Beer Flow Routing

The Beer Flow Routing problem combines several classic Operations Research problems:

- Shortest Path
- Single Commodity Flow
- Multi Commodity Flow
- Network Design

Problem 1

- Find the shortest path between node A and node G

Sets

- $b \in Bars = \{"A","B","C","D","E","F","G"\}$

Parameters

- X_coor_b: X coordinate for bar b
- Y_coor_b: Y coordinate for bar b
- $Distance_{b1,b2}$: Distance between bar $b1$ and $b2$
- $Network_{b1,b2}$: Incidence matrix where $Network_{b1,b2} = 1$ if there is a network connection between $b1$ and $b2$

Decision variables

- Flow variable for flow from bar $b1$ to bar $b2$: $x_{b1,b2} \geq 0$

Model

Objective:

- Minimize the number of arcs used from node :

$$\sum_{b1,b2} x_{b1,b2}$$

Constraints:

- For each node (bar) in the network, flow one unit out of node A and flow one unit into node G:

$$1|(b1 = A) + \sum_{b2} x_{b2,b1} = \sum_{b2} x_{b1,b2} + 1|(b1 = G) \ \forall \ b1$$

- Limit flow to existing network:

$$x_{b1,b2} \leq Network_{b1,b2} \ \forall \ b1, b2$$

In the above model we have a balance constraint where the left hand side flows into a node and the right hand side flow out of a node.

The full model in Julia/JuMP, available with the name

`BeerFlowRouting1.jl`

from the book web-site, is given below:

```
#*****************************************************************************
# Beer Flow Routing 1, "Mathematical Programming Modelling" (42112)
# Intro definitions
using JuMP
using HiGHS
#*****************************************************************************

#*****************************************************************************
# Data
Bars=["A","B","C","D","E","F","G"]
B=length(Bars)
X_coor=[ 3.0  9.0 8.0 2.0 6.0 10.0 11.0]
Y_coor=[10.0 10.0 6.0 5.0 2.0  2.0  7.0]
Distance = zeros(Float64,B,B)
for b1 in 1:B
```

```
    for b2 in 1:B
        Distance[b1,b2]= sqrt( (X_coor[b1]-X_coor[b2])*(X_coor[b1]-X_coor[b2]) + (Y_coor[b1]
    end
end

Network=zeros(Int64,B,B)

# A-B
Network[1,2]=1
Network[2,1]=1

# A-C
Network[1,3]=1
Network[3,1]=1

# A-D
Network[1,4]=1
Network[4,1]=1

# B-C
Network[2,3]=1
Network[3,2]=1

# C-E
Network[3,5]=1
Network[5,3]=1

# C-F
Network[3,6]=1
Network[6,3]=1

# F-G
Network[6,7]=1
Network[7,6]=1
#**************************************************************************

#**************************************************************************
# Model
BN = Model(HiGHS.Optimizer)

# flow from node i to node j
@variable(BN, x[i=1:B,j=1:B] >= 0)

# Minimize path cost
@objective(BN,
           Min, sum( x[i,j] for i=1:B,j=1:B)
```

```
            )

@constraint(BN, [i=1:B],
             (i==1 ? 1 : 0) + sum( x[j,i] for j=1:B) ==
             sum( x[i,j] for j=1:B) + (i==B ? 1 : 0)
             )

@constraint(BN, [i=1:B,j=1:B],
             x[i,j] <= Network[i,j]
             )
#*****************************************************************************

#*****************************************************************************
# solve
optimize!(BN)
println("Termination status: $(termination_status(BN))")
#*****************************************************************************

#*****************************************************************************
# Report results
if termination_status(BN) == MOI.OPTIMAL
    println("RESULTS:")
    println("BN result: $(objective_value(BN))")
    for i=1:B
        for j=1:B
            if value(x[i,j]) > 0.001
                println("$(Bars[i]) -> $(Bars[j]) val: $(value(x[i,j]))")
            end
        end
    end
else
  println("  No solution")
end
#*****************************************************************************
```

Comments to the Julia/JuMP program:

- Notice: The use of **conditional parts of a constraint**:

  ```
  1|(b1=A)
  ```

Problem 2

- Find the shortest single commodity flow distance for delivery from node A to all the others

New Parameters

- $Demand_b$: The demand from node A to node b
- $PipeCap = 10$: The amount of beer which can be flow through a pipe

Model

Objective:

- Minimize the summed flow distance :

$$\sum_{b1,b2} Distance_{b1,b2} \cdot x_{b1,b2}$$

Constraints:

- For each node (bar) in the network, flow one unit out of node A and flow one unit into node G:

$$\sum_{b2} x_{b2,b1} - \sum_{b2} x_{b1,b2} = Demand_{b1} \ \forall \ b1$$

- Limit flow to existing network:

$$x_{b1,b2} \leq PipeCap \cdot Network_{b1,b2} \ \forall \ b1, b2$$

In the above model the balance constraint is changed by including the $Demand_b$, which is -1 for node A and 1 for all the others

The full model in Julia/JuMP, available with the name

`BeerFlowRouting2.jl`

from the book web-site, is given below:

```
#*****************************************************************
# Beer Flow Routing 2, "Mathematical Programming Modelling" (42112)
# Intro definitions
using JuMP
using HiGHS
#*****************************************************************

#*****************************************************************
# Data
```

```
Bars=["A","B","C","D","E","F","G"]
B=length(Bars)
X_coor=[3 9 8 2 6 10 11]
Y_coor=[10 10 6 5 2 2 7]
Distance = zeros(Float64,B,B)
for b1 in 1:B
    for b2 in 1:B
        Distance[b1,b2]= sqrt( (X_coor[b1]-X_coor[b2])*(X_coor[b1]-X_coor[b2]) + (Y_coor[b1]
    end
end

Network=zeros(Int64,B,B)

# A-B
Network[1,2]=1
Network[2,1]=1

# A-C
Network[1,3]=1
Network[3,1]=1

# A-D
Network[1,4]=1
Network[4,1]=1

# B-C
Network[2,3]=1
Network[3,2]=1

# C-E
Network[3,5]=1
Network[5,3]=1

# C-F
Network[3,6]=1
Network[6,3]=1

# F-G
Network[6,7]=1
Network[7,6]=1

Demand=zeros(Int64,B)

# A
Demand[1]=-6
```

```
# B
Demand[2]=1

# C
Demand[3]=1

# D
Demand[4]=1

# E
Demand[5]=1

# F
Demand[6]=1

# G
Demand[7]=1

PipeCap=10

#*****************************************************************************

#*****************************************************************************
# Model
BN = Model(HiGHS.Optimizer)

# flow from node i to node j
@variable(BN, x[i=1:B,j=1:B] >= 0)

# Minimize path cost
@objective(BN,
           Min, sum( Distance[i,j]*x[i,j] for i=1:B,j=1:B)
           )

@constraint(BN, [i=1:B],
            # Going into i             Going out of i           demand, negative for going
            sum( x[j,i] for j=1:B) - sum( x[i,j] for j=1:B) == Demand[i]
            )

@constraint(BN, [i=1:B,j=1:B],
            x[i,j] <= PipeCap*Network[i,j]
            )
#*****************************************************************************

#*****************************************************************************
# solve
```

```
optimize!(BN)
println("Termination status: $(termination_status(BN))")
#*****************************************************************************

#*****************************************************************************
# Report results
if termination_status(BN) == MOI.OPTIMAL
    println("RESULTS:")
    println("BN result: $(objective_value(BN))")
    for i=1:B
        for j=1:B
            if value(x[i,j]) > 0.001
                println("$(Bars[i]) -> $(Bars[j]) val: $(value(x[i,j]))")
            end
        end
    end
else
  println("  No solution")
end
#*****************************************************************************
```

Problem 3

- Find the shortest multi commodity flow distance for delivery from node A and B to all the others

Changed Decision variables

- Flow variable for flow of beer from origin bar o to destination bar d over link from bar $b1$ to bar $b2$: $x^{o,d}_{b1,b2} \geq 0$

New Parameters

- $Demand_{b1,b2}$: The demand from node A and B to all the other nodes
- $PipeCap = 500$: The amount of beer which can be flow through a pipe (increased from 10)

Model

Objective:

- Minimize the summed beer travel distance

$$\sum_{o,d,b1,b2} Distance_{b1,b2} \cdot x_{b1,b2}^{o,d}$$

Constraints:

- For each node (bar) in the network, flow one unit out of node A and flow one unit into node G:

$$\sum_{b2} x_{b2,b1}^{o,d} - \sum_{b2} x_{b1,b2}^{o,d} = Demand_{b1} \ \forall \ o, d, b1$$

- Limit flow to existing network:

$$sum_{o,d} x_{b1,b2}^{o,d} \leq PipeCap \cdot Network_{b1,b2} \ \forall \ b1, b2$$

In the above model the flow is not differentiated in flows from node A to the other nodes and from node B to the other nodes.
The full model in Julia/JuMP, available with the name

```
BeerFlowRouting3.jl
```

from the book web-site, is given below:

```
#*****************************************************************************
# Beer Flow Routing, "Mathematical Programming Modelling" (42112)
# Intro definitions
using JuMP
using HiGHS
#*****************************************************************************

#*****************************************************************************
# Data
Bars=["A","B","C","D","E","F","G"]
B=length(Bars)
X_coor=[3 9 8 2 6 10 11]
Y_coor=[10 10 6 5 2 2 7]
Distance = zeros(Float64,B,B)
for b1 in 1:B
    for b2 in 1:B
        Distance[b1,b2]= sqrt( (X_coor[b1]-X_coor[b2])*(X_coor[b1]-X_coor[b2]) + (Y_coor[b1]
    end
end

Network=zeros(Int64,B,B)
```

```
# A-B
Network[1,2]=1
Network[2,1]=1

# A-C
Network[1,3]=1
Network[3,1]=1

# A-D
Network[1,4]=1
Network[4,1]=1

# B-C
Network[2,3]=1
Network[3,2]=1

# C-E
Network[3,5]=1
Network[5,3]=1

# C-F
Network[3,6]=1
Network[6,3]=1

# F-G
Network[6,7]=1
Network[7,6]=1

Demand=zeros(Int64,B,B)

# A
Demand[1,2]=70

# B
Demand[2,1]=50

# C
Demand[1,3]=80
Demand[2,3]=90

# D
Demand[1,4]=20
Demand[2,4]=50

# E
```

```
Demand[1,5]=80
Demand[2,5]=10

# F
Demand[1,6]=20
Demand[2,6]=100

# G
Demand[1,7]=40
Demand[2,7]=50

PipeCap=500

#*****************************************************************************

#*****************************************************************************
# Model
BN = Model(HiGHS.Optimizer)

# flow from node k to node l on edge from node i to node j
@variable(BN, x[k=1:B,l=1:B,i=1:B,j=1:B] >= 0)

# Minimize path cost
@objective(BN,
           #Min, pc
           Min, sum( Distance[i,j]*x[k,l,i,j] for i=1:B,j=1:B,k=1:B,l=1:B)
           )

@constraint(BN, [i=1:B,k=1:B,l=1:B],
            sum( x[k,l,j,i] for j=1:B) - sum( x[k,l,i,j] for j=1:B) ==
            (i==k ? -Demand[k,l] : 0) + (i==l ? Demand[k,l] : 0)
            )

@constraint(BN, [i=1:B,j=1:B],
            sum( x[k,l,i,j] for k=1:B,l=1:B) <= Network[i,j]*PipeCap
            )
#*****************************************************************************

#*****************************************************************************
# solve
optimize!(BN)
println("Termination status: $(termination_status(BN))")
#*****************************************************************************

#*****************************************************************************
# Report results
```

```
if termination_status(BN) == MOI.OPTIMAL
    println("RESULTS:")
    println("BN result: $(objective_value(BN))")
    for k=1:B
        for l=1:B
            if Demand[k,l]>0.5
                println("Demand: $(Bars[k]) -> $(Bars[l])")
                for i=1:B
                    for j=1:B
                        if value(x[k,l,i,j]) > 0.001
                            println("$(Bars[i]) -> $(Bars[j]) val: $(value(x[k,l,i,j]))")
                        end
                    end
                end
            end
        end
    end
else
  println("  No solution")
end
#*****************************************************************************
```

Problem 4

- Find the shortest multi commodity pipe distance for delivery from node A and B to all the others

Changed Decision variables

- Binary pipe existence variable for flow originating from node o over link from bar $b1$ to bar $b2$: $y^o_{b1,b2} \in \{0,1\}$

Model

Objective:

- Minimize the summed beer travel distance

$$\sum_{o,b1,b2} Distance_{b1,b2} \cdot y^o_{b1,b2}$$

Constraints:

- For each node (bar) in the network, flow one unit out of node A and flow one unit into node G:

$$\sum_{b2} x^{o,d}_{b2,b1} - \sum_{b2} x^{o,d}_{b1,b2} = Demand_{b1} \ \forall \ o, d, b1$$

- Force the creation of the necessary pipes:

$$\sum_{d} x^{o,d}_{b1,b2} \leq PipeCap \cdot y^{o}_{b1,b2} \ \forall \ o, b1, b2$$

Very simple extension: A new variable defining the existing of a pipe between two bars $b1$ and $b2$ for flow between two other bars o and b.

The full model in Julia/JuMP, available with the name

`BeerFlowRouting4.jl`

from the book web-site, is given below:

```
#*****************************************************************************
# Beer Flow Routing, "Mathematical Programming Modelling" (42112)
# Intro definitions
using JuMP
using HiGHS
#*****************************************************************************

#*****************************************************************************
# Data
Bars=["A","B","C","D","E","F","G"]
B=length(Bars)
X_coor=[3 9 8 2 6 10 11]
Y_coor=[10 10 6 5 2 2 7]
Distance = zeros(Float64,B,B)
for b1 in 1:B
    for b2 in 1:B
        Distance[b1,b2]= sqrt( (X_coor[b1]-X_coor[b2])*(X_coor[b1]-X_coor[b2]) + (Y_coor[b1]
    end
end

Network=zeros(Int64,B,B)

# A-B
Network[1,2]=1
Network[2,1]=1
```

```
# A-C
Network[1,3]=1
Network[3,1]=1

# A-D
Network[1,4]=1
Network[4,1]=1

# B-C
Network[2,3]=1
Network[3,2]=1

# C-E
Network[3,5]=1
Network[5,3]=1

# C-F
Network[3,6]=1
Network[6,3]=1

# F-G
Network[6,7]=1
Network[7,6]=1

Demand=zeros(Int64,B,B)

# A
Demand[1,2]=70

# B
Demand[2,1]=50

# C
Demand[1,3]=80
Demand[2,3]=90

# D
Demand[1,4]=20
Demand[2,4]=50

# E
Demand[1,5]=80
Demand[2,5]=10

# F
Demand[1,6]=20
```

```
Demand[2,6]=100

# G
Demand[1,7]=40
Demand[2,7]=50

PipeCap=500

#*****************************************************************************

#*****************************************************************************
# Model
BN = Model(HiGHS.Optimizer)

# flow from node k to node l on edge from node i to node j
@variable(BN, x[k=1:B,l=1:B,i=1:B,j=1:B] >= 0)

# design binary variable
@variable(BN, y[k=1:B,i=1:B,j=1:B], Bin)
for k=1:B
    if sum(Demand[k,l] for l=1:B)==0
        for i=1:B
            for j=1:B
                fix(y[k,i,j],0; force = true)
            end
        end
    end
end

for k=1:B
    for i=1:B
        fix(y[k,i,i],0; force = true)
    end
end

# Minimize path cost
@objective(BN,
           #Min, pc
           Min, sum( Distance[i,j]*y[k,i,j] for k=1:B,i=1:B,j=1:B)
           )

# balance constraint
@constraint(BN, [i=1:B,k=1:B,l=1:B],
            sum( x[k,l,j,i] for j=1:B) - sum( x[k,l,i,j] for j=1:B) ==
            (i==k ? -Demand[k,l] : 0) + (i==l ? Demand[k,l] : 0)
            )
```

```
# network design constraint
@constraint(BN, [k=1:B,i=1:B,j=1:B],
            sum( x[k,l,i,j] for l=1:B) <= PipeCap*y[k,i,j]
            )
#*********************************************************************

#*********************************************************************
# solve
optimize!(BN)
println("Termination status: $(termination_status(BN))")
#*********************************************************************

#*********************************************************************
# Report results
if termination_status(BN) == MOI.OPTIMAL
    println("RESULTS:")
    println("BN result: $(objective_value(BN))")

    for k=1:B
        for i=1:B
            for j=1:B
                if value(y[k,i,j]) > 0.001
                    println("beer: $(k) y:  $(Bars[i]) -> $(Bars[j]) val: $(value(y[k,i,j]))
                end
            end
        end
    end
else
  println("  No solution")
end
#*********************************************************************
```

Problem 5

- Find the shortest digging distance for the beer delivery network

Changed Decision variables

- Binary digging requirement variable: $q_{b1,b2} \in \{0, 1\}$

Model

Objective:

- Minimize the digging distance

$$\sum_{b1,b2} Distance_{b1,b2} \cdot q_{b1,b2}$$

New Constraints:

- For each node (bar) in the network, flow one unit out of node A and flow one unit into node G:

$$\sum_{b2} x^{o,d}_{b2,b1} - \sum_{b2} x^{o,d}_{b1,b2} = Demand_{b1} \ \forall \ o,d,b1$$

- Force the creation of the necessary pipes:

$$\sum_{o,d} x^{o,d}_{b1,b2} \leq PipeCap \cdot y^{o}_{b1,b2} \ \forall \ b1,b2$$

- Force the digging variables:

$$y^{o}_{b1,b2} + y^{o}_{b2,b1} \leq q_{b1,b2} + q_{b2,b1} \ \forall \ o,b1,b2$$

Again a very simple extension: A new variable defining digging between two bars $b1$ and $b2$. Notice that in the model, we only allow one direction digging (see the julia program below):

The full model in Julia/JuMP, available with the name

`BeerFlowRouting5.jl`

from the book web-site, is given below:

```
#************************************************************************
# Beer Flow Routing, "Mathematical Programming Modelling" (42112)
# Intro definitions
using JuMP
using HiGHS
#************************************************************************

#************************************************************************
# Data
```

```
Bars=["A","B","C","D","E","F","G"]
B=length(Bars)
X_coor=[3 9 8 2 6 10 11]
Y_coor=[10 10 6 5 2 2 7]
Distance = zeros(Float64,B,B)
for b1 in 1:B
    for b2 in 1:B
        Distance[b1,b2]= sqrt( (X_coor[b1]-X_coor[b2])*(X_coor[b1]-X_coor[b2]) + (Y_coor[b1]
    end
end

Network=zeros(Int64,B,B)

# A-B
Network[1,2]=1
Network[2,1]=1

# A-C
Network[1,3]=1
Network[3,1]=1

# A-D
Network[1,4]=1
Network[4,1]=1

# B-C
Network[2,3]=1
Network[3,2]=1

# C-E
Network[3,5]=1
Network[5,3]=1

# C-F
Network[3,6]=1
Network[6,3]=1

# F-G
Network[6,7]=1
Network[7,6]=1

Demand=zeros(Int64,B,B)

# A
Demand[1,2]=70
```

```
# B
Demand[2,1]=50

# C
Demand[1,3]=80
Demand[2,3]=90

# D
Demand[1,4]=20
Demand[2,4]=50

# E
Demand[1,5]=80
Demand[2,5]=10

# F
Demand[1,6]=20
Demand[2,6]=100

# G
Demand[1,7]=40
Demand[2,7]=50

PipeCap=500

#*****************************************************************************

#*****************************************************************************
# Model
BN = Model(HiGHS.Optimizer)

# Flow from node k to node l on edge from node i to node j
@variable(BN, x[k=1:B,l=1:B,i=1:B,j=1:B] >= 0)

# design pipe binary variable
@variable(BN, y[k=1:B,i=1:B,j=1:B], Bin)
for k=1:B
    if sum(Demand[k,l] for l=1:B)==0
        for i=1:B
            for j=1:B
                fix(y[k,i,j],0; force = true)
            end
        end
    end
end
```

```
# digging variables: q[i,j]=1, if a digging is done between bar i and bar j
@variable(BN, q[i=1:B,j=1:B], Bin)
for i=1:B
    for j=1:B
        if i>=j
            fix(q[i,j],0; force = true)
        end
    end
end

# Minimize path cost
@objective(BN,
           #Min, pc
           Min, sum( Distance[i,j]*q[i,j] for i=1:B,j=1:B)
           )

# balance constraint
@constraint(BN, [i=1:B,k=1:B,l=1:B],
            sum( x[k,l,j,i] for j=1:B) - sum( x[k,l,i,j] for j=1:B) ==
            (i==k ? -Demand[k,l] : 0) + (i==l ? Demand[k,l] : 0)
            )

# network design constraint
@constraint(BN, [k=1:B,i=1:B,j=1:B],
            sum( x[k,l,i,j] for l=1:B) <= PipeCap*y[k,i,j]
            )


# network design constraint
@constraint(BN, [k=1:B,i=1:B,j=1:B],
            y[k,i,j] + y[k,j,i] <= q[i,j]+q[j,i]
            )
#*****************************************************************************

#*****************************************************************************
# solve
optimize!(BN)
println("Termination status: $(termination_status(BN))")
#*****************************************************************************

#*****************************************************************************
# Report results
if termination_status(BN) == MOI.OPTIMAL
    println("RESULTS:")
    println("BN result: $(objective_value(BN))")
```

```
    for i=1:B
        for j=1:B
            if value(q[i,j]) > 0.001
                println("q:  $(Bars[i]) <-> $(Bars[j]) val: $(value(q[i,j]))  Distance: (Dis
                for k=1:B
                    if value(y[k,i,j]) > 0.001
                        println("  beer: $(k) y:  $(Bars[i]) -> $(Bars[j]) val: $(value(y[k,
                    end
                    if value(y[k,j,i]) > 0.001
                        println("  beer: $(k) y:  $(Bars[j]) -> $(Bars[i]) val: $(value(y[k,
                    end
                end
            end
        end
    end
else
  println("  No solution")
end
#*****************************************************************************
```