tstart[i] + duration[i].
Then I set tstart[1]=0, and the object is to minimize the end time of the last task.

```
In [5]: using JuMP, Clp

        tasks = 1:18
```

```
    durations = [2 16 9 8 10 6 2 2 9 5 3 2 1 7 4 3 9 1]
    predecessors = ([], [1], [2], [2], [3], [4,5], [4], [6], [4,6],
                    [4], [6], [9], [7], [2], [4,14], [8,11,14], [12], [17])

    m = Model(solver=ClpSolver())

    @variable(m, tstart[tasks])

    for i in tasks
        for j in predecessors[i]
            @constraint(m, tstart[i] >= tstart[j] + durations[j])
        end
    end
    @constraint(m, tstart[1] == 0)
    @objective(m, Min, tstart[18] + durations[18])

    solve(m)
    println(getvalue(tstart))
    println("minimum duration: ", getobjectivevalue(m))
```

```
tstart: 1 dimensions:
[ 1] = -0.0
[ 2] = 2.0
[ 3] = 18.0
[ 4] = 18.0
[ 5] = 27.0
[ 6] = 37.0
[ 7] = 26.0
[ 8] = 43.0
[ 9] = 43.0
[10] = 26.0
[11] = 43.0
[12] = 52.0
[13] = 28.0
[14] = 18.0
[15] = 26.0
[16] = 46.0
[17] = 54.0
[18] = 63.0
minimum duration: 64.0
```

So we can see that it will take at least 64 days to complete the construction.

### 0.4.2 (b)

In this question, I set two groups of variables and they are tstart and reduc.
The meaning of tstart doesn't change. The meaning of reduc is how many weeks we should reduce

in each task, for example, reduc[4]=1 means we should reduce 1 weeks in task 4.
The constraints of this LP problem are the combination of the constraints in (a) and the constraints about reduc[]. I think the constraints are obviuos.
The object is to maximize the profit and it's also not difficult to evaluate.

```
In [8]: using JuMP, Clp

        tasks = 1:18
        durations = [2 16 9 8 10 6 2 2 9 5 3 2 1 7 4 3 9 1]
        predecessors = ([], [1], [2], [2], [3], [4,5], [4], [6], [4,6],
                        [4], [6], [9], [7], [2], [4,14], [8,11,14], [12], [17])

        # additional columns of data (maximum reduction possible )
        max_reduction =  [0,  3,  1,  2,  2,  1, 1, 0,  2,  1,  1, 0, 0,  2,  2, 1,  3, 0]
        cost_reduction = [0, 30, 26, 12, 17, 15, 8, 0, 42, 21, 18, 0, 0, 22, 12, 6, 16, 0]
        bonus_amount = 30     # bonus for expediting the project ($1,000/week )

        m = Model(solver=ClpSolver())

        @variable(m, tstart[tasks])
        @variable(m, reduc[tasks] >= 0)

        for i in tasks
            @constraint(m, reduc[i] <= max_reduction[i])
        end

        for i in tasks
            for j in predecessors[i]
                @constraint(m, tstart[i] >= tstart[j] + durations[j] - reduc[i])
            end
        end

        @constraint(m, tstart[1] == 0)

        @objective(m, Max, (64 - (tstart[18] + durations[18])) * bonus_amount
                    - sum(reduc[i] * cost_reduction[i] for i in tasks))

        solve(m)
        println(getvalue(tstart))
        println(getvalue(reduc))
        println("max profit: ", getobjectivevalue(m))
        println("minimum duration:", getvalue(tstart[18]) + durations[18])

tstart: 1 dimensions:
[ 1] = -0.0
[ 2] = -1.0
[ 3] = 14.0
[ 4] = 15.0
[ 5] = 21.0
[ 6] = 30.0
[ 7] = 23.0
[ 8] = 36.0
[ 9] = 36.0
[10] = 23.0
[11] = 36.0
[12] = 45.0
[13] = 25.0
[14] = 15.0
[15] = 23.0
[16] = 39.0
[17] = 44.0
[18] = 53.0
reduc: 1 dimensions:
[ 1] = 0.0
[ 2] = 3.0
[ 3] = 1.0
[ 4] = 0.0
[ 5] = 2.0
[ 6] = 1.0
[ 7] = 0.0
[ 8] = 0.0
[ 9] = 0.0
[10] = 0.0
[11] = 0.0
[12] = 0.0
[13] = 0.0
[14] = 0.0
[15] = 0.0
[16] = 0.0
[17] = 3.0
[18] = 0.0
max profit: 87.0
minimum duration:54.0
```

We can see that the construction can be finished in 54 days. The task plan and reduction plan can be seen above.
The maximum profit is $87k

### 0.5 Problem 4 (Museum site planning)

Just use the method that solves the Chebyshev problem and replace 'r' by 'r+50'. The five lines we have are:

$$2x + 3y \leq 2100 \tag{1}$$
$$3x - y \leq 1500 \tag{2}$$
$$y \geq 0 \tag{3}$$
$$y \leq 500 \tag{4}$$
$$x \geq 0 \tag{5}$$

So the code is as follows:

```
In [2]: A = [2 3; 3 -1; 0 -1; 0 1; -1 0]
        b = [2100; 1500; 0; 500; 0]

        using JuMP, Clp

        m = Model(solver=ClpSolver())
        @variable(m, r >= 0)           # radius
        @variable(m, x[1:2])           # coordinates of center
        for i = 1:size(A,1)
            @constraint(m, A[i,:]'*x + (r + 50)*norm(A[i,:]) <= b[i])
        end
        @objective(m, Max, r)      # maximize radius

        status = solve(m)
        center = getvalue(x)
        radius = getvalue(r)

        println(status)
        println("The coordinates of the Chebyshev center are: ", center)
        println("The largest possible radius is: ", radius)
```

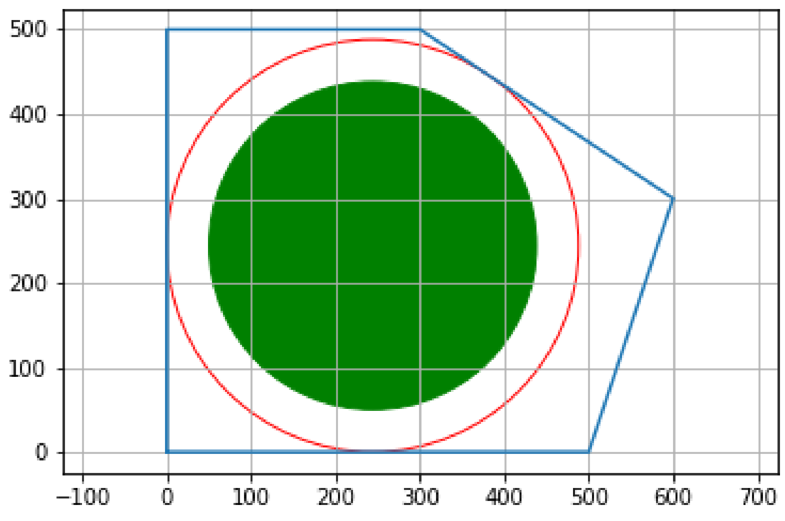
```
Optimal
The coordinates of the Chebyshev center are: [244.029, 244.029]
The largest possible radius is: 194.02852679380186
```

So the optimal radius is 194.02852679380186 feet and the center is at [244.029, 244.029]. I use python3 + matplotlib to draw the picture. The code is as follows:

```
In [ ]: import matplotlib.pyplot as plt
        fig = plt.gcf()
        ax = fig.gca()

        plt.plot([0, 0, 300, 600,500,0], [0, 500, 500, 300,0,0])

        circle1=plt.Circle((244.029,244.029),194.02852679380186, color="green")
        circle2=plt.Circle((244.029,244.029),194.02852679380186 + 50 , color="red", fill=False)
```



Museum Location

```
ax.add_artist(circle1)
ax.add_artist(circle2)

plt.axis('equal')
plt.grid()
plt.show()
```

The red circle is the largest circle. The green area is the location of the best place to build the museum.