Homework 6

August 4, 2020

1 Homework #6

Due August 5 @ 11:59pm

1.1 Submission requirements

Upload a **single PDF file** of your IJulia notebook for this entire assigment. Clearly denote which question each section of your PDF corresponds to.

1.2 Problem 1 – Vehicle Routing

A local bakeshop wants determine the minimum cost plan for picking up its daily supply of milk, butter, and eggs from the four farms that supply the bakeshop. The distance (in miles) between the bakeshop (B) and each farm, and also between each pair of farms, is given in the table below. The table also gives the volume of milk, butter, and eggs (total, in ft^3) to be collected from each farm each day. (The distance between locations is the same in both directions, so for each pair of locations the distance is only reported once.)

	B	F1	F2	F3	F4
B	-	5	12	7	15
F1	-	-	4	10	7
F2	-	-	-	14	20
F3	-	-	-	-	8
	F1	F2	F3	F4	
Supply (ft^3)	7	2	6	3	

The bakeshop has one truck that can carry at most 10 ft^3 of supplies at a time. Because of the size limit, the truck will need to make multiple trips each day to collect the supplies from all the farms. Each trip may pick up supplies from one or more farms, provided the total collected in the trip does not exceed the truck limit.

Formulate an integer program to help the bakeshop assign farms to the trips so that the total number of trips required every day is minimized (Hint: model the problem as a set covering problem. The first step will be to list all possible routes a truck can take.)

[1]:
```
# model
# M = {F1, F2, F3, F4}; S is set of subsets of M covering
# all elements of M. The goal is to minimize the S cardinality
```

```
# first step: list all possible routes a truck can take
# S = {{F1}, {F2}, {F3}, {F4}, {F1, F2}, {F1, F4}, {F2, F3}, {F2, F4}, {F3,␣
 ↪F4}}

using JuMP, Gurobi, NamedArrays

# array of trips
trips = [1,2,3,4,5,6,7,8,9]

# array of farms
farms = [1,2,3,4]


# A is a matrix with rows representing possible trips and columns representing
# the farms covered by that trip
A = [1 0 0 0
    0 1 0 0
    0 0 1 0
    0 0 0 1
    1 1 0 0
    1 0 0 1
    0 1 1 0
    0 1 0 1
    0 0 1 1]
;

m = Model(Gurobi.Optimizer)
set_optimizer_attribute(m, "OutputFlag", 0)

# binary variable for whether include a farm in each trip
@variable(m, x[trips], Bin)

# each farm should be covered by at least one trip
for i in farms
    @constraint(m, sum(A[j,i]x[j] for j in trips ) >= 1)
end

# minimize the total number of trips
@objective(m, Min, sum(x))

optimize!(m)


for i in trips
    if value(x[i]) == 1
        println("trip ", i)
```

```
    end
end
```

```
Academic license - for non-commercial use only
Academic license - for non-commercial use only
trip 6
trip 7
```

Ans: If not considering the distance, then the minimum amount of trips is 2, with {F1, F4} and {F2, F3}.

1.3 Problem 2 – The Magical Baked Goods Machine

Suppose you are in charge of a magical baked goods machine that creates delicious baked goods of many varieties. Every day, the machine creates batches of 5 different baked goods. To produce a batch of a baked good, you must first clean the machine to remove the remnants of the previous batch of bakery treats (e.g., the workflow could be, "clean, make bread, clean, make donuts,…"). The durations of baking each of the 5 items (donuts, bread, cookies, scones, and coffee cake) are 40, 32, 50, 28, and 47 minutes respectively. The cleaning times depend on the item that was previously made in the machine. For example, a long cleaning period is required if bread is made after scones, since the scones leave a sticky residue from the dried fruit they contain that can ruin the bread. The times are given in minutes in the following table. Each pair (i, j) is the cleaning time required if batch j is baked after batch i.

Baked good	Donut	Scone	Cookie	Bread	Coffee Cake
Donut	0	10	6	15	9
Scone	4	0	6	17	8
Cookie	10	11	0	20	14
Bread	7	15	6	0	2
Coffee Cake	5	8	7	7	0

We'd obviously like to spend as little time as possible baking and cleaning. What order should we produce the 5 bakery items in? How long do we spend baking and cleaning each day? The order is the same every day, 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.

Solve this problem as a TSP. You may use either MTZ reformulation or subtour elimination (or both, for fun!).

1.4 Get all subtours helper function

[4]:
```
# HELPER FUNCTION: returns the cycle containing the city START.
function getSubtour(x,start)
    subtour = [start]
    while true
        j = subtour[end]
        for k in foods
```

```
            if x[k,j] == 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 = foods
    subtours = []
    while length(nodesRemaining) > 0
        subtour = getSubtour(x,nodesRemaining[1])
        push!(subtours, subtour)
        nodesRemaining = setdiff(nodesRemaining,subtour)
    end
    return subtours
end
```

[4]:
```
getAllSubtours (generic function with 1 method)
```

1.5 Solve TSP using adaptive subtour elimination

[2]:
```
# Define the problem data (baked goods and costs)
using JuMP, NamedArrays, Gurobi

foods = [:donut, :scone, :cookie, :bread, :coffeecake]

costs  =    [ 0  10  6  15   9
              4   0  6  17   8
             10  11  0  20  14
              7  15  6   0   2
              5   8  7   7   0]

c = NamedArray(costs,(foods,foods))
N = size(costs,1);
```

[7]:
```
m = Model(Gurobi.Optimizer)
set_optimizer_attribute(m, "OutputFlag", 0)

@variable(m, x[foods,foods], Bin)
```

```
@constraint(m, c1[j in foods], sum( x[i,j] for i in foods ) == 1)    # exacly␣
 ↪one edge out of each node
@constraint(m, c2[i in foods], sum( x[i,j] for j in foods ) == 1)    # exactly␣
 ↪one edge into each node
@constraint(m, c3[i in foods], x[i,i] == 0 )                          # no␣
 ↪self-loops
@objective(m, Min, sum( x[i,j]*c[i,j] for i in foods, j in foods␣
 ↪)+40+32+50+28+47)  # minimize total cost
optimize!(m)

sols = []
# we'll run the heuristic 30 times and hope we get an optimal solution
for iters = 1:30
    optimize!(m)
    # total  length of current tour
    println("Tour length: ", objective_value(m))
    xx = value.(x) # save solution
    push!(sols,xx) # save solution
    subtours = getAllSubtours(xx)  # get all the subtours
    display(subtours)
    sleep(1)
    # get length of the subtour list
    len = length(subtours)
    if len == 1
        # solution is just a single tour!
        println("SOLVED!")
        break
    else
        for subtour in subtours
            L = length(subtour)
            # add constraints that cut off each subtour in the list (add two␣
 ↪for each subtour)
            @constraint(m, sum( x[subtour[k+1],subtour[k]] for k = 1:L-1 ) <=␣
 ↪L-2)
            @constraint(m, sum( x[subtour[k],subtour[k+1]] for k = 1:L-1 ) <=␣
 ↪L-2)
        end
    end
end
```

```
2-element Array{Any,1}:
 Symbol[:donut, :scone, :cookie, :donut]
 Symbol[:bread, :coffeecake, :bread]


Academic license - for non-commercial use only
Academic license - for non-commercial use only
```

```
Tour length: 227.0

1-element Array{Any,1}:
 Symbol[:donut, :scone, :cookie, :bread, :coffeecake, :donut]


Tour length: 234.0
SOLVED!
```

1.6 Miller-Tucker-Zemlin formulation

[8]:
```
m = Model(Gurobi.Optimizer)
set_optimizer_attribute(m, "OutputFlag", 0)

@variable(m, x[foods,foods], Bin)                                    # must␣
 ↪formulate as IP this time
@constraint(m, c1[j in foods], sum( x[i,j] for i in foods ) == 1)    # one␣
 ↪out-edge
@constraint(m, c2[i in foods], sum( x[i,j] for j in foods ) == 1)    # one␣
 ↪in-edge
@constraint(m, c3[i in foods], x[i,i] == 0 )                          # no␣
 ↪self-loops
@objective(m, Min, sum( x[i,j]*c[i,j] for i in foods, j in foods␣
 ↪)+40+32+50+28+47)   # minimize total cost

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

optimize!(m)
xx = value.(x)
subtours = getAllSubtours(xx)
display(subtours)
println("Tour length: ", objective_value(m))
```

```
1-element Array{Any,1}:
 Symbol[:donut, :scone, :cookie, :bread, :coffeecake, :donut]


Academic license - for non-commercial use only
Academic license - for non-commercial use only
Tour length: 234.0
```

Ans: (1)The order should be donut, scone, cookie, bread, coffeecake, donut, … . (2)We need 234 min to finish a batch. (cleaning: 37 min & baking: 197 min).