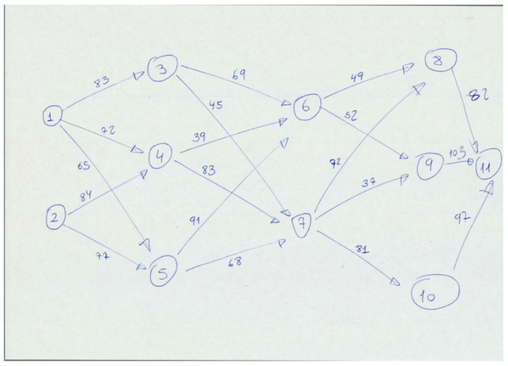
# Problem set # 2

Andres Mejia

10/26/2021

## 1. Humanitarian supply flow.

Consider the network in Figure 1. Node 11 represents a city in a state of emergency, which urgently needs humanitarian supplies. Nodes 1 and 2 are origins of the required supplies. The numbers shown on arcs are their daily transportation capacities, e.g., at most 83 units of supplies can be sent in a day from node 1 to node 3. Consider the problem of sending as many humanitarian supplies a s possible in a day from nodes 1 and 2 to node 11.



### a. Formulate the problem as an integer optimization model and formulate its linear relaxation.

This problem is a network flow problem. Let $x_{i,j}$ be the flow between nodes $i$ and $j$, Let $b_{i,j}$ the max flow between two nodes. Then the problem becomes:

$$\max \sum_{i} x_{j,11} = x_{8,11} + x_{9,11} + x_{10,11}$$

Edge flow constraints:

$$x_{i,j} \leq b_{i,j}$$

$$\begin{array}{cc}
x_{1,3} \leq 83 & x_{1,4} \leq 42 \\
x_{1,5} \leq 65 & x_{2,4} \leq 84 \\
x_{2,5} \leq 77 & x_{3,6} \leq 69 \\
x_{3,7} \leq 45 & x_{4,6} \leq 39 \\
x_{4,7} \leq 83 & x_{5,6} \leq 91 \\
x_{5,7} \leq 68 & x_{6,8} \leq 49 \\
x_{6,9} \leq 52 & x_{7,8} \leq 72 \\
x_{7,9} \leq 37 & x_{7,10} \leq 81 \\
x_{8,11} \leq 82 & x_{9,11} \leq 103 \\
x_{10,11} \leq 97 &
\end{array}$$

Continuity constraints:

$$\sum_j x_{j,i} - \sum_j x_{i,j} = 0$$
$$i \in \{3, \dots, 10\}$$

$$x_{3,6} + x_{3,7} - x_{1,3} - x_{1,3} = 0$$
$$x_{4,6} + x_{4,7} - x_{1,4} - x_{2,4} = 0$$
$$x_{5,6} + x_{5,7} - x_{1,5} - x_{2,5} = 0$$
$$x_{6,8} + x_{6,9} - x_{3,6} - x_{4,6} - x_{5,6} = 0$$
$$x_{7,8} + x7,9 + x_{7,10} - x_{3,7} - x_{4,7} - x_{5,7} = 0$$
$$x_{8,11} - x_{6,8} - x_{7,8} = 0$$
$$x_{9,11} - x_{6,9} - x_{7,9} = 0$$
$$x_{10,11} - x_{7,10} = 0$$

Integer constraints:

$$x_{i,j} \in \mathbb{Z}^+ \cup \{0\}$$

**b. Implement in Gurobi–Python the model and solve the linear relaxation. Do you obtain an integer solution? Why?**

Let's first read a csv that includes the problem information i.e the edges capacity.

```
import pandas as pd

node_info=pd.read_csv("Documents/Input data p1.csv")
node_info
```

```
##      I   j  cap max
## 0    1   3       83
## 1    1   4       42
## 2    1   5       65
## 3    2   4       84
## 4    2   5       77
## 5    3   6       69
## 6    3   7       45
## 7    4   6       39
## 8    4   7       83
## 9    5   6       91
## 10   5   7       68
## 11   6   8       49
## 12   6   9       52
## 13   7   8       72
## 14   7   9       37
## 15   7  10       81
## 16   8  11       82
## 17   9  11      103
## 18  10  11       97
```

Now let's build the problem first we build the relaxed problem. The only diference is that we don't restrict the variables to be integers.

```
import gurobipy as gp

Flow_model=gp.Model("Flow_model")
```

```
arcs=[(node_info.iloc[k,0],node_info.iloc[k,1]) for k in range(19)  ]
limites={(node_info.iloc[k,0],node_info.iloc[k,1]):
  node_info.iloc[k,2] for k in range(19)}
```

```
arcs_var=Flow_model.addVars(arcs,vtype=gp.GRB.CONTINUOUS,name="Flow")

Flow_model.setObjective(gp.quicksum(arcs_var[i]for i in arcs if
i[1]==11),gp.GRB.MAXIMIZE)

capacidad_constr=Flow_model.addConstrs((arcs_var[k]<=limites[k] for k in
arcs),name="Capacidad")

Flow_model.addConstrs((gp.quicksum(arcs_var[k] for k in arcs if
k[1]==j )-gp.quicksum(arcs_var[k] for k in arcs if k[0]==j )==0 for j in
range(3,11)),"Flujo")
```

```
Flow_model.update()
Flow_model.optimize()
```

```
## Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (linux64)
## Thread count: 8 physical cores, 16 logical processors, using up to 16 threads
## Optimize a model with 27 rows, 19 columns and 49 nonzeros
## Model fingerprint: 0xfd49ea59
## Coefficient statistics:
##   Matrix range     [1e+00, 1e+00]
##   Objective range  [1e+00, 1e+00]
##   Bounds range     [0e+00, 0e+00]
##   RHS range        [4e+01, 1e+02]
## Presolve removed 22 rows and 8 columns
## Presolve time: 0.00s
## Presolved: 5 rows, 11 columns, 17 nonzeros
##
## Iteration    Objective       Primal Inf.    Dual Inf.      Time
##        0    2.5200000e+02   2.525000e+01   0.000000e+00      0s
##        6    2.5200000e+02   0.000000e+00   0.000000e+00      0s
##
## Solved in 6 iterations and 0.00 seconds
## Optimal objective  2.520000000e+02
```

The optimal solution is 252. this is integer valued due to the Ford Fulkerson Algorithm that guarantees the existance of a solution built using sums and subtractions of edges capacities. This guarantees that there is at least one solution that is integer valued and that the flow through the edges is also integer valued.

**c. Generate through Gurobi–Python sensitivity information and interpret it.**

```
for v in Flow_model.getVars():
    if v.X != 0:
        print("%s %s %8.2f %s %8.2f %s %8.2f %s %8.2f" %
              (v.Varname, "=", v.X, ", reduced cost = ", v.RC,
              ", from coeff = ", v.SAObjLow, "to coeff = ", v.SAObjUp))
```

```
## Flow[1,3] =    68.00 , reduced cost =      0.00 , from coeff =      0.00 to coeff =      0.00
## Flow[1,4] =    38.00 , reduced cost =      0.00 , from coeff =     -0.00 to coeff =      0.00
## Flow[1,5] =    62.00 , reduced cost =      0.00 , from coeff =      0.00 to coeff =     -0.00
## Flow[2,4] =    84.00 , reduced cost =      0.00 , from coeff =     -0.00 to coeff =       inf
## Flow[3,6] =    23.00 , reduced cost =      0.00 , from coeff =      0.00 to coeff =      0.00
## Flow[3,7] =    45.00 , reduced cost =      0.00 , from coeff =     -0.00 to coeff =       inf
## Flow[4,6] =    39.00 , reduced cost =      0.00 , from coeff =     -0.00 to coeff =       inf
## Flow[4,7] =    83.00 , reduced cost =      0.00 , from coeff =     -0.00 to coeff =       inf
## Flow[5,7] =    62.00 , reduced cost =      0.00 , from coeff =     -0.00 to coeff =      0.00
## Flow[6,8] =    10.00 , reduced cost =      0.00 , from coeff =     -1.00 to coeff =      0.00
## Flow[6,9] =    52.00 , reduced cost =      0.00 , from coeff =     -1.00 to coeff =       inf
## Flow[7,8] =    72.00 , reduced cost =      0.00 , from coeff =     -0.00 to coeff =       inf
## Flow[7,9] =    37.00 , reduced cost =      0.00 , from coeff =     -1.00 to coeff =       inf
## Flow[7,10] =    81.00 , reduced cost =      0.00 , from coeff =     -1.00 to coeff =       inf
## Flow[8,11] =    82.00 , reduced cost =      0.00 , from coeff =     -0.00 to coeff =       inf
## Flow[9,11] =    89.00 , reduced cost =      0.00 , from coeff =     -0.00 to coeff =       inf
## Flow[10,11] =    81.00 , reduced cost =      0.00 , from coeff =     -0.00 to coeff =       inf
```

The reduced cost is zero in all cases, this is not surprising as there is a positive flow in all edges.

```python
for c in Flow_model.getConstrs():
        print("%s %s %8.2f %s %8.2f %s %8.2f" % (c.ConstrName,
        ": shadow price = ", c.Pi, ", from RHS = ", c.SARHSLow, "to RHS = ",
        c.SARHSUp))
```

```
## Capacidad[1,3] : shadow price =      0.00 , from RHS =     68.00 to RHS =       inf
## Capacidad[1,4] : shadow price =      0.00 , from RHS =     38.00 to RHS =       inf
## Capacidad[1,5] : shadow price =      0.00 , from RHS =     62.00 to RHS =       inf
## Capacidad[2,4] : shadow price =      0.00 , from RHS =     80.00 to RHS =    122.00
## Capacidad[2,5] : shadow price =      0.00 , from RHS =      0.00 to RHS =       inf
## Capacidad[3,6] : shadow price =      0.00 , from RHS =     23.00 to RHS =       inf
## Capacidad[3,7] : shadow price =      0.00 , from RHS =     42.00 to RHS =     60.00
## Capacidad[4,6] : shadow price =      0.00 , from RHS =     24.00 to RHS =     43.00
## Capacidad[4,7] : shadow price =      0.00 , from RHS =     80.00 to RHS =     87.00
## Capacidad[5,6] : shadow price =      0.00 , from RHS =      0.00 to RHS =       inf
## Capacidad[5,7] : shadow price =      0.00 , from RHS =     62.00 to RHS =       inf
## Capacidad[6,8] : shadow price =      0.00 , from RHS =     10.00 to RHS =       inf
## Capacidad[6,9] : shadow price =      1.00 , from RHS =     29.00 to RHS =     66.00
## Capacidad[7,8] : shadow price =      0.00 , from RHS =     57.00 to RHS =     75.00
## Capacidad[7,9] : shadow price =      1.00 , from RHS =      0.00 to RHS =     40.00
## Capacidad[7,10] : shadow price =      1.00 , from RHS =     19.00 to RHS =     84.00
## Capacidad[8,11] : shadow price =      1.00 , from RHS =     72.00 to RHS =     97.00
## Capacidad[9,11] : shadow price =      0.00 , from RHS =     89.00 to RHS =       inf
## Capacidad[10,11] : shadow price =      0.00 , from RHS =     81.00 to RHS =       inf
## Flujo[3] : shadow price =      0.00 , from RHS =    -68.00 to RHS =     15.00
## Flujo[4] : shadow price =      0.00 , from RHS =    -38.00 to RHS =      4.00
## Flujo[5] : shadow price =      0.00 , from RHS =    -62.00 to RHS =      3.00
## Flujo[6] : shadow price =      0.00 , from RHS =    -23.00 to RHS =     15.00
## Flujo[7] : shadow price =      0.00 , from RHS =    -62.00 to RHS =      3.00
## Flujo[8] : shadow price =      0.00 , from RHS =    -10.00 to RHS =     15.00
## Flujo[9] : shadow price =     -1.00 , from RHS =    -14.00 to RHS =     89.00
## Flujo[10] : shadow price =     -1.00 , from RHS =    -16.00 to RHS =     81.00
```

On the other hand the shadow prices indicate where we can find a bottleneck in the network. In places where it is valued one, increasing the restriction value will increase the optimum.

**d. If you had a limited budget to increase the capacity of some arcs, which ones would you prioritize? Why?**

The sensitivity information (shadow prices) is on the output above. The shadow prices imply that increasing the capacity at edges $(6,9),(7,9),(7,10)$ and $(8,11)$ will increase the output. In all cases increasing one unit of max flow in that edge will increase one unit of max output.

We also see how much we can increase this in each edge by checking the deltas of the sensitivity analysis. Considering this the edge $(8,11)$ can increase up to 15 units and is the one that can grow the most.

## 2. A person wants to visit by car the following Spanish cities, starting and ending in Madrid: Alicante, Barcelona, Cordoba, La Coruna, Valencia and Granada.

**a. Formulate the problem of finding the shortest tour. How many constraints has in this case the integer optimization formulation seen in class?**

Let $C$ be set of cities we will be visiting and $d_{ij}$ the distance between city i and city j, and $x_{ij}$ be a binary desition variable valued 1 if we travel between city i and j, 0 otherwise. Then the problem becomes:

$$\min \sum_{i \in C, j \in C. i \neq j} d_{ij} x_{ij}$$

subject to:

Entering constraint (each city must have an incoming edge one constraint by city a total of 7 constraints).

$$\sum_{j \in C, j \neq i} x_{ij} = 1 \; i \in C$$

Exiting constraint (each city must have an incoming edge one constraint by city a total of 7 constraints).

$$\sum_{i \in C, j \neq i} x_{ij} = 1 \; j \in C$$

We also have the following constraint to assure no subtours. For all $W \in \mathcal{P}(C)$

$$\sum_{i \in W, j \in W^c} x_{ij} \geq 1$$

This gives us $2^7 = 128$ constraints, however we will only add only those contraint to break sub-tours, when adding this contraint for a set we will also include the contraint for the complementary set.

This is turn brings the total contraints to 187 (including the fact that each variable (42) must be binary)

## b. Apply the iterative procedure seen in class to try to find an optimal tour, carrying out at most three iterations using Gurobi–Python. Discuss the results.

Implementing this in python, let's first read a csv that includes the problem information i.e distances.

```
import pandas as pd

distances=pd.read_csv("Distances.csv", index_col=0)
distances
```

```
##            Madrid  Barcelona  Valencia  Alicante  Granada  Cordoba  La Coruna
## Madrid          0        624       357       421      420      394        597
## Barcelona     624          0       351       538      888      862       1088
## Valencia      357        351         0       165      546      521        950
## Alicante      421        538       165         0      351      552       1017
## Granada       420        888       546       351        0      208       1009
## Cordoba       394        862       524       552      208        0        986
## La Coruna     597       1088       950      1017     1009      986          0
```

The following code creates the model.

```
import gurobipy as gp

TSP_model=gp.Model("TSP")

cities={j for j in distances.columns}

arcs=[(j,k) for j in cities for k in cities  if j!=k]
distance_dict={(j,k): distances.loc[j,k] for j in cities for k in cities}
arcs_var=TSP_model.addVars(arcs,vtype=gp.GRB.BINARY)

TSP_model.setObjective(gp.quicksum(arcs_var[i]*distance_dict[i] for i in arcs ),gp.GRB.MINIMIZE)

TSP_model.addConstrs((gp.quicksum(arcs_var[(i,j)] for i in cities if i!=j)==1 for j in cities ),name="E
```

```
TSP_model.addConstrs((gp.quicksum(arcs_var[(i,j)] for j in cities if j!=i)==1 for i in cities),name="Sa
```

```


TSP_model.update()
TSP_model.optimize()
```

```
## Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (linux64)
## Thread count: 8 physical cores, 16 logical processors, using up to 16 threads
## Optimize a model with 14 rows, 42 columns and 84 nonzeros
## Model fingerprint: 0xf7a0e87d
## Variable types: 0 continuous, 42 integer (42 binary)
## Coefficient statistics:
##   Matrix range     [1e+00, 1e+00]
##   Objective range  [2e+02, 1e+03]
##   Bounds range     [1e+00, 1e+00]
##   RHS range        [1e+00, 1e+00]
## Found heuristic solution: objective 4225.0000000
## Presolve time: 0.00s
## Presolved: 14 rows, 42 columns, 84 nonzeros
## Variable types: 0 continuous, 42 integer (42 binary)
##
## Root relaxation: objective 2.664000e+03, 12 iterations, 0.00 seconds
##
##     Nodes    |    Current Node    |     Objective Bounds      |     Work
##  Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
##
## *    0     0               0    2664.0000000 2664.00000  0.00%     -    0s
##
## Explored 0 nodes (12 simplex iterations) in 0.00 seconds
## Thread count was 16 (of 16 available processors)
```

```
##
## Solution count 2: 2664 4225
##
## Optimal solution found (tolerance 1.00e-04)
## Best objective 2.664000000000e+03, best bound 2.664000000000e+03, gap 0.0000%
```

These functions are defined to manipulate sub-tours and add restrictions to break them.

```
def find_next_city(actual_city):
  for j in cities:
    if j==actual_city:
      continue
    if arcs_var[(actual_city,j)].X-1>-0.1:
      return(j)


def find_tour(city1):
  cities2=set({})
  j=city1
  for i in range(100):
    cities2.add(j)
    j=find_next_city(j)
    if set({j}).issubset(cities2):
      break
  return(cities2)


def add_tour_restriction(tour):
  anti_tour=cities.difference(tour)
  TSP_model.addConstr((gp.quicksum(arcs_var[(i,j)] for i in tour for j in anti_tour)>=1),name="tour_out
  TSP_model.addConstr((gp.quicksum(arcs_var[(i,j)] for j in tour for i in anti_tour)>=1),name="tour_in"
  return


def find_tour_2(city1):
  cities2=set({})
  cities3=list({})
  j=city1
  for i in range(100):
    cities2.add(j)
    cities3.append(j)
    j=find_next_city(j)
    if set({j}).issubset(cities2):
      break
  return(cities3)
```

Using these functions we can find and break the subtours in current solution:

```
find_tour_2("Madrid")
```

```
## ['Madrid', 'La Coruna']
```

```
find_tour_2("Barcelona")
```

```
## ['Barcelona', 'Alicante', 'Valencia']
```

```
find_tour_2("Granada")
```

```
## ['Granada', 'Cordoba']
```

```
add_tour_restriction(find_tour("Madrid"))
add_tour_restriction(find_tour("Barcelona"))
add_tour_restriction(find_tour("Granada"))
```

With these extra restrictions we can solve the problem also we can check that the final problem has only one tour.

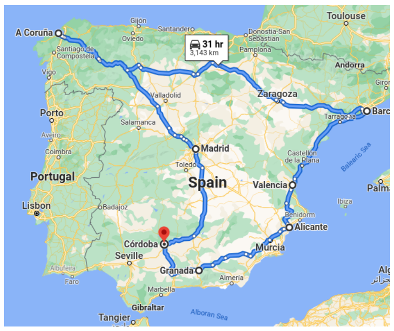
```
TSP_model.update()
TSP_model.optimize()
```

```
## Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (linux64)
## Thread count: 8 physical cores, 16 logical processors, using up to 16 threads
## Optimize a model with 20 rows, 42 columns and 148 nonzeros
## Model fingerprint: 0x64792675
## Variable types: 0 continuous, 42 integer (42 binary)
## Coefficient statistics:
##   Matrix range     [1e+00, 1e+00]
##   Objective range  [2e+02, 1e+03]
##   Bounds range     [1e+00, 1e+00]
##   RHS range        [1e+00, 1e+00]
##
## MIP start from previous solve did not produce a new incumbent solution
## MIP start from previous solve violates constraint tour_out by 1.000000000
##
## Found heuristic solution: objective 4349.0000000
## Presolve time: 0.00s
## Presolved: 20 rows, 42 columns, 148 nonzeros
## Variable types: 0 continuous, 42 integer (42 binary)
##
## Root relaxation: objective 3.154000e+03, 15 iterations, 0.00 seconds
##
##     Nodes    |    Current Node    |     Objective Bounds      |     Work
##  Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
##
## *    0     0               0    3154.0000000 3154.00000  0.00%     -    0s
##
## Explored 0 nodes (15 simplex iterations) in 0.00 seconds
## Thread count was 16 (of 16 available processors)
##
## Solution count 2: 3154 4349
##
## Optimal solution found (tolerance 1.00e-04)
## Best objective 3.154000000000e+03, best bound 3.154000000000e+03, gap 0.0000%
```

```
find_tour_2("Madrid")
```

```
## ['Madrid', 'Cordoba', 'Granada', 'Alicante', 'Valencia', 'Barcelona', 'La Coruna']
```

This is the optimal tour that is 3154 km long, note that the reverse is also an optimal tour. The tour is shown in the following plot.


31 hr
3,143 km
A Coruña
Gijón
Santander
Oviedo
Donostia-San Sebastian
Toulouse
Santiago de Compostela
Pamplona
Andorra
Vigo
Zaragoza
Valladolid
Porto
Salamanca
Tarragona
Aveiro
Madrid
Coimbra
Castellón de la Plana
Toledo
Portugal
Spain
Valencia
Balearic Sea
Lisbon
Badajoz
Ibiza
Benidorm
Alicante
Córdoba
Seville
Murcia
Granada
Albufeira
Faro
Almería
Marbella
Gibraltar
Alboran Sea
Tangier