Tennis

The Tennis problem is a version of the classic Operations Research problem, the Chinese Postman problem. Chinese because it was first recognized by a Chinese researcher. The postman needs to minimize the traveled route, at the same time he has to traverse certain streets. But the problem can also be found in other situations like snow-plowing, tennis sweeping etc.

Problem Q1

- This problem can easily be solved analytically, hence no Julia/JuMP model. The "interior" rectangle, can be cleaned in one direction sweep, but the rest of the connections needs to be traversed twice.

Problem Q2

- To establish the data, calculate the euclidean distance between all the nodes, based on the node coordinates, and create an incidence matrix with 1 for a connection between them. Then make a model such that all lines are sweeped, and the person returns to the same place.

Sets

- Points: $p \in Points = \{"A0", "B0", "D0", "E0", "B1",$ $"C1", "D1", "A2", "B2", "C2", "D2", "E2"\}$

Parameters

- $Distance_{p1,p2}$: Distance in Feet between node $p1$ and node $p2$
- $Lines_{p1,p2}$: Incidence where $Lines_{p1,p2} = 1$ if there is a line to sweep between node $p1$ and $p2$. Notice, the matrix is oriented.

Decision variables

- Walk from node $p1$ to node $p2$: $x_{p1,p2} \in \{0,1\}$

Model

Objective:

- Minimize the walked distance:

$$\sum_{p1,p2} Distance_{p1,p2} \cdot x_{p1,p2}$$

Constraints:

- Balance constraint: The number of times going into a node should be equal to the number of times going out of a node:

$$\sum_{p2} x_{p2,p1} = \sum_{p2} x_{p1,p2} \ \forall \ p1$$

- All the existing lines have to be cleaned:

$$x_{p1,p2} + x_{p2,p1} \geq Lines_{p1,p2} \ \forall \ p1, p2$$

The above model is actually relatively simple, but requires to return to the start.

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

`Tennis2.jl`

from the book web-site, is given below:

```
#*****************************************************************************
# Tennis 2 Assignment, "Mathematical Programming Modelling" (42112)
# Basic model
using JuMP
using HiGHS
#*****************************************************************************

#*****************************************************************************
# PARAMETERS
#          1    2    3    4    5    6    7    8    9    10   11   12
Points = ["A0" "B0" "D0" "E0" "B1" "C1" "D1" "A2" "B2" "C2" "D2" "E2"]
P=length(Points)
```

```
x_pos = [0    4.5  31.5 36   4.5  18   31.5 0    4.5  18   31.5 36]
y_pos = [0    0    0    0    18   18   18   39   39   39   39   39]
Distance = zeros(P,P)
for p in 1:P
    for pp in 1:P
        Distance[p,pp]= sqrt( (x_pos[p]-x_pos[pp])*(x_pos[p]-x_pos[pp]) + (y_pos[p]-y_pos[pp
    end
end

Lines=zeros(Int8,P,P)
Lines[1,2]=1 #L('A0','B0') = 1;
Lines[1,8]=1 # L('A0','A2') = 1;
Lines[2,5]=1 # L('B0','B1') = 1;
Lines[2,3]=1 # L('B0','D0') = 1;
Lines[3,4]=1 # L('D0','E0') = 1;
Lines[3,7]=1 # L('D0','D1') = 1;
Lines[4,12]=1 # L('E0','E2') = 1;
Lines[5,6]=1 # L('B1','C1') = 1;
Lines[5,9]=1 # L('B1','B2') = 1;
Lines[6,7]=1 # L('C1','D1') = 1;
Lines[6,10]=1 # L('C1','C2') = 1;
Lines[7,11]=1 # L('D1','D2') = 1;
#*************************************************************************

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

@variable(tennis, x[1:P, 1:P], Bin)

# Minimize walking-Distance
@objective(tennis, Min,
           sum(Distance[p,pp]*x[p,pp] for p=1:P, pp=1:P))

# what goes in must come out
@constraint(tennis, [p=1:P],
            sum(x[p,pp] for pp=1:P) == sum(x[pp,p] for pp=1:P))

# force clean lines
@constraint(tennis, [p=1:P,pp=1:P],
            x[p,pp] + x[pp,p] >= Lines[p,pp] )

print(tennis)
#*************************************************************************

#*************************************************************************
```

```
# solve
optimize!(tennis)
println("Termination status: $(termination_status(tennis))")
#########################################################################

#########################################################################
# Report results
println("------------------------------------");
if termination_status(tennis) == MOI.OPTIMAL
    println("RESULTS:")
    println("objective = $(objective_value(tennis))")
else
  println("  No solution")
end
println("------------------------------------");
#########################################################################
```

Problem Q3

- Why should the cleaner return to the start point ? This is not needed, so we change our model by introducing starting point binary variable s_p and ending point binary e_p. This is then used to modify the balance constraint.

Decision variables

- Start at point p: $s_p \in \{0, 1\}$
- End at point p: $e_p \in \{0, 1\}$

Updated Constraints:

- Balance constraint: The number of times going into a node should be equal to the number of times going out of a node:

$$\sum_{p2} x_{p2,p1} + s_p = \sum_{p2} x_{p1,p2} + e_p \ \forall \ p1$$

- You can only start at one point:

$$\sum_{p} s_p = 1 \ \forall \ p1$$

- You can only end at one point:

$$\sum_{p} e_p = 1 \ \forall \ p1$$

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

`Tennis3.jl`

from the book web-site, is given below:

```
#*************************************************************************
# Tennis 3 Assignment, "Mathematical Programming Modelling" (42112)
# Model, where start and stop can occur at different places
using JuMP
using HiGHS
#*************************************************************************

#*************************************************************************
# PARAMETERS
#          1    2    3    4    5    6    7    8    9    10   11   12
Points = ["A0" "B0" "D0" "E0" "B1" "C1" "D1" "A2" "B2" "C2" "D2" "E2"]
P=length(Points)
x_pos = [0    4.5  31.5 36   4.5  18   31.5 0    4.5  18   31.5 36]
y_pos = [0    0    0    0    18   18   18   39   39   39   39   39]
Distance = zeros(P,P)
for p in 1:P
    for pp in 1:P
        Distance[p,pp]= sqrt( (x_pos[p]-x_pos[pp])*(x_pos[p]-x_pos[pp]) + (y_pos[p]-y_pos[pp
    end
end

Lines=zeros(Int8,P,P)
Lines[1,2]=1 #L('A0','B0') = 1;
Lines[1,8]=1 # L('A0','A2') = 1;
Lines[2,5]=1 # L('B0','B1') = 1;
Lines[2,3]=1 # L('B0','D0') = 1;
Lines[3,4]=1 # L('D0','E0') = 1;
Lines[3,7]=1 # L('D0','D1') = 1;
Lines[4,12]=1 # L('E0','E2') = 1;
Lines[5,6]=1 # L('B1','C1') = 1;
Lines[5,9]=1 # L('B1','B2') = 1;
Lines[6,7]=1 # L('C1','D1') = 1;
Lines[6,10]=1 # L('C1','C2') = 1;
Lines[7,11]=1 # L('D1','D2') = 1;
#*************************************************************************


#*************************************************************************
# Model
tennis = Model(HiGHS.Optimizer)
```

```
@variable(tennis, x[1:P, 1:P], Bin)

@variable(tennis, s[1:P], Bin)

@variable(tennis, e[1:P], Bin)

# Minimize walking-distance
@objective(tennis, Min,
           sum(Distance[p,pp]*x[p,pp] for p=1:P, pp=1:P))

# what goes in must come out
@constraint(tennis, [p=1:P],
            sum(x[pp,p] for pp=1:P) + s[p] == sum(x[p,pp] for pp=1:P) + e[p] )

# one start
@constraint(tennis, sum(s[p] for p=1:P) == 1)

# one end
@constraint(tennis, sum(e[p] for p=1:P) == 1)

# force clean lines
@constraint(tennis, [p=1:P,pp=1:P],
            x[p,pp] + x[pp,p] >= Lines[p,pp])

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

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

#**************************************************************************
# Report results
println("---------------------------------------");
if termination_status(tennis) == MOI.OPTIMAL
    println("RESULTS:")
    println("objective = $(objective_value(tennis))")
else
  println("  No solution")
end
println("---------------------------------------");
#**************************************************************************
```

```
#*********************************************************************
println("Successfull end of $(PROGRAM_FILE)")
#*********************************************************************
```

Problem Q4

- When you check the solution, you realize that one of the ending points is an "interior point", i.e. you have to walk across the sweeped area. Hence you want to ban this solution. This turns out to be relatively easy: Identify the critical points: "B1", "C1", "D1", "B2", "C2", "D2". The solutions is simly to require the variables s_p and e_p to take the value 0 for these points. This can be done using constraints, but here we will simply fix the value to 0. See the Julia/JuMP model below.

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

`Tennis4.jl`

from the book web-site, is given below:

```
#*********************************************************************
# Tennis 4 Assignment, "Mathematical Programming Modelling" (42112)
# Model, where start and stop can occur at different places
using JuMP
using HiGHS
#*********************************************************************

#*********************************************************************
# PARAMETERS
#          1    2    3    4    5    6    7    8    9    10   11   12
Points = ["A0" "B0" "D0" "E0" "B1" "C1" "D1" "A2" "B2" "C2" "D2" "E2"]
P=length(Points)
x_pos = [0    4.5  31.5 36   4.5  18   31.5 0    4.5  18   31.5 36]
y_pos = [0    0    0    0    18   18   18   39   39   39   39   39]
Distance = zeros(P,P)
for p in 1:P
    for pp in 1:P
        Distance[p,pp]= sqrt( (x_pos[p]-x_pos[pp])*(x_pos[p]-x_pos[pp]) + (y_pos[p]-y_pos[pp
    end
end

Lines=zeros(Int8,P,P)
Lines[1,2]=1 #L('A0','B0') = 1;
Lines[1,8]=1 # L('A0','A2') = 1;
Lines[2,5]=1 # L('B0','B1') = 1;
```

```
Lines[2,3]=1 # L('B0','D0') = 1;
Lines[3,4]=1 # L('D0','E0') = 1;
Lines[3,7]=1 # L('D0','D1') = 1;
Lines[4,12]=1 # L('E0','E2') = 1;
Lines[5,6]=1 # L('B1','C1') = 1;
Lines[5,9]=1 # L('B1','B2') = 1;
Lines[6,7]=1 # L('C1','D1') = 1;
Lines[6,10]=1 # L('C1','C2') = 1;
Lines[7,11]=1 # L('D1','D2') = 1;
#*****************************************************************************

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

@variable(tennis, x[1:P, 1:P], Bin)

@variable(tennis, s[1:P], Bin)
fix(s[5],0; force = true)
fix(s[6],0; force = true)
fix(s[7],0; force = true)
fix(s[9],0; force = true)
fix(s[10],0; force = true)
fix(s[11],0; force = true)

@variable(tennis, e[1:P], Bin)
fix(e[5],0; force = true)
fix(e[6],0; force = true)
fix(e[7],0; force = true)
fix(e[9],0; force = true)
fix(e[10],0; force = true)
fix(e[11],0; force = true)

# Minimize walking-distance
@objective(tennis, Min,
           sum(Distance[p,pp]*x[p,pp] for p=1:P, pp=1:P))

# what goes in must come out
@constraint(tennis, [p=1:P],
            sum(x[p,pp] for pp=1:P) - sum(x[pp,p] for pp=1:P) ==
            s[p] - e[p])

# one start
@constraint(tennis, sum(s[p] for p=1:P) == 1)

# one end
```

```
@constraint(tennis, sum(e[p] for p=1:P) == 1)

# force clean lines
@constraint(tennis, [p=1:P,pp=1:P],
            x[p,pp] + x[pp,p] >= Lines[p,pp])
#*****************************************************************************

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

#*****************************************************************************
# Report results
println("-------------------------------------");
if termination_status(tennis) == MOI.OPTIMAL
    println("RESULTS:")
    println("objective = $(objective_value(tennis))")
else
  println("  No solution")
end
println("-------------------------------------");
#*****************************************************************************
```