Homework 2 Solutions

July 2, 2021

1 Homework #2

Due June 30 @ 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 – Plant Production Planning

Prof. Smith is considering a side gig selling propogated cuttings of popular houseplants. The propogation process involves putting the plant cuttings (or seeds) in a medium (often water) and waiting for roots to grow. The rooted cuttings can then be planted and/or sold. The propogation process is, unfortunately, highly variable. Prof. Smith has categorized the results of the process into five "classes" of cuttings: "sickly," "poor," "fair," "good," and "excellent." Two different methods can be used to propogate cuttings. On average, method 1 costs \$0.50 per cutting and method 2 costs \$0.40 per cutting. The qualities of rooted cuttings produced by each method are shown in the following table (in percent of total cuttings):

Quality	Method 1	Method 2
sickly	30	20
poor	30	20
fair	20	25
good	15	20
excellent	5	15

Prof. Smith has a rooting enzyme that she can apply to propogated cuttings in an attempt to improve the outcome. It costs $0.25 per cutting to use the enzyme. The results of the attempted improvements can be seen in the table below (in percent of total cuttings). (For example, rooting enzyme applied to good cuttings would produce excellent cuttings 50% of the time, but would not change the quality the remaining 50% of the time.)

Quality	sickly	poor	fair	good	excellent
sickly	30	25	15	20	10
poor	-	30	30	20	20

Quality	sickly	poor	fair	good	excellent
fair	-	-	40	30	30
good	-	-	-	50	50

Prof. Smith has sufficient space at home to propogate at most 200 cuttings per month. Her anticipated monthly demands are 10 excellent cuttings, 20 good cuttings, 30 fair cuttings, and 30 poor cuttings.

(a) Assuming only one application of rooting enzyme is allowed, model the propogation process and determine Prof. Smith's minimum cost strategy for meeting monthly demand.

1.3 Solution

[14]:
```
using JuMP, Clp, NamedArrays

qual = [:sickly,:poor,:fair,:good,:excellent] # possible quality of cutttings
methods = [:1,:2] # propogation methods

cost_method = Dict(zip(methods,[0.5,0.4])) # cost of each method
demand = Dict(zip(qual,[0,30,30,20,10])) # demand for each cutting quality

enz_cost = 0.25 # cost of rooting enzyme per cutting
capacity = 200 # total capacity for cuttings

# create a Named Array of the quality percents produced by each propogation␣
 ↪method
quality_method_matrix = [30 20; 30 20; 20 25; 15 20; 5 15]
quality_method_NA =␣
 ↪NamedArray(quality_method_matrix,(qual,methods),("quality","method"))

# create a Named Array of the quality percents produced by applying the rooting␣
 ↪enzyme
quality_enzyme_matrix = [30 25 15 20 10; 0 30 30 20 20; 0 0 40 30 30; 0 0 0 50␣
 ↪50]
quality_enzyme_NA = NamedArray(quality_enzyme_matrix,(qual[1:
 ↪4],qual),("quality","quality"))

m = Model(Clp.Optimizer)

@variable(m, prop[methods]>=0) # number of cuttings propogated using each method
@variable(m,prod1[qual]>=0) # number of cuttings of each quality produced␣
 ↪through propogation
@variable(m,enz[qual]>=0) # number of cuttings of each quality we apply rooting␣
 ↪enzyme to
@variable(m,prod2[qual]>=0) # number of cuttings of each quality produced in␣
 ↪total
```

```julia
# minimize total cost (cost of each method plus cost of rooting enzyme)
@objective(m,Min,sum(cost_method[i]*prop[i] for i in methods)␣
 ↪+sum(enz_cost*enz[i] for i in qual))

# total number of cuttings of each quality produced by each propogation method
@constraint(m, num_after_prop[i in qual], prod1[i] ==␣
 ↪sum(prop[j]*quality_method_NA[i,j]/100 for j in methods))
# number of total cuttings produced of each quality. cuttings produced = number␣
 ↪propogated - number we apply enzyme to
# + number of cuttings of each quality produced from rooting enzyme
@constraint(m, num_after_enz[i in qual], prod2[i] == (prod1[i] - enz[i]) +␣
 ↪sum((quality_enzyme_NA[j,i]/100)*enz[j] for j in qual[1:4]))
# cannot apply enzyme to more cuttings than are produced
@constraint(m, lim_enz[i in qual], enz[i] <= prod1[i])
# total number of cuttings (propogated plus number of cuttings we apply enzyme␣
 ↪to) must be less than the total capacity
@constraint(m, cap, sum(prop[i]  for i in methods) + sum(enz[j] for j in qual)␣
 ↪<= capacity)
# total cuttings of each quality must be at least the demand for that quality
@constraint(m, dem[i in qual],prod2[i] >= demand[i])

set_optimizer_attribute(m, "LogLevel",false)
optimize!(m)

# print total cost
println("Cost of this strategy: \$", objective_value(m))
# print number of cuttings to propogate with each method
println("Number to propogate with method 1: ", value(prop[:1]))
println("Number to propogate with method 2: ", value(prop[:2]))

println("number of each quality of cutting to which we apply rooting enzyme: ")
for i in qual
    println(i, ": ", value(enz[i]))
end
println("Total cuttings produced of each quality: ")
for i in qual
    println(i, ": ", value(prod2[i]))
end
```

```
Cost of this strategy: $51.94285714285714
Number to propogate with method 1: 20.571428571428523
Number to propogate with method 2: 89.1428571428572
number of each quality of cutting to which we apply rooting enzyme:
sickly: 23.999999999999996
poor: 0.0
fair: 0.0
```

```
good: 0.0
excellent: 0.0
Total cuttings produced of each quality:
sickly: 7.200000000000004
poor: 30.0
fair: 30.000000000000004
good: 25.71428571428572
excellent: 16.800000000000008
```

(b) The model in part (a) was for a single month of operation. To be profitable, Prof. Smith knows she needs to plan at least 3 months in advance. She anticipates demand for the following three months to be:

Month	poor	fair	good	excellent
1	30	30	20	10
2	40	25	15	15
3	20	10	15	30

It costs Prof. Smith \$0.10 to store cuttings produced in one month for a future month. As in part (a), only one application of rooting enzyme is allowed for each cutting. Rooting enzyme cannot be applied to stored cuttings, regardless of whether or not it was applied in a prior month. Modify your model from part (a) to plan for the next three month's of production at a minimum cost.

1.4 Solution

[2]:
```
using JuMP, Clp, NamedArrays

qual = [:sickly,:poor,:fair,:good,:excellent] # possible quality of cutttings
methods = [:1,:2] # propogation methods

cost_method = Dict(zip(methods,[0.5,0.4])) # cost of each method
demand = Dict(zip(qual,[0,30,30,20,10])) # demand for each cutting quality

enz_cost = 0.25 # cost of rooting enzyme per cutting
capacity = 200 # total capacity for cuttings

# create a Named Array of the quality percents produced by each propogation␣
 ↪method
quality_method_matrix = [30 20; 30 20; 20 25; 15 20; 5 15]
quality_method_NA =␣
 ↪NamedArray(quality_method_matrix,(qual,methods),("quality","method"))

# create a Named Array of the quality percents produced by applying the rooting␣
 ↪enzyme
quality_enzyme_matrix = [30 25 15 20 10; 0 30 30 20 20; 0 0 40 30 30; 0 0 0 50␣
 ↪50]
```

```
quality_enzyme_NA = NamedArray(quality_enzyme_matrix,(qual[1:
 ↪4],qual),("quality","quality"))

months = [:1,:2,:3] # plan for the next 3 months
cost_method = Dict(zip(methods,[0.5,0.4]))

# build demand array for each month
demand = Dict()
demand[:1] = Dict(zip(qual,[0,30,30,20,10]))
demand[:2] = Dict(zip(qual,[0,40,25,15,15]))
demand[:3] = Dict(zip(qual,[0,20,10,15,30]))

inv_cost = 0.1 # cost per cutting of storing in inventory

m = Model(Clp.Optimizer)

@variable(m, prop[methods,months] >= 0) # number of cuttings propogated using␣
 ↪each method
@variable(m,prod1[qual,months] >= 0) # number of cuttings of each quality␣
 ↪produced through propogation
@variable(m,enz[qual,months]>=0) # number of cuttings of each quality we apply␣
 ↪rooting enzyme to
@variable(m,prod2[qual,months]>=0) # number of cuttings of each quality produced␣
 ↪in total
@variable(m,inv[qual,months] >= 0) # number of cuttings of each type we store in␣
 ↪inventory

# minimize total cost (cost of each method plus cost of rooting enzyme plus cost␣
 ↪of inventory)
@objective(m,Min,sum(cost_method[i]*prop[i,j] for i in methods, j in months) +
    sum(enz_cost*enz[i,j] for i in qual, j in months) + sum(inv_cost*inv[i,j]␣
 ↪for i in qual, j in months))

# total number of cuttings of each quality produced by each propogation method␣
 ↪in each month
@constraint(m, num_after_prop[i in qual, k in months],
    prod1[i,k] == sum(prop[j,k]*quality_method_NA[i,j]/100 for j in methods))
# number of total cuttings produced of each quality in month 1.
# cuttings produced = number propogated - number we apply enzyme to
# + number of cuttings of each quality produced from rooting enzyme
@constraint(m, num_after_enz_init[i in qual, k in months],
    prod2[i,k] == (prod1[i,k] - enz[i,k]) + sum((quality_enzyme_NA[j,i]/
 ↪100)*enz[j,k] for j in qual[1:4]) )
# number of total cuttings produced of each quality in months 2 and 3.
# cuttings produced = number propogated - number we apply enzyme to
# + number of cuttings of each quality produced from rooting enzyme + inventory␣
 ↪from previous month
@constraint(m, num_after_enz[i in qual, k in months[2:3]],
    prod2[i,k] == (prod1[i,k] - enz[i,k]) + sum((quality_enzyme_NA[j,i]/
 ↪100)*enz[j,k] for j in qual[1:4]) + inv[i,k-1])
# cannot apply enzyme to more cuttings than are produced each month
@constraint(m, lim_enz[i in qual, k in months],
    enz[i,k] <= prod1[i,k])
# total number of cuttings (propogated plus number of cuttings we apply enzyme␣
 ↪to)
# must be less than the total capacity
@constraint(m, cap[k in months[2:end]],
    sum(prop[i,k]  for i in methods) + sum(enz[j,k] for j in qual)  +␣
 ↪sum(inv[j,k-1] for j in qual) <= capacity)
# initial month's capacity
@constraint(m, cap_init,
    sum(prop[i,1]  for i in methods) + sum(enz[j,1] for j in qual)  <= capacity)
# produce at least that month's demand each month
@constraint(m, dem[i in qual, k in months], prod2[i,k] >= demand[k][i])

set_optimizer_attribute(m, "LogLevel",false)
optimize!(m)

# print total cost
println("Cost of this strategy: \$", objective_value(m))

println("Total cuttings produced of each quality: ")
for k in months
    println("Month ", k)
    for i in qual
        println(i, ": ", value(prod2[i,k]))
    end
end
println("Total cuttings stored each month: ")
for k in months
    println("Month ", k)
    for i in qual
        println(i, ": ", value(inv[i,k]))
    end
end
```

```
Cost of this strategy: $164.57074829931972
Total cuttings produced of each quality:
Month 1
sickly: 7.200000000000004
poor: 30.0
fair: 30.000000000000004
```

```
good: 25.71428571428572
excellent: 16.800000000000008
Month 2
sickly: 9.6
poor: 40.0
fair: 26.133333333333326
good: 15.933333333333321
excellent: 15.0
Month 3
sickly: 12.244897959183687
poor: 20.0
fair: 10.20408163265306
good: 17.346938775510196
excellent: 30.0
Total cuttings stored each month:
Month 1
sickly: 0.0
poor: 0.0
fair: 0.0
good: 0.0
excellent: 0.0
Month 2
sickly: 0.0
poor: 0.0
fair: 0.0
good: 0.0
excellent: 0.0
Month 3
sickly: 0.0
poor: 0.0
fair: 0.0
good: 0.0
excellent: 0.0
```

1.5 Problem 2 – Aging like a fine… coffee?

Coffee Co is considering expanding their roast options to include a special line of aged beans. This is a limited-time-only offering, so Coffee Co only has a total of $n = 8$ bags of beans dedicated to this. The current age of each bag of beans is a_i (in months) and is listed in the table below. Each month, we update the current age of bag i as $T_i = a_i + t$ for $t \in \{0, 1, 2, 3, 4, 5, 6, 7\}$. The flavor of the beans deepens with age. The flavor profile can be described as a function of the age of the beans: $f_i(T_i) = b_i T_i^3 - c_i T_i$ for some known coefficients b_i and c_i. All the relevant data are summarized in the following table:

Bag	a	b	c
1	20	2	10
2	5	3	15

Bag	*a*	*b*	*c*
3	4	5	20
4	20	3	5
5	10	1	25
6	15	4	15
7	9	4	10
8	12	5	20

Formulate and solve a linear program that will allow Coffee Co to select one bag of beans to sell each month for the next 8 months that will maximize the overall flavor profile of beans sold. Bags can only be sold once (once a bag is sold, it cannot be sold again in a future month).

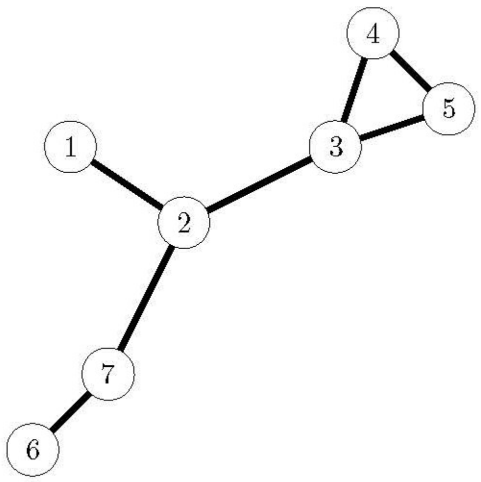
1.6 Solution

[40]:
```
using JuMP, Clp

months = [0 1 2 3 4 5 6 7 8]
bags = [1 2 3 4 5 6 7 8]

age = Dict(zip(bags,[20 5 4 20 10 15 9 12]))

b_vals = Dict(zip(bags,[2 3 5 3 1 4 4 5]))
c_vals = Dict(zip(bags,[10 15 20 5 25 15 10 20]))

flavor_age_matrix = zeros(8,8)
for i in 1:8
    for j in 1:8
        flavor_age_matrix[i,j] = b_vals[i]*(age[i] + months[j])^3 -␣
 ↪c_vals[i]*(age[i] + months[j])
    end
end

m = Model(Clp.Optimizer)

@variable(m, 0 <= x[1:8,1:8] <= 1)

@objective(m, Max, sum(flavor_age_matrix[i,j]*x[i,j] for i in 1:8, j in 1:8))

@constraint(m, month_con[i in 1:8], sum(x[j,i] for j in 1:8) == 1)
@constraint(m, bag_con[i in 1:8], sum(x[i,j] for j in 1:8) == 1)
set_optimizer_attribute(m, "LogLevel",false)

optimize!(m)
sol = [value(x[i,j]) for i in 1:8, j in 1:8]

println(NamedArray(sol,(1:8,1:8),("bags","months")))
```

```
8×8 Named Matrix{Float64}
bags  months    1    2    3    4    5    6    7    8

1             0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0
2             1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3             0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
4             0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0
5             0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
6             0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
7             0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0
8             0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
```

1.7 Problem 3 – We've got the power!

Madison Gas & Electric is planning power generation for the next 24 hours and would like to consult an optimization expert for advice. The figure shown below represents the power distribution network that connects power generating points with power consuming points. Each node represents a neighborhood in the greater Madison area.



Note that these arcs are all undirected, meaning power can flow either direction along each edge in the network. Neighborhoods 1, 4, and 7 and power generating points with the following generator supply and unit costs:

Generator	1	4	7
Capacity (in thousands of KWH)	90	50	85
Unit cost (\$/thousand KWH)	14	13.5	21

Neighborhoods 2, 5, and 6 are power consuming nodes with demands of 30,000, 50,000, and 60,000 KWH, respectively. There is no limit to how much power can be sent along the transmission lines connecting the neighborhoods. It costs \$10 per 1000 KWH to use a transmission line.

Build and solve this problem to minimize the cost of meeting the power demand. Formulate this as a min-cost network flow problem (linear program). Make sure you use directed arcs in your formulation (*hint:* you will have more arcs in your formulation than are shown in the figure).

[19]:
```
using JuMP, Clp

# all nodes, plus create dummy node to "catch" extra supply
nodes = 1:7

# we'll build the arcs list manually. there are other ways to do this.
arcs =␣
 ↪[(1,2),(2,1),(2,3),(3,2),(2,7),(7,2),(3,4),(4,3),(3,5),(5,3),(4,5),(5,4),(7,6)]

arc_cost = 10 # unit cost on each arc is 10

supply_nodes = [1 4 7]
demand_nodes = [2 5 6]
supply_cost = Dict(zip(supply_nodes,[14 13.5 21]))

supply = Dict(zip(supply_nodes,[90 50 85]))# in thousands of KWH. at nodes 1, 4,␣
 ↪7 respectively
demand = Dict(zip(demand_nodes,[30 50 60])) # in thousands of KWH

m = Model(Clp.Optimizer)

@variable(m, x[arcs] >= 0)

@objective(m, Min, sum(supply_cost[i[1]]*x[i] for i in arcs if i[1] in␣
 ↪supply_nodes) +
    arc_cost*sum(x[i] for i in arcs))

for i in nodes
    if i in supply_nodes
        @constraint(m,
                sum(x[(i,j)] for j in nodes if (i,j) in arcs) -
            sum(x[(j,i)] for j in nodes if (j,i) in arcs) <= supply[i])
    elseif i in demand_nodes
        @constraint(m,
                sum(x[(i,j)] for j in nodes if (i,j) in arcs) + demand[i]  <=
        sum(x[(j,i)] for j in nodes if (j,i) in arcs))
    else

        @constraint(m,
                sum(x[(i,j)] for j in nodes if (i,j) in arcs) -
            sum(x[(j,i)] for j in nodes if (j,i) in arcs) == 0)
    end
end

set_optimizer_attribute(m, "LogLevel",false)

optimize!(m)
```

```julia
for i in arcs
    if value(x[i]) > 0.001
        println("Send ", value(x[i]), " KWH on arc ", i)

    end
end
```

```
Send 30.0 KWH on arc (1, 2)
Send 50.0 KWH on arc (4, 5)
Send 60.0 KWH on arc (7, 6)
```

[ ]:

```

```