Solved for a class exercise in the course 'Optimization And Simulation Methods.'

-- A. Nambiar

The Minimum Cost Flow Problem

Problem:

Given a network, with a cost c_ij , upper bound u_ij , and lower bound 0 associated with each directed path (i, j), and supply of, or demand for, b(i) units of some commodity at each node.

Minimize the cost flow of the network.

Problem Restated

In simpler terms, we are given a network for the flow of some units of a commodity. There are set paths (i, j), each with a cost (c_ij) for taking the path. The paths also have an upper limit (u_ij) for the amount of commodity that can be taken through.

Furthermore, the network has nodes, which can be thought of as trading ports. Each node, has a demand or supply for b(i) units of some commodity.

Satisfying these conditions, minimize the cost flow of the network.

Problem Data

In [57]:
```
#import Pkg
#using Pkg
#Pkg.add("CSV")
#Pkg.add("DataFrames")
#Pkg.add("ImageView")
```

In [44]:
```
using CSV
using DataFrames

df = CSV.read("/Users/ankitanambiar/Desktop/simple_network.csv", DataFrame)
print(df)
```

8×4 DataFrame

Row	start_node_i	end_node_j	c_ij	u_ij
	Int64	Int64	Int64	Float64
1	1	2	2	Inf
2	1	3	5	Inf
3	2	3	3	Inf
4	3	4	1	Inf
5	3	5	2	1.0
6	4	1	0	Inf
7	4	5	2	Inf
8	5	2	4	Inf

Network Image

The image of the network can be found on the project's Github profile.

The arrows indicate possible paths in the network. The numbers on each arrow represent the cost of taking the path.

The circles with numbers represent the Node values. The values outside of the circles represent the units of some commodity that the Node will take when it is passed through.

Solution (Preperation for Linear-Programming)

Variable Creation

Each variable is the path (i, j) between 2 nodes (visualized by an arrow).

Ex: x1 is the path from Node 1 to Node 2.

x1 = (1,2)

x2 = (1,3)

x3 = (2,3)

x4 = (3,4)

x5 = (3,5)

x6 = (4,5)

x7 = (4,1)

x8 = (5,2)

Objective Function Creation

The objective function will find the minimized cost of using the above paths. In the equation, each path is multiplied by the corresponding cost, as seen in the network image.

For example, Path x1 (1,2) has a cost of 2, so it's cost is represented by 2x1.

Minimize, $2x1 + 5$x2 + $3x3 + 1$x4 + $2x5 + 2$x6 + $0x7 + 4$x8

Constraint Creation

Set constraints. Make sure the movement of commodities meet the Node Limits.

Node 1 Example

At Node 1, 5 goods are kept. To meet the limit, check the paths that lead into and out of Node 1.

Paths moving into Node 1: Path from Node 4 to Node 1 (x7)

Paths moving out of Node 1: Path from Node 1 to 3 (x2), Path from Node 1 to 2 (x1)

Since Node 1 takes 5 goods, the total goods at Node 1 is: x7 - x1 - x2 - 5 = 0, as written below.

Total Constraints

Node 1: x7 - x1 - x2 = 5

Node 2: x1 + x8 - x3 = -10

Node 3: x3 + x2 - x4 - x5 = 0

Node 4: x4 - x7 - x6 = 2

Node 5: x5 + x6 - x8 = 3

Upper Goods Limit for Path 5: x5 <= 1

Solution in Julia Code

import Optimization packages into Julia

In [59]:
```
#import Pkg
#Pkg.add("JuMP")
#Pkg.add("Clp")
#Pkg.add("PyPlot")
#Pkg.add("GLPK")
```

In [21]:
```
using JuMP, GLPK

myModel = Model(GLPK.Optimizer)
@variable(myModel, x1 >= 0)
@variable(myModel, x2 >= 0)
@variable(myModel, x3 >= 0)
@variable(myModel, x4 >= 0)
@variable(myModel, x5 >= 0)
@variable(myModel, x6 >= 0)
@variable(myModel, x7 >= 0)
@variable(myModel, x8 >= 0)

@constraint(myModel, x7 - x1 - x2 == 5)
@constraint(myModel, x4 - x7 - x6 == 2)
@constraint(myModel, x3 + x2 - x4 - x5 == 0)
@constraint(myModel, x1 + x8 - x3 == -10)
@constraint(myModel, x5 + x6 - x8 == 3)
@constraint(myModel, x5 <= 1)

@objective(myModel, Min, 2*x1 + 5*x2 + 3*x3 + 1*x4 + 2*x5 + 2*x6 + 0*x7 + 4*x8)

print(myModel)
```

$$
\begin{aligned}
\min \quad & 2x1 + 5x2 + 3x3 + x4 + 2x5 + 2x6 + 4x8 \\
\text{Subject to} \quad & -x1 - x2 + x7 = 5.0 \\
& x4 - x6 - x7 = 2.0 \\
& x2 + x3 - x4 - x5 = 0.0 \\
& x1 - x3 + x8 = -10.0 \\
& x5 + x6 - x8 = 3.0 \\
& x5 \leq 1.0 \\
& x1 \geq 0.0 \\
& x2 \geq 0.0 \\
& x3 \geq 0.0 \\
& x4 \geq 0.0 \\
& x5 \geq 0.0 \\
& x6 \geq 0.0 \\
& x7 \geq 0.0 \\
& x8 \geq 0.0
\end{aligned}
$$

In [22]:
```
@time begin
status = optimize!(myModel)
end
println("Objective value: ", JuMP.objective_value(myModel))
println("x1 = ", JuMP.value(x1))
println("x2 = ", JuMP.value(x2))
println("x3 = ", JuMP.value(x3))
println("x4 = ", JuMP.value(x4))
println("x5 = ", JuMP.value(x5))
println("x6 = ", JuMP.value(x6))
println("x7 = ", JuMP.value(x7))
println("x8 = ", JuMP.value(x8))
```

```
  0.002423 seconds (240 allocations: 15.516 KiB)
Objective value: 45.0
x1 = 0.0
x2 = 0.0
x3 = 10.0
x4 = 9.0
x5 = 1.0
x6 = 2.0
x7 = 5.0
x8 = 0.0
```

Solution Explained

The solution shows us that the path usage at the equilibrium state where the minimimum cost flow occurs.

As seen above, with linear programming, I found:

Objective value: 45.0

x1 = 0.0

x2 = 0.0

x3 = 10.0

x4 = 9.0

x5 = 1.0

x6 = 2.0

x7 = 5.0

x8 = 0.0

The minimum cost for the network's flow is 45.

The minimized flow, only uses paths x3, x4, x5, x6, and x7.

The values for each variable represent the number of times the path is used in the flow:

x3 = (2,3) = 10 times

x4 = (3,4) = 9 times

x5 = (3,5) = 1 time

x6 = (4,5) = 2 times

x7 = (4,1) = 5 times

The objective value for the minimized cost equals the sum of the cost of using paths x3, x4, x5, x6, and x7.

Objective function:

2*x1* + *5*x2 + 3*x3* + *1*x4 + 2*x5* + *2*x6 + 0*x7* + *4*x8

––>

2(0) + 5(0) + 3(10) + 1(9) + 2(10) + 2(2) + 0(5) + 4(0) = 45