Week 2 Practice Problem Solution Sketch

June 24, 2021

1 Practice Problems based on Week 2 content – Solution Sketch

1.1 Multiperiod Planning Problems Concepts

- In the ShoeCo example, you are asked to evaluate the solution and identify any problem(s) with it. Do you believe there's anything wrong with the solution? If so, what? How might you suggest fixing that issue (don't worry about following linear programming rules to answer this)?

1.1.1 Answer

There were fractional employees! Obviously we can't hire "half a person." How could we solve it? We could make all employee variables integer. Or, we could decide that we want to hire part-time employees to make up the fractional portions of our solution. Or you might have a more creative idea!

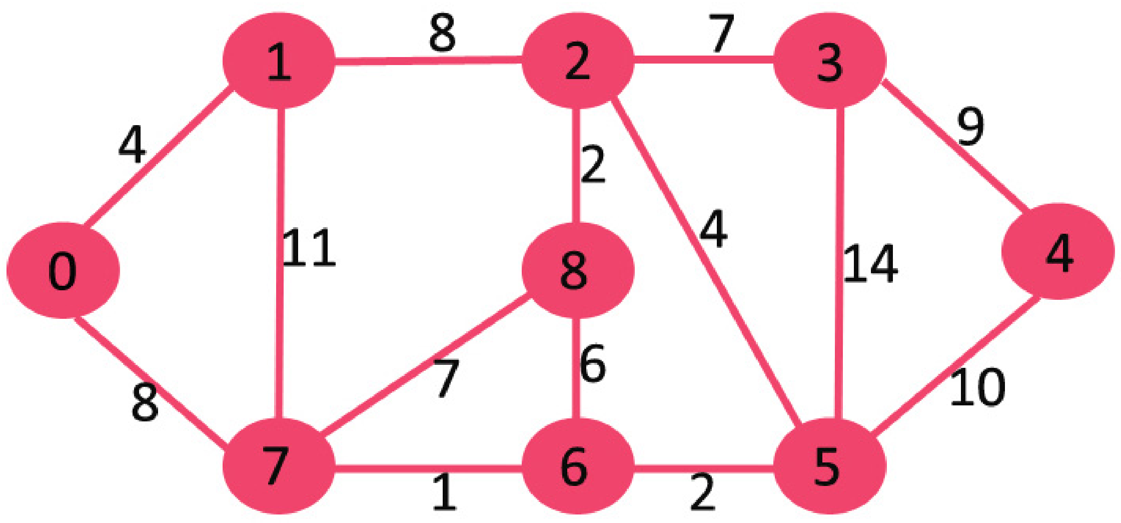
1.2 Min-Cost Network Flow Concepts and Definitions

- In Lecture 3, we define the concept of "bipartite graphs." Is the second graph given on that slide bipartite?

1.2.1 Answer:

No. The graph shown on the slide is not bipartite. To see this, try coloring node 1 "red." Then all nodes adjacent to node 1 (3, 6) are "blue." Nodes adjacent to 3 and 6 (2, 5, 7, 8) must all be "red." But 7 and 8 are adjacent, so they cannot both be red. Therefore, it is not possible to create a partition where no nodes in the same set are adjacent. This is the result of the 3 nodes 6, 7, and 8 all being connected to each other. If we ignore the directions of the arrows (so imagine all edges instead of arcs), we call this set of three nodes an "odd cycle" (a cycle of nodes with an odd number of edges) or a "clique" (all nodes connected to each other).

- Find a matching for the following graph:

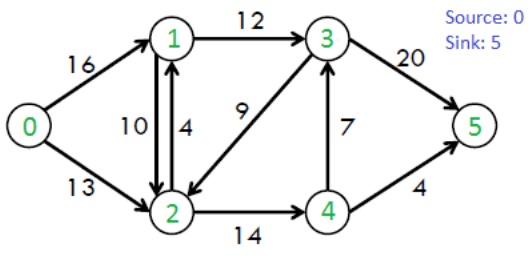


Is it a perfect matching?

1.2.2 Answer

There are many correct answers to this question. One possible matching is the following set of edges: {(0,1), (7,8), (2,3), (6,5)}. This is not a perfect matching because node 4 does not have any edges of the matching incident upon it.

- Build an incidence matrix for the following graph:



1.2.3 Answer

We build an incidence matrix with nodes on the rows and arcs on the columns:

Node/Arc	(0,1)	(0,2)	(1,2)	(1,3)	(2,1)	(2,4)	(3,2)	(3,5)	(4,3)	(4,5)
0	1	1	0	0	0	0	0	0	0	0
1	-1	0	1	1	-1	0	0	0	0	0
2	0	-1	-1	0	1	1	-1	0	0	0
3	0	0	0	-1	0	0	1	1	-1	0
4	0	0	0	0	0	-1	0	0	1	1
5	0	0	0	0	0	0	0	-1	0	-1

- Is a balanced MCNF always feasible?

1.2.4 Answer

Flow balance constraints always satisfied, but could have lb > ub in capacity constraints on an arc, making it infeasible!

- Find a better lower bound on the "Building a Stadium" longest path example given in Lecture 6.

1.2.5 Answer

There are many possible answers to this question. One better lower bound is given by the sequence S-1-2-4-15-F. This sequence has a length of 30 weeks.

1.3 Modeling practice

1.3.1 Blending Problem

The company Steelco has received an order for 500 tons of steel to be used in shipbuilding. The steel must have the following characteristics:

Chemical Element	Minimum Grade (%)	Maximum Grade (%)
Carbon (C)	2	3
Copper (Cu)	0.4	0.6
Manganese (Mn)	1.2	1.65

The company has seven different raw materials in stock that may be used for the production of this steel. The following table lists the grades, available amounts and prices for all materials:

Raw material	C%	Cu%	Mn%	Availability (tons)	Cost ($/ton)
Iron alloy 1	2.5		1.3	400	200
Iron alloy 2	3		0.8	300	250
Iron alloy 3		0.3		600	150
Copper 1		90		500	200
Copper 2		96	4	200	240
Aluminum 1		0.4	1.2	300	200
Aluminum 2		0.6		250	165

(a) Determine the composition of the steel that minimizes the production cost.

(b) Now you must also produce at least 50 tons of oil. Oil is made up of the same raw materials and has the same min and max chemical grade requirements. Modify your model to include the required production of oil.

1.3.2 Answer

(a)

[1]:

```
using JuMP, Clp

### DATA ###

raw_mat = [:i1, :i2, :i3, :c1, :c2, :a1, :a2]
c_per = Dict(zip(raw_mat, [2.5 3 0 0 0 0 0]))
cu_per = Dict(zip(raw_mat, [0 0 0.3 90 96 0.4 0.6]))
mn_per = Dict(zip(raw_mat, [1.3 0.8 0 0 4 1.2 0]))
avail = Dict(zip(raw_mat, [400 300 600 500 200 300 250]))
cost = Dict(zip(raw_mat, [200 250 150 200 240 200 165]))

chem = [:c, :cu, :mn]
min_grade = Dict(zip(chem,[2 0.4 1.2]))
max_grade = Dict(zip(chem, [3 0.6 1.65]))

demand = 500

### MODEL ###

m = Model(Clp.Optimizer)

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

@objective(m, Min, sum(x[i]*cost[i] for i in raw_mat))

@constraint(m, meet_dem, sum(x) == 500)

@constraint(m, use_avail[i in raw_mat], x[i] <= avail[i])

# for each chemical element, the weighted average chemical / ton must be at␣
 ↪least the minimum grade
@constraint(m, min_req_c, sum(c_per[i]*x[i] for i in raw_mat) >= min_grade[:c] *␣
 ↪500)
@constraint(m, min_req_cu, sum(cu_per[i]*x[i] for i in raw_mat) >= min_grade[:
 ↪cu] * 500)
@constraint(m, min_req_mn, sum(mn_per[i]*x[i] for i in raw_mat) >= min_grade[:
 ↪mn] * 500)

# for each chemical element, the weighted average chemical / ton must be at most␣
 ↪the maximum grade
@constraint(m, max_req_c, sum(c_per[i]*x[i] for i in raw_mat) <= max_grade[:c] *␣
 ↪500)
@constraint(m, max_req_cu, sum(cu_per[i]*x[i] for i in raw_mat) <= max_grade[:
 ↪cu] * 500)
@constraint(m, max_req_mn, sum(mn_per[i]*x[i] for i in raw_mat) <= max_grade[:
 ↪mn] * 500)
```

```

optimize!(m)

println("Use the following tons of each raw material: ")
for i in raw_mat
    println(value(x[i]), " tons of raw material ", i)
end
println()
```

```
Use the following tons of each raw material:
400.0 tons of raw material i1
0.0 tons of raw material i2
39.77630199231043 tons of raw material i3
0.0 tons of raw material c1
2.761272282418735 tons of raw material c2
57.46242572527083 tons of raw material a1
0.0 tons of raw material a2

Coin0506I Presolve 7 (-7) rows, 7 (0) columns and 29 (-7) elements
Clp0006I 0  Obj 7979.9985 Primal inf 1020.1325 (4)
Clp0006I 3  Obj 98121.636
Clp0000I Optimal - objective value 98121.636
Coin0511I After Postsolve, objective 98121.636, infeasibilities - dual 0 (0),
primal 0 (0)
Clp0032I Optimal objective 98121.63579 - 3 iterations time 0.012, Presolve 0.00
```

(b) Adding oil into the mix:

[2]:
```
m = Model(Clp.Optimizer)

@variable(m, x_steel[raw_mat] >= 0)
@variable(m, x_oil[raw_mat] >= 0)

@objective(m, Min, sum((x_steel[i]+x_oil[i])*cost[i] for i in raw_mat))

@constraint(m, meet_steel_dem, sum(x_steel) == 500)
@constraint(m, meet_oil_dem, sum(x_oil) == 50)

@constraint(m, use_avail[i in raw_mat], x_steel[i] + x_oil[i] <= avail[i])

### STEEL GRADE CONSTRAINTS ###
# for each chemical element, the weighted average chemical / ton must be at␣
↪least the minimum grade
@constraint(m, sum(c_per[i]*x_steel[i] for i in raw_mat) >= min_grade[:c] * 500)
@constraint(m, sum(cu_per[i]*x_steel[i] for i in raw_mat) >= min_grade[:cu] *␣
↪500)
@constraint(m, sum(mn_per[i]*x_steel[i] for i in raw_mat) >= min_grade[:mn] *␣
↪500)
```

```
# for each chemical element, the weighted average chemical / ton must be at most␣
↪the maximum grade
@constraint(m, sum(c_per[i]*x_steel[i] for i in raw_mat) <= max_grade[:c] * 500)
@constraint(m, sum(cu_per[i]*x_steel[i] for i in raw_mat) <= max_grade[:cu] *␣
↪500)
@constraint(m, sum(mn_per[i]*x_steel[i] for i in raw_mat) <= max_grade[:mn] *␣
↪500)


### OIL GRADE CONSTRAINTS ###
# for each chemical element, the weighted average chemical / ton must be at␣
↪least the minimum grade
@constraint(m, min_req_c, sum(c_per[i]*x_oil[i] for i in raw_mat) >= min_grade[:
↪c] * 50)
@constraint(m, min_req_cu, sum(cu_per[i]*x_oil[i] for i in raw_mat) >=␣
↪min_grade[:cu] * 50)
@constraint(m, min_req_mn, sum(mn_per[i]*x_oil[i] for i in raw_mat) >=␣
↪min_grade[:mn] * 50)

# for each chemical element, the weighted average chemical / ton must be at most␣
↪the maximum grade
@constraint(m, max_req_c, sum(c_per[i]*x_oil[i] for i in raw_mat) <= max_grade[:
↪c] * 50)
@constraint(m, max_req_cu, sum(cu_per[i]*x_oil[i] for i in raw_mat) <=␣
↪max_grade[:cu] * 50)
@constraint(m, max_req_mn, sum(mn_per[i]*x_oil[i] for i in raw_mat) <=␣
↪max_grade[:mn] * 50)

optimize!(m)

println("Use the following tons of each raw material to make steel: ")
for i in raw_mat
    println(value(x_steel[i]), " tons of raw material ", i)
end
println()

println("Use the following tons of each raw material to make oil: ")
for i in raw_mat
    println(value(x_oil[i]), " tons of raw material ", i)
end
println()
```

```
Use the following tons of each raw material to make steel:
370.9388016661327 tons of raw material i1
24.217665278222732 tons of raw material i2
29.2089013165561 tons of raw material i3
```

```
0.0 tons of raw material c1
2.729952687336797 tons of raw material c2
72.90467905175173 tons of raw material a1
0.0 tons of raw material a2

Use the following tons of each raw material to make oil:
29.061198333867342 tons of raw material i1
9.115668055110552 tons of raw material i2
0.0 tons of raw material i3
0.0 tons of raw material c1
0.26433835309195775 tons of raw material c2
11.558795257930152 tons of raw material a1
0.0 tons of raw material a2

Coin0506I Presolve 21 (0) rows, 14 (0) columns and 72 (0) elements
Clp0006I 0  Obj 0 Primal inf 1224.9694 (8)
Clp0006I 10  Obj 110325.99
Clp0000I Optimal - objective value 110325.99
Clp0032I Optimal objective 110325.9932 - 10 iterations time 0.002
```

1.3.3 Production Planning Problem

The company Sailco manufactures sailboats. During the next 4 months the company must meet the following demands for their sailboats:

Month	1	2	3	4
Demand	40	60	70	25

At the beginning of Month 1, Sailco has 10 boats in inventory. Each month it must determinehow many boats to produce. During any month, Sailco can produce up to 40 boats with regular labor and an unlimited number of boats with overtime labor. Boats produced with regular labor cost \$400 each to produce, while boats produced with overtime labor cost \$450 each. It costs \$20 to hold a boat in inventory from one month to the next. Build the LP model that determines the production and inventory schedule that minimizes the cost of meeting the next 4 months demands. Solve the model in Julia.

- Now modify your model to allow backlogging. It costs \$30 to backlog a sailboat every month. Build the LP model that incorporates backlogging and solve it in Julia.

1.3.4 Answer

[3]:
```julia
using Clp, JuMP

### NO BACKLOGGING ###

### DATA ###

```

```julia
months = [:1, :2, :3, :4]
demand = Dict(zip(months,[40 60 70 25]))

i_0 = 10 # starting inventory

### MODEL ###

m = Model(Clp.Optimizer)

@variable(m, prod_reg[i in months] >= 0) # boats produced with regular labor
@variable(m, prod_ot[i in months] >= 0) # boats produced with overtime labor
@variable(m, inv[i in months] >= 0) # inventory at end of each month

@objective(m, Min, 400*sum(prod_reg) + 450*sum(prod_ot) + 20*sum(inv))

@constraint(m, init_inv, i_0 + prod_reg[1]+prod_ot[1] == inv[1] + demand[1])
@constraint(m, inv_bal[i in months[2:4]], inv[i-1] + prod_reg[i]+prod_ot[i] == 
 ↪inv[i] + demand[i])

@constraint(m, labor_con[i in months], prod_reg[i] <= 40)

optimize!(m)

println("Produce the following boats each month: ")
for i in months
    println(value(prod_reg[i]), " boats with regular labor, ", 
 ↪value(prod_ot[i]), " boats with overtime labor")
end
println("Inventory each month: ")
for i in months
    println(value(inv[i]), " boats in inventory")
end
println()
```

```
Produce the following boats each month:
40.0 boats with regular labor, 0.0 boats with overtime labor
40.0 boats with regular labor, 10.0 boats with overtime labor
40.0 boats with regular labor, 30.0 boats with overtime labor
25.0 boats with regular labor, 0.0 boats with overtime labor
Inventory each month:
10.0 boats in inventory
0.0 boats in inventory
0.0 boats in inventory
0.0 boats in inventory

Coin0506I Presolve 4 (-4) rows, 12 (0) columns and 15 (-4) elements
Clp0006I 0  Obj 0 Primal inf 185 (4)
```

```
Clp0006I 5  Obj 76200
Clp0000I Optimal - objective value 76200
Coin0511I After Postsolve, objective 76200, infeasibilities - dual 0 (0), primal
0 (0)
Clp0032I Optimal objective 76200 - 5 iterations time 0.002, Presolve 0.00
```

[12]:
```
### NOW WITH BACKLOGGING ###

m = Model(Clp.Optimizer)

@variable(m, prod_reg[i in months] >= 0) # boats produced with regular labor
@variable(m, prod_ot[i in months] >= 0) # boats produced with overtime labor
@variable(m, inv[i in months] ) # inventory at end of each month (can be␣
 ↪negative)
@variable(m, L[i in months] >= 0) # leftover inventory at end of each month
@variable(m, S[i in months] >= 0) # shortage each month

@objective(m, Min, 400*sum(prod_reg) + 450*sum(prod_ot) + 20*sum(L) + 30*sum(S))

@constraint(m, inv_id[i in months], inv[i] == L[i] - S[i])
@constraint(m, meet_eventually, inv[4] >= 0)
@constraint(m, init_inv, i_0 + prod_reg[1]+prod_ot[1] == inv[1] + demand[1])
@constraint(m, inv_bal[i in months[2:4]], inv[i-1] + prod_reg[i]+prod_ot[i] ==␣
 ↪inv[i] + demand[i])

@constraint(m, labor_con[i in months], prod_reg[i] <= 40)

optimize!(m)

println("Produce the following boats each month: ")
for i in months
    println(value(prod_reg[i]), " boats with regular labor, ",␣
 ↪value(prod_ot[i]), " boats with overtime labor")
end
println("Inventory each month: ")
for i in months
    println(value(inv[i]), " boats in inventory")
end
println()
```

```
Produce the following boats each month:
40.0 boats with regular labor, 0.0 boats with overtime labor
40.0 boats with regular labor, 10.0 boats with overtime labor
40.0 boats with regular labor, 15.0 boats with overtime labor
40.0 boats with regular labor, 0.0 boats with overtime labor
Inventory each month:
10.0 boats in inventory
0.0 boats in inventory
```

```
-15.0 boats in inventory
0.0 boats in inventory

Coin0506I Presolve 4 (-9) rows, 15 (-5) columns and 21 (-11) elements
Clp0006I 0  Obj 0 Primal inf 215 (4)
Clp0006I 6  Obj 75900
Clp0000I Optimal - objective value 75900
Coin0511I After Postsolve, objective 75900, infeasibilities - dual 0 (0), primal
0 (0)
Clp0032I Optimal objective 75900 - 6 iterations time 0.002, Presolve 0.00
```

1.3.5 Shipping Plushies

Suppose you are an optimization expert working for popular plushie manufacturer, Assemble-an-Animal. You have been tasked with figuring out how to distribute plushies. You have a total of 160 orders, which you are shipping from the manufacturing site to four different warehouses in Milwaukee, Paris, Sydney, and New Delhi. The plushies will then be shipped to three different distribution centers in Amsterdam, London, and Beijing. Each warehouse can hold a maximum number of orders (shown in table below). The plushies can either be flown to the distribution centers or taken by train. Because of travel distance restrictions, your deliveries must abide by the following rules: * Milwaukee can only deliver to Amsterdam and London and only by air. * Paris can deliver to Amsterdam and London by air or rail and Beijing by air. * Sydney can only deliver to Beijing, and only by air. * New Delhi can deliver to Beijing by rail, Amsterdam by rail or air, and London by air.

Costs of rail and air transport per order are in the table below.

The contract you have with the airline company requires that any single shipment contains no more than 45 orders. The only restriction on rail transport is that no more than 60 orders can fit on a single shipment.

In addition, to maintain good favor with your contracted airline, you must send at least 15 orders by air to Beijing. 5 orders by air to Amsterdam, and 10 orders by air to London.

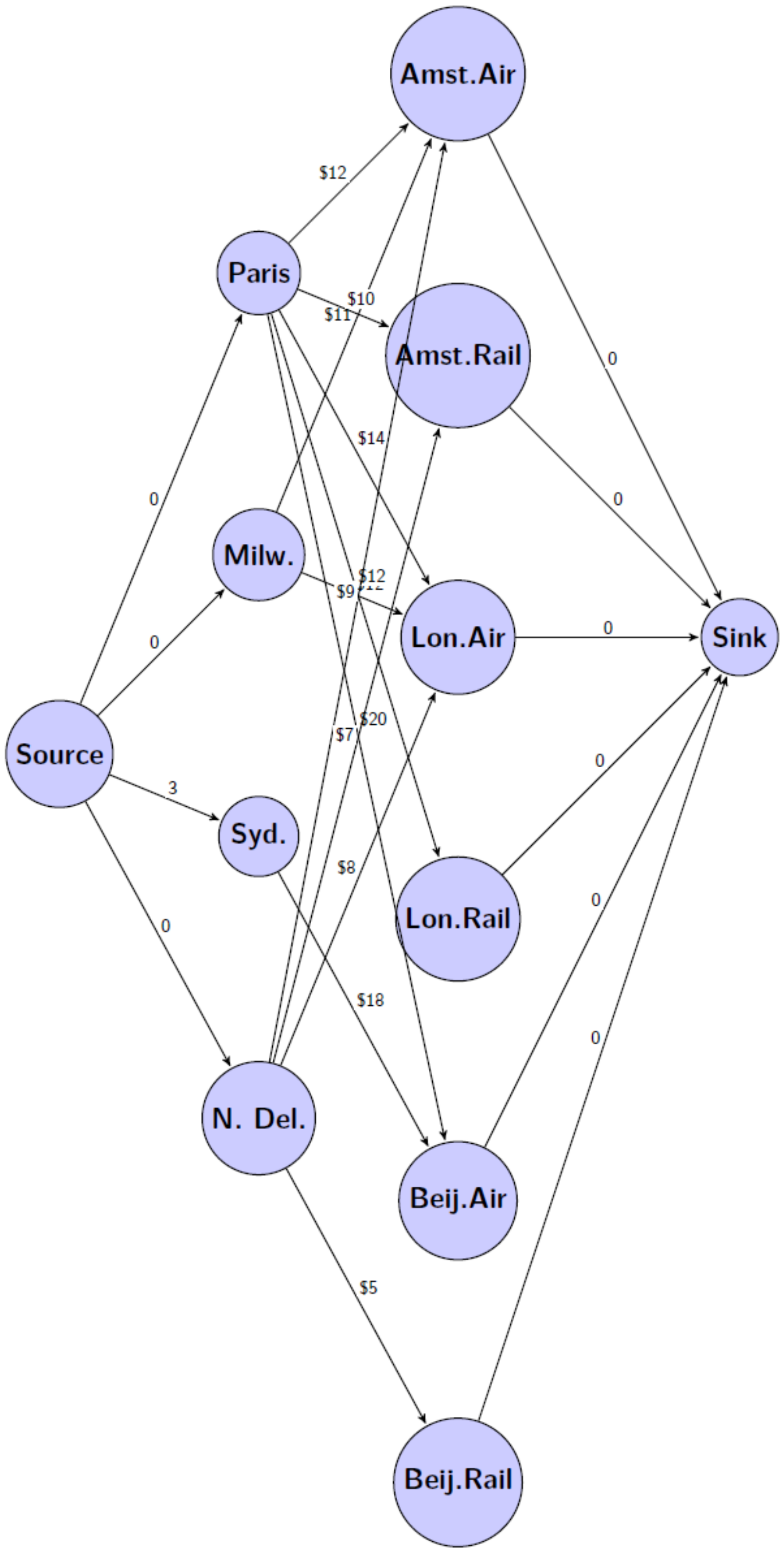
Warehouse/Distribution Center	Amsterdam air (rail)	London air (rail)	Beijing air (rail)	Warehouse capacity
Milwaukee	\$12k (-)	\$11k (-)	- (-)	45
Paris	\$12k (\$10k)	\$14k (\$12k)	\$20k (-)	50
Sydney	- (-)	- (-)	\$18k (-)	70
New Delhi	\$9k (\$7k)	\$8k (-)	- (\$5k)	30

- What type of network flow problem is this?
- Draw the network. How (if at all) do you need to modify the network to fit the structure of the problem you named above?
- Build and solve a MCNF problem to determine how you should transport your 160 orders to minimize the total transportation cost. You can use the code snippet below to initilize your problem:

1.3.6 Answer

This is a transshipment problem with edge capacities. Create a dummy source that supplies 160 orders and a dummy sink that receives all orders. Create a set of destinations that is indexed by city and transportation (e.g., Amsterdam-Air, Amsterdam-Rail,…). Connect warehouses to destinations as given in the table above.

The resulting network is:



```
[4]: num_orders = 160 # total orders
     w = [:M, :P, :S, :N] # warehouses
```

```
dc = [:A, :L, :B] # distribution centers
modes = [:air, :rail] # possible modes of transportation

max_w_cap = Dict(zip(w,[45,50,70,30])) # capacity of each warehouse

nodes = [:s; w; :AA; :AR; :LA; :LR; :BA; :BR; :t] # nodes in the network. index␣
 ↪destinations by both air and rail
# make a list of arcs (as tuples) that exist in the network
arcs = [(:s,:M),(:s,:P),(:s,:S),(:s,:N), (:M,:AA), (:M,:LA),(:P,:AA),(:P,:AR),(:
 ↪P,:LA),(:P,:LR),(:P,:BA),
        (:S,:BA),(:N,:AA),(:N,:AR), (:N,:LA), (:N,:BR),(:AA,:t),(:AR,:t),(:LA,:
 ↪t),(:LR,:t),(:BA,:t),(:BR,:t)]

# dictionary of travel cost (per order) on each arc
costs = Dict(zip(arcs,[0 0 0 0 12 11 12 10 14 12 20 18 9 7 8 5 0 0 0 0 0 0]))
#max capacity of each arc
max_cap = Dict(zip(arcs,[45 50 70 30 45 45 45 60 45 60 45 45 45 60 45 60 160 160␣
 ↪160 160 160 160]))

using JuMP, Clp

m = Model(Clp.Optimizer)

@variable(m, x[arcs] >= 0) # variable for number of orders sent on each arc

# minimize the total cost of shipping orders
@objective(m, Min, sum(costs[a]*x[a] for a in arcs))

# for each node
for i in nodes
    # if node is not source or sink
    if i != :s && i != :t
        # balance flow into and out of node
        @constraint(m, sum(x[j] for j in arcs if j[2] == i) == sum(x[j] for j in␣
 ↪arcs if j[1] == i))
    end
    if i == :s # node is source
        # balance flow out with number of orders (160)
        @constraint(m,sum(x[j] for j in arcs if j[1] == i) == 160)
    end
    if i == :t # node is sink
        # balance flow in with number of orders (160)
        @constraint(m,sum(x[j] for j in arcs if j[2] == i) == 160)
    end

end
# don't exceed capacity restrictions on each arc
```

```
for i in arcs
    @constraint(m, x[i] <= max_cap[i])
end
# ship at least 15 orders by air to Beijing
@constraint(m, sum(x[i] for i in arcs if i[2] == :BA) >= 15)
# ship at least 5 orders by air to Amsterdam
@constraint(m, sum(x[i] for i in arcs if i[2] == :AA) >= 5)
# ship at least 10 orders by air to London
@constraint(m, sum(x[i] for i in arcs if i[2] == :LA) >= 10)
# supply is total number of orders
@constraint(m, sum(x[i] for i in arcs if i[1] == :s) == num_orders)

optimize!(m)

println("Shipping cost: \$", objective_value(m))
println("Number of orders received by sink (should be 160): ", sum(value(x[i])
 ↪for i in arcs if i[2] == :t))
println("Ship orders as follows: ")
for a in arcs

    println(" on arc ", a, " send ", value(x[a]))
end
println()
```

```
Shipping cost: $1780.0
Number of orders received by sink (should be 160): 160.0
Ship orders as follows:
 on arc (:s, :M) send 45.0
 on arc (:s, :P) send 50.0
 on arc (:s, :S) send 35.0
 on arc (:s, :N) send 30.0
 on arc (:M, :AA) send 5.0
 on arc (:M, :LA) send 40.0
 on arc (:P, :AA) send 0.0
 on arc (:P, :AR) send 50.0
 on arc (:P, :LA) send 0.0
 on arc (:P, :LR) send 0.0
 on arc (:P, :BA) send 0.0
 on arc (:S, :BA) send 35.0
 on arc (:N, :AA) send 0.0
 on arc (:N, :AR) send 0.0
 on arc (:N, :LA) send 0.0
 on arc (:N, :BR) send 30.0
 on arc (:AA, :t) send 5.0
 on arc (:AR, :t) send 50.0
 on arc (:LA, :t) send 40.0
 on arc (:LR, :t) send 0.0
 on arc (:BA, :t) send 35.0
 on arc (:BR, :t) send 30.0

Coin0506I Presolve 7 (-31) rows, 12 (-10) columns and 30 (-48) elements
Clp0006I 0  Obj 628.19997 Primal inf 265.2 (7)
Clp0006I 10  Obj 1780
Clp0000I Optimal - objective value 1780
Coin0511I After Postsolve, objective 1780, infeasibilities - dual 0 (0), primal
0 (0)
Clp0032I Optimal objective 1780 - 10 iterations time 0.002, Presolve 0.00
```

1.3.7 Project Planning

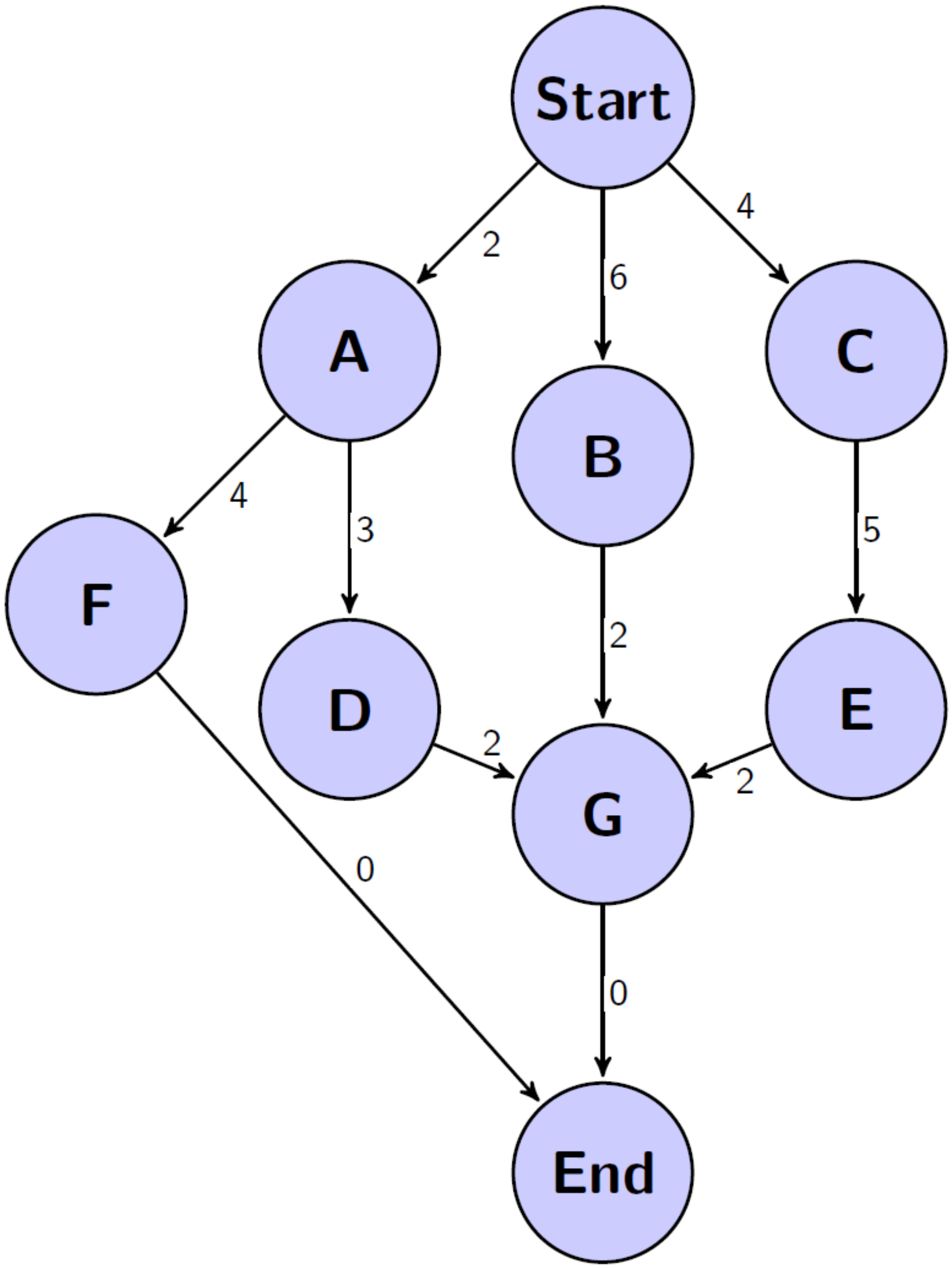
Suppose you are making a plan for completing your final project in CS 524 this summer. You've come up with a list of tasks, their predecessor relationships, and the duration (in days) of each task you must complete before handing in your project. You've constructed the following table containing the relevant information:

Activity	Description	Predecessor	Duration
A	Build model	n/a	2
B	Write report intro	n/a	6
C	Collect data	n/a	4
D	Run/debug model	A	3
E	Incorporate data in model	C	5
F	Perform sensitivity analysis	A	4
G	Finish report	B, D, E	2

- Draw the network of this project planning problem
- Build and solve a linear programming model that will allow you to determine when you need to start your project to ensure you'll be done by the due date (in terms of days before project is due – e.g., must start at least 5 days before due date).

1.3.8 Answer

Here is the network:



```
[5]: using JuMP, Clp

     ### DATA ###

     T = [:Start, :A, :B, :C, :D, :E, :F, :G, :End]
     dur = Dict(zip(T,[0 2 6 4 3 6 4 2 0]))
```

```julia
using NamedArrays

# initialize a table of which tasks precede others with all 0s
zero_list = zeros(length(T),length(T))
pred_list = NamedArray(zero_list, (T,T), ("Task", "Task"))

pred_list[:Start,:A] = 1; pred_list[:Start,:B] = 1; pred_list[:Start,:C] = 1;
pred_list[:A, :D] = 1; pred_list[:C, :E] = 1; pred_list[:A, :F] = 1;
pred_list[:B,:G] = 1; pred_list[:D, :G] = 1; pred_list[:E, :G] = 1;
pred_list[:G, :End] = 1; pred_list[:F, :End] = 1


### MODEL ###

m = Model(Clp.Optimizer)

@variable(m, x[T] >= 0) # variable for the time we begin each task

# create constraint for every pair of tasks (i,j) where task j is preceded by␣
 ↪task i
@constraint(m,constr[i in T, j in T; pred_list[i,j] == 1], x[j] >= x[i] + dur[i])


# minimize the time we start task F (finish project)
@objective(m, Min, x[:End])

# solve this isntance of the longest path problem
optimize!(m)

# record the value of the variables
xsol = value.(x)

println("Earliest completion time: ", objective_value(m), " days")
for i in T
    println("Start task ", i , " in day ", xsol[i])
end

println()
```

```
Earliest completion time: 12.0 days
Start task Start in day 0.0
Start task A in day 0.0
Start task B in day 4.0
Start task C in day 0.0
Start task D in day 7.0
Start task E in day 4.0
Start task F in day 8.0
Start task G in day 10.0
Start task End in day 12.0

Coin0506I Presolve 0 (-11) rows, 0 (-9) columns and 0 (-22) elements
Clp3002W Empty problem - 0 rows, 0 columns and 0 elements
Clp0000I Optimal - objective value 12
Coin0511I After Postsolve, objective 12, infeasibilities - dual 0 (0), primal 0
(0)
Clp0032I Optimal objective 12 - 0 iterations time 0.002, Presolve 0.00
```

[ ]: