Groceries

The Groceries problem illustrates the two most important **routing** problems in Operations Resarch: The Travelling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). Both problems have been studied intensely over the last 50 years and a large number of practical application, like delivering groceries. Both TSP and VRP are also computationally hard optimization problems, even for a small problem sizes. Here we focus on relatively simple models. If larger problems needs to be solved, more advanced algorithmic approaches are needed.

Problem 1

- Minimize the travel distance of the van.

Sets

- Set of Customers, the first being the grocery: $c \in Customers = \{1, 2, 3, 4, 5, 6\}$

Parameters

- $Coord_{c,coord}$: The x and x coordinates of customer c used to calculate the Distance matrix
- $Distance_{c1,c2}$: The (Euclidean) distance between customer $c1$ and $c2$, calculated based on the coordinates.

Decision variables

- Drive van from customer $c1$ to $c2 : x_{c1,c2} \in \{0, 1\}$. Notice that the variable $x_{c,c}$, i.e. driving from node c to node c does not make any sense, so in the Julia/JuMP program we fix it to zero, but not in the model below.

Model

Objective:

- Minimize the driven distance:

$$\sum_{c1,c2} Distance_{c1,c2} \cdot x_{c1,c2}$$

Constraints:

- You have to drive **to** each customer exactly once:

$$\sum_{c} 2x_{c2,c1} = 1 \ \forall c1$$

- You have to drive **away** from each customer exactly once:

$$\sum_{c} 2x_{c1,c2} = 1 \ \forall c1$$

The above model is very simple and fast to solve, but has a serious issue (Problem 2 below).

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

`Groceries1.jl`

from the book web-site, is given below:

```
#*****************************************************************
# Groceries1, "Mathematical Programming Modelling" (42112)
#*****************************************************************
# Intro definitions
using JuMP
using HiGHS
#*****************************************************************

#*****************************************************************
# PARAMETERS
C=6 # no of customers
coord = zeros(C,2)

coord[1,1]=0; coord[1,2]=0
coord[2,1]=104; coord[2,2]=19
```

```
coord[3,1]=370; coord[3,2]=305
coord[4,1]=651; coord[4,2]=221
coord[5,1]=112; coord[5,2]=121
coord[6,1]=134; coord[6,2]=515

Distance = zeros(Float64,C,C)
for c1 in 1:C
    for c2 in 1:C
        Distance[c1,c2]= sqrt( (coord[c1,1]-coord[c2,1])*(coord[c1,1]-coord[c2,1]) + (coord
    end
end
#*****************************************************************************

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

@variable(TSP, x[1:C,1:C],Bin)
for c in 1:C
    fix(x[c,c],0; force = true)
end

# Minimize TSP distance
@objective(TSP, Min,
           sum(Distance[c1,c2]*x[c1,c2] for c1=1:C, c2=1:C))

# you enter all cities
@constraint(TSP, city_enter_con[c1=1:C],
            sum(x[c2,c1] for c2=1:C)==1)

# you exit all cities
@constraint(TSP, city_exit_con[c1=1:C],
            sum(x[c1,c2] for c2=1:C)==1)
#*****************************************************************************

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

#*****************************************************************************
println("objective = $(objective_value(TSP))")
println("Solve time: $(solve_time(TSP))")
println(round.(Int8,value.(x)))
#*****************************************************************************
```

If you look at the solution, it is clear that the solution consists of two separate loops: $c1 \rightarrow c2 \rightarrow c5 \rightarrow c1$ and $c3 \rightarrow c6 \rightarrow c4 \rightarrow c3$. The problem is that these routes actually satisfy the constraints, but constraints are missing which will disallow loops. What is needed is the addition of a new variable and one constraint.

Problem 2

- Minimize the travel distance of the van, now without loops.

Parameters

- C the number of cities

Decision variables

- Counting variable $u_c \geq 0$. If a customer following another customer has to have a counting variable 1 higher, except the first one.

Constraints:

- The constraint which sets the counting variable for each customer:

$$u_{c1} + 1 \leq u_{c2} + C(1 - x_{c1,c2}) \; \forall \; c1, c2 | c1 \neq c2 \wedge c2! = 1$$

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

`Groceries2.jl`

from the book web-site, is given below:

```
#*****************************************************************
# GroceriesTSP2, "Mathematical Programming Modelling" (42112)
#*****************************************************************
# Intro definitions
using JuMP
using HiGHS
#*****************************************************************

#*****************************************************************
# PARAMETERS
C=6 # no of customers
coord = zeros(C,2)
```

```
coord[1,1]=0; coord[1,2]=0
coord[2,1]=104; coord[2,2]=19
coord[3,1]=370; coord[3,2]=305
coord[4,1]=651; coord[4,2]=221
coord[5,1]=112; coord[5,2]=121
coord[6,1]=134; coord[6,2]=515

Distance = zeros(Float64,C,C)
for c1 in 1:C
    for c2 in 1:C
        Distance[c1,c2]= sqrt( (coord[c1,1]-coord[c2,1])*(coord[c1,1]-coord[c2,1]) + (coord
    end
end
#*****************************************************************************

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

@variable(TSP, x[1:C,1:C],Bin)
for c1 in 1:C
    fix(x[c1,c1],0; force = true)
end

@variable(TSP, u[1:C] >= 0)

# Minimize TSP distance
@objective(TSP, Min,
           sum(Distance[c1,c2]*x[c1,c2] for c1=1:C, c2=1:C))

# you enter all cities
@constraint(TSP, city_enter_con[c1=1:C],
            sum(x[c2,c1] for c2=1:C)==1)

# you exit all cities
@constraint(TSP, city_exit_con[c1=1:C],
            sum(x[c1,c2] for c2=1:C)==1)


# counter constraint
@constraint(TSP, counter_con[c1=1:C,c2=2:C,c1!=c2],
            u[c1] + 1 <= u[c2] + C*(1-x[c1,c2])
            )
#*****************************************************************************

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

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

#*************************************************************************
println("objective = $(objective_value(TSP))")
println("Solve time: $(solve_time(TSP))")
println("u: ", round.(Int64,value.(u)))
#*************************************************************************
```

If you now look at the solution, the solution is now one tour visiting all customers: $c1 \rightarrow c2 \rightarrow c4 \rightarrow c3 \rightarrow c6 \rightarrow c5 \rightarrow c1$. But the new tour is more expensive. Notice also the way the extra constraint, both a condition to ensure that $c1 \neq c2$, in julia $c1! = c2$. But the second requirement, that the first customer, the grocery, can reset the u_c counting variable, is simply done by numbering: $c2 = 1 : C$.

When there are too many customers, with too large demands, several van's are necessary. This complicates the problem significantly and creates the Vehicle Routing Problem (VRP). In this version of the problem, K (3) vans needs to be routed to customers such that: All customers get their goods and that the capacity, of 26, is not exceeded. Hence we need three "TSP" like routes, and the simple solution is: K routes leave the grocery, customer 1, and K routes needs to arrive at the customer 1. But all the customers have the same requirements as before. Finally, each route cannot exceed the capacity. Here we utilize the u_c counting variable, to count how much consumer goods the vans transport until this node. So instead of adding 1 unit we add q_c unit. Finally, we change the domain of the u_c variable to be the capacity of vans, hence ensuring that not too many customers are visited by a van.

Problem 3

- Minimize the travel of all the vans

New Parameters

- K: Number of vans, here 3
- Q: Van capacity, here 26

Changed decision variables

- Counting variable $0 \geq u_c \leq Q$. If a customer following another customer has to have a counting variable q_c higher, except the first one.

Model

Objective:

- Minimize the driven distance:

$$\sum_{c1,c2} Distance_{c1,c2} \cdot x_{c1,c2}$$

Constraints:

- You have to drive **to** each customer (except the grocer) exactly once:

$$\sum_{c} 2x_{c2,c1} = 1 \ \forall \ c1 \mid c1 \neq 1$$

- K vans have to drive **away** from the grocer):

$$\sum_{c} 2x_{c1,c2} = 1 \ \forall \ c1 \mid c1 \neq 1$$

- You have to drive **away** from each customer (except the grocer) exactly once:

$$\sum_{c} 2x_{1,c2} = K$$

- K vans have to drive **to** the grocer:

$$\sum_{c} 2x_{c2,1} = K$$

- The constraint which sets the counting variable for each customer:

$$u_{c1} + q_{c1} \leq u_{c2} + C(1 - x_{c1,c2}) \ \forall \ c1, c2 \mid c1 \neq c2 \wedge c2 \neq 1$$

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

`Groceries3.jl`

from the book web-site, is given below:

```
#*************************************************************************
# Vehicle Routing, "Mathematical Programming Modelling" (42112)
using JuMP
using HiGHS
#*************************************************************************

#*************************************************************************
# PARAMETERS
C=13 # no of customers

# customer coordinates
coord = zeros(C,2)
coord[1,1]=0; coord[1,2]=0
coord[2,1]=104; coord[2,2]=19
coord[3,1]=370; coord[3,2]=305
coord[4,1]=651; coord[4,2]=221
coord[5,1]=112; coord[5,2]=121
coord[6,1]=134; coord[6,2]=515
coord[6,1]=134; coord[6,2]=515;
coord[7,1]=797; coord[7,2]=424;
coord[8,1]=347; coord[8,2]=444;
coord[9,1]=756; coord[9,2]=141;
coord[10,1]=304; coord[10,2]=351;
coord[11,1]=236; coord[11,2]=775;
coord[12,1]=687; coord[12,2]=310;
coord[13,1]=452; coord[13,2]=57;

# distance between customers
Distance = zeros(Float64,C,C)
for c1 in 1:C
    for c2 in 1:C
        Distance[c1,c2]= sqrt( (coord[c1,1]-coord[c2,1])*(coord[c1,1]-coord[c2,1]) + (coord
    end
end

# size of deliveries
q = zeros(C)
q[1]=0;
q[2]=3;
q[3]=9;
q[4]=7;
q[5]=11;
q[6]=11;
q[7]=6;
q[8]=7;
q[9]=7;
```

```
q[10]=2;
q[11]=4;
q[12]=2;
q[13]=8;

# number of vans
K=3

# capacity of each of the vans
Q=26
#*****************************************************************************

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

@variable(VRP, x[1:C,1:C],Bin)
for c in 1:C
    fix(x[c,c],0; force = true)
end

@variable(VRP, 0 <= u[1:C] <= Q)

# Minimize VRP distance
@objective(VRP, Min,
           sum(Distance[c1,c2]*x[c1,c2] for c1=1:C, c2=1:C))

# every city is entered by one truck, except the depot on node 1
@constraint(VRP, [c1=2:C],
            sum(x[c2,c1] for c2=1:C) == 1)

# every city is exited by one truck, except the depot on node 1
@constraint(VRP, [c1=2:C],
            sum(x[c1,c2] for c2=1:C) == 1)

# K trucks going out of node 1
@constraint(VRP,
            sum(x[1,c2] for c2=2:C) == K)

# K trucks going in of node 1
@constraint(VRP,
            sum(x[c2,1] for c2=2:C) == K)

# counter constraint
@constraint(VRP, [c1=2:C,c2=1:C,c1!=c2],
            u[c1] + q[c1] <= u[c2] + Q*(1-x[c1,c2]) )
```

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

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

#*****************************************************************************
#print(VRP)
println("objective = $(objective_value(VRP))")
println("Solve time: $(solve_time(VRP))")
#*****************************************************************************
```

If you look at the solution, it is clear that the solution consists of two separate loops: $c1 \rightarrow c2 \rightarrow c5 \rightarrow c1$ and $c3 \rightarrow c6 \rightarrow c4 \rightarrow c3$. The problem is that these routes actually satisfy the constraints, but constraints are missing which will disallow loops. What is needed is the addition of a new variable and one constraint.