IO - Pràctica 2 - Informe

AUTORS: AARÓN ACOSTA, JÚLIA GASULL

Enunciat de la pràctica

Tal i com es mostra al pdf penjat al racó, per aquesta pràctica se'ns demana implementar, a partir d'un graf dirigit amb costos, un problema de vehicle routing amb constriccions addicionals MTZ. Abans, però s'ha de modificar la matriu d'aquests costos amb l'algoritme Floyd-Warshall.

Anem a veure què diu exactament aquest enunciat:

Cada estudiant tindrà assignat un fitxer gràfic on apareixerà l'esquema d'una ciutat representada per artèries i nusos principals, formant un graf dirigit.

D'entre els nusos caldrà destacar els marcats com A i B. Els estudiants amb DNI parell adoptaran com depot per a problema de VR a resoldre el nus A i el B com una localització més on distribuir mercaderia, mentre que els estudiants amb DNI senar adoptaran com depot el nus B i el A com una destinació.

En el fitxer gràfic s'especifica quin és el cost de viatge d'un (i,j) mitjançant una fórmula basada en les coordenades xc, yc dels nusos i, j.

El fitxers .mod proporcionats presenten codis AMPL d'exemple que utilitzen el fitxer .dat que defineix el graf de la ciutat i també que resolen un problema TSP en n nusos mitjançant la implementació de la formulació MTZ.

Utilitzeu una capacitat de transport dels vehicles de C = 40 unitats.

Es demana:

(1) Implementeu l'algoritme de Floyd-Warshall per a calcular els costs de viatge mínims entre les diferents destinacions (incloent-t'hi el depot)

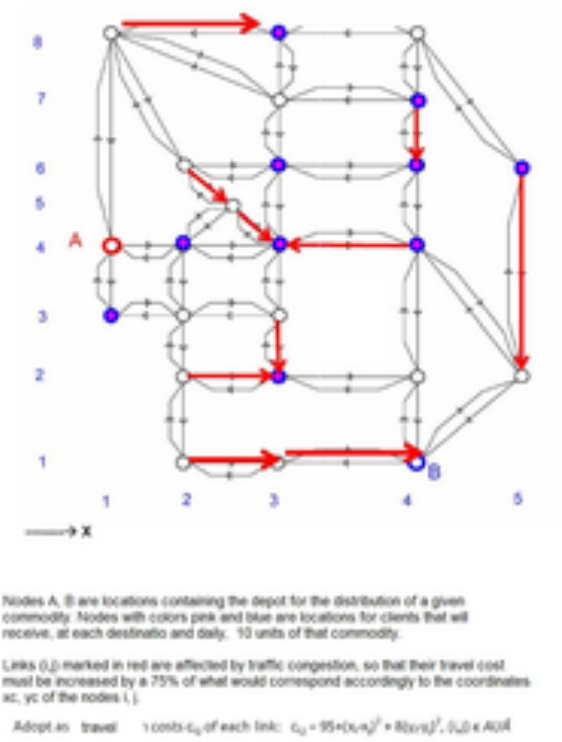
(2) Implementeu la formulació MTZ del Vehicle Routing Problem i reporteu la seva solució per a K = 3, 4, 5 vehicles com grandària de flota.

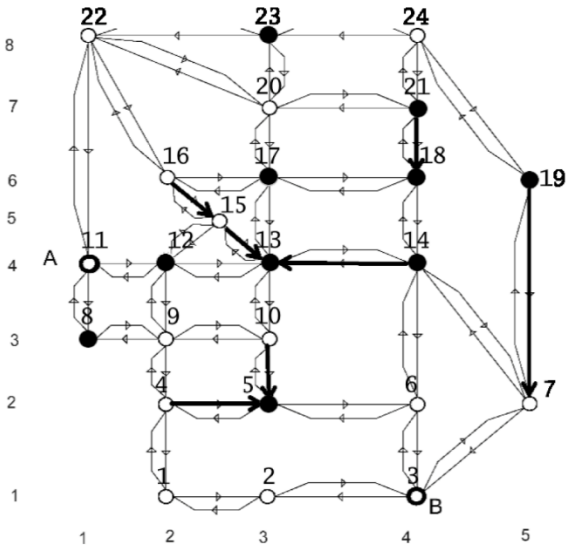
Formulació MTZ del VRP.
Se suposa un graf dirigit $G = (N \cup \{0\}, A), N = \{1, 2, 3, ..., n\}$

$$
\begin{array}{lll}
Min_{x,\,u} & \sum_{(i,j)\in A} c_{i,j}x_{i,j} & \\
& \sum_{i\in N} x_{i,j} = 1, & j \in N \\
& \sum_{j\in N} x_{i,j} = 1, & i \in N \\
& \sum_{j\in N} x_{0,j} = K, & \\
& \sum_{j\in N} x_{i,0} = K, & \\
& u_j \geq u_i + a_j - C(1 - x_{i,j}) & i \neq 0,\ j \neq 0 \\
& a_i \leq u_i \leq C & i \in N \\
& x_{i,j} \in \{0, 1\} &
\end{array}
$$

Implementació de la pràctica

Per començar, dels dos arxius que teníem per fer la pràctica, hem triat aquest, tenint com a depot el node B, és a dir, el node 3.





Floyd-Warshall

Per començar, i segons es va comentar a classe, l'única part que necessitem d'aquest algoritme és la que a les transparències s'anomena `matriu D`, que ens dona els costos mínims d'un graf complet amb els mateixos nodes que el graf original.

Anem a veure com és aquest algoritme:

ALGORISME de CAMINS MÍNIMS TOTS AMB TOTS (Floyd-Warshall)

Inicialització:

$P(i,j) = i,\ D(i,j) = c_{i,j}$ si $(i,j) \in A$
$D(i,j) = 0$ si $i = j$
$P(i,j) = 0\ \ D(i,j) = \infty$ si $(i,j) \notin A$

Per $\ell = 1, ..., n$
  Per $i = 1, ..., n$
    Per $j = 1, ..., n$
      Si $D(i,j) > D(i,\ell) + D(\ell,j)$
        $D(i,j) = D(i,\ell) + D(\ell,j)$
        $P(i,j) = P(\ell,j)$
Fi Per

- Al finalitzar $P(i,*)$ és el vector de predecessors de l'arbre originat a $i \in N$.
- Si hi han cicles negatius apareixerà algún $d_{ii}^{\ell} < 0$, essent i el nus de numeració més alta dins del cicle i d_{ii}^{ℓ} el cost del cicle.

La manera en la que nosaltres ho hem implementat ha estat la següent:

- Donat que necessitem una matriu de costos, i que el que se'ns ha assignat a nosaltres és el que segueix la fòrmula: $c_{i,j}= 95 + (x_i-x_j)^2 + 8((y_i-y_j)^2$, primer hem creat una matriu cost on,
    - per tots els arcs existents al nostre graf inicial, el cost es calcula seguint la fòrmula anterior
    - per cada parell de nodes equivalents, és a dir, la diagonal de la matriu, el cost equival a 0
    - per cada arc no existent al graf inicial, el cost equival a infinit
- Un cop fet això, només calia aplicar l'algoritme de la següent manera al fitxer `netdes.run`:

```
for {i in NODES} {
  let cost[i,i] := 0;
}

for {(i,j) in ARCS} {
  let cost[i,j] := 95 + (xcoord[i]-xcoord[j])**2 + 8*((ycoord[i]-ycoord[j])**2);
}

for {l in 1 .. card(NODES)} {
  for {i in NODES, j in NODES} {
    if cost[i,j] > cost[i,l] + cost[l,j] then {
      let cost[i,j] := cost[i,l] + cost[l,j];
    }
  }
}
```

VRP amb MTZ

Per tal d'implementar VRP amb MTZ, només hem hagut de seguir la formulació exposada a la pràctica, de manera que es complissin totes les restriccions per als nodes als que considerem clients.

Les constriccions, doncs, les hem implementat de la següent manera:

```
# Funció objectiu
minimize total_cost:
 sum {i in CLIENTS, j in CLIENTS} cost[i,j] * x[i,j];

# Degree constraints
subject to Degree_Out {i in CLIENTS: i!=3}:

  sum {j in CLIENTS: j!=i } x[i,j] = 1;
subject to Degree_In  {j in CLIENTS: j!=3}:

  sum {i in CLIENTS: i!=j } x[i,j] = 1;

# Arrivals to depot
subject to Depot_arrivals {j in CLIENTS: j=3}:

  sum {i in CLIENTS: i!=3 } x[i,j]=K;

# Departures from depot
subject to Depot_departures {i in CLIENTS: i=3}:
 sum {j in CLIENTS: j!=3} x[i,j]=K;

# Approximate subtour elimination constraints
subject to subtour_elimination {
  (i,j) in (CLIENTS diff {3}) cross (CLIENTS diff {3})
}:
  u[j] >= u[i] + demand[j] - C*(1 - x[i,j]);
```

```
# Lowerbound on u
subject to lowerbound {i in CLIENTS diff {3}}: u[i]>=demand[i];

# Upperbound on u
subject to upperbound {i in CLIENTS diff {3}}: u[i]<=C;
```

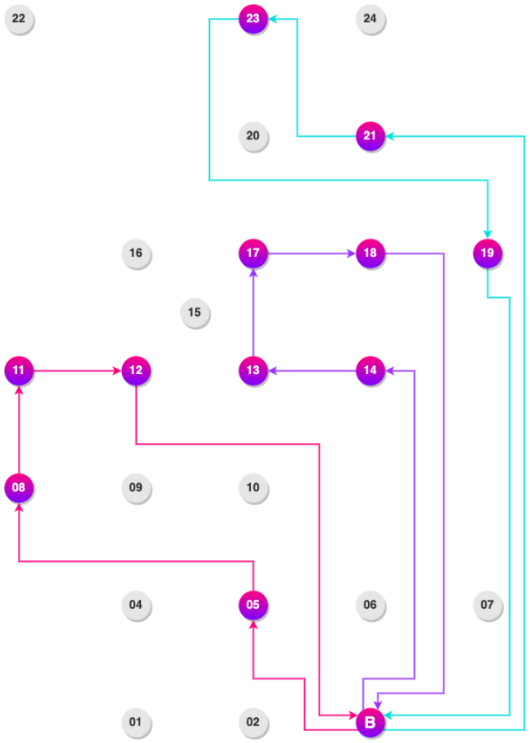
Pel que fa el fitxer netdes.dat, hem declarat els següents camps:

- NODES → set dels 24 nodes que tenim al graf
- CLIENTS → set dels clients als que es vol arribar (3 5 8 11 12 13 14 17 18 19 21 23)
- ORIGIN → el node B (3)
- ARCS → arcs presents al graf inicial
- xcoord, ycoord → coordenades de cada node
- demand → per a cada client, la demanda (en positiu) i, per al depot, la oferta (en negatiu)
- C → capacitat màxima dels camions (40)
- K → nombre de camions que tenim (3, 4 i 5)

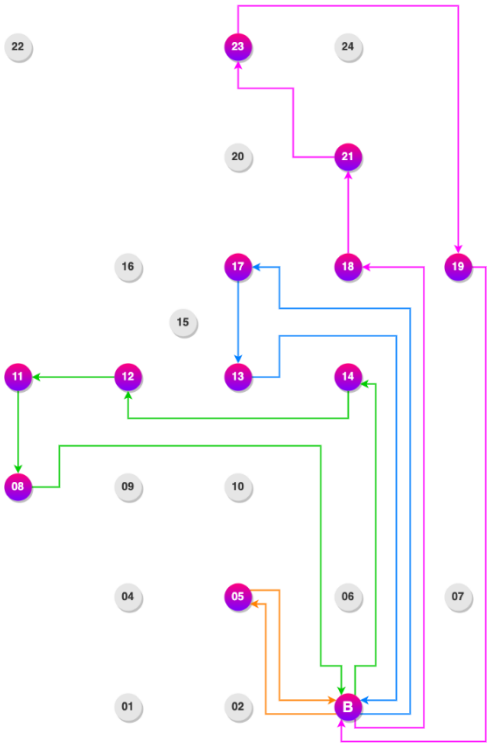
Resultats

Per a K=3

```
3 --> 5
3 --> 14
3 --> 21
5 --> 8
8 --> 11
11 --> 12
12 --> 3
13 --> 17
14 --> 13
17 --> 18
18 --> 3
19 --> 3
21 --> 23
23 --> 19
```



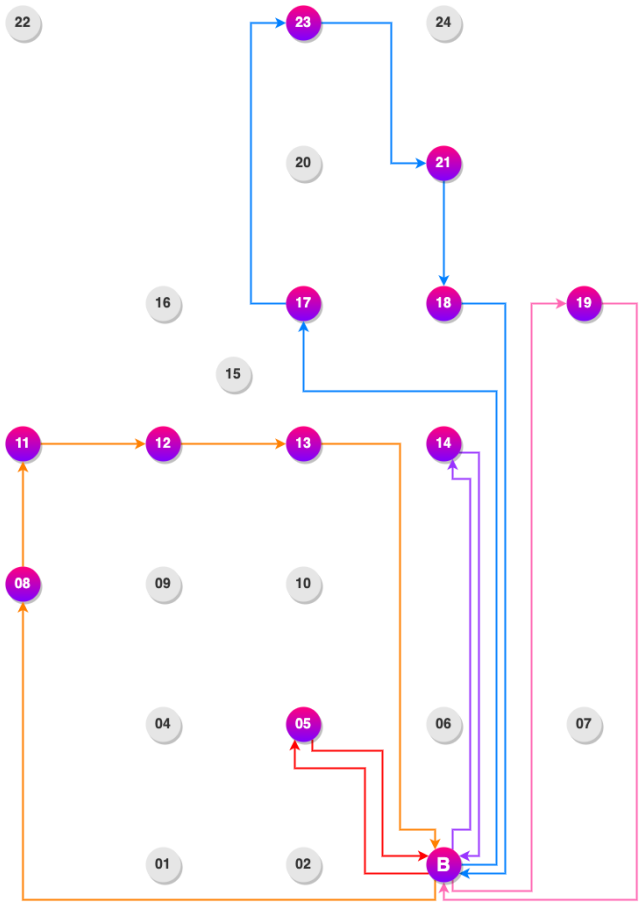
Per a K=4




```
 3 --> 5
 3 --> 14
 3 --> 17
 3 --> 18
 5 --> 3
 8 --> 3
11 --> 8
12 --> 11
13 --> 3
14 --> 12
17 --> 13
18 --> 21
19 --> 3
21 --> 23
23 --> 19
```

Per a K=5

```
3 --> 5
3 --> 8
3 --> 14
3 --> 17
3 --> 19
5 --> 3
8 --> 11
11 --> 12
12 --> 13
13 --> 3
14 --> 3
17 --> 23
18 --> 3
19 --> 3
21 --> 18
23 --> 21
```



Codi

netdes.run

```
reset;

model netdes.mod;
data netdes.dat;

# Initialization of cost
for {i in NODES} {
  let cost[i,i] := 0;
}

for {(i,j) in ARCS} {
  let cost[i,j] := 95 + (xcoord[i]-xcoord[j])**2 + 8*((ycoord[i]-ycoord[j])**2);
}


display cost;

for {l in 1 .. card(NODES)} {
  for {i in NODES, j in NODES} {
    if cost[i,j] > cost[i,l] + cost[l,j] then {
      let cost[i,j] := cost[i,l] + cost[l,j];
    }
  }
}

display cost;

option solver './gurobi';
solve;

for {(i,j) in CLIENTS cross CLIENTS}
  if x[i,j] >= 1
  then {printf "%i --> %i\n", i, j};
```

netdes.mod

```
set NODES;
set CLIENTS within NODES;
set ORIGIN within NODES;
set ARCS within NODES cross NODES;

/*---------------------*/
/*---Floyd-Warshall---*/
/*---------------------*/

param cost {NODES,NODES} default Infinity;
param xcoord {NODES};
param ycoord {NODES};

/*---------------------*/
/*----VRP with MTZ----*/
/*---------------------*/

param C > 0; # capacidad del camion
param K > 0; # numero de camiones
param demand {CLIENTS}; # demanda de los clientes

var x {CLIENTS,CLIENTS} binary;
var u {CLIENTS} >= 0;

minimize total_cost:
  sum {i in CLIENTS, j in CLIENTS} cost[i,j] * x[i,j];

# Degree constraints
subject to Degree_Out {i in CLIENTS: i!=3}:
  sum {j in CLIENTS: j!=i } x[i,j] = 1;
subject to Degree_In {j in CLIENTS: j!=3}:
  sum {i in CLIENTS: i!=j } x[i,j] = 1;

# Arrivals to depot
subject to Depot_arrivals {j in CLIENTS: j=3}:
  sum {i in CLIENTS: i!=3 } x[i,j]=K;

# Departures from depot
subject to Depot_departures {i in CLIENTS: i=3}:
  sum {j in CLIENTS: j!=3} x[i,j]=K;

# Approximate subtour elimination constraints
subject to subtour_elimination
  {(i,j) in (CLIENTS diff {3}) cross (CLIENTS diff {3})}:
    u[j] >= u[i] + demand[j] - C*(1 - x[i,j]);

# Lowerbound on u
subject to lowerbound {i in CLIENTS diff {3}}: u[i]>=demand[i];

# Upperbound on u
subject to upperbound {i in CLIENTS diff {3}}: u[i]<=C;
```

netdes.dat

```
#set of nodos
set NODES := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24;
set CLIENTS := 3 5 8 11 12 13 14 17 18 19 21 23;
#set of  origins
set ORIGIN := 3;

#fixed links
set ARCS :=
  (1, 2) (1, 4)
  (2, 1) (2, 3)
  (3, 2) (3, 6) (3, 7)
  (4, 1) (4, 5) (4, 9)
  (5, 4) (5, 6) (5, 10)
  (6, 3) (6, 5) (6, 14)
  (7, 3) (7, 14) (7, 19)
  (8, 9) (8, 11)
  (9, 4) (9, 8) (9, 10) (9, 12)
  (10, 5) (10, 9) (10, 13)
  (11, 8) (11, 12) (11, 22)
  (12, 9) (12, 11) (12, 13) (12, 15)
  (13, 10) (13, 12) (13, 14) (13, 15) (13, 17)
  (14, 6) (14, 7) (14, 13) (14, 18)
  (15, 12) (15, 13) (15, 16)
  (16, 15) (16, 17) (16, 22)
  (17, 13) (17, 16) (17, 18) (17, 20)
  (18, 14) (18, 17) (18, 21)
  (19, 7) (19, 24)
  (20, 17) (20, 21) (20, 22) (20, 23)
  (21, 18) (21, 20) (21, 24)
  (22, 11) (22, 16) (22, 20) (22, 23)
  (23, 20) (23, 24)
  (24, 19) (24, 21)
;

# xc: x coordinates of nodes
param xcoord:=
        1 2
        2 3
        3 4
        4 2
        5 3
        6 4
        7 5
        8 1
        9 2
        10 3
        11 1
        12 2
        13 3
        14 4
        15 2.5
        16 2
        17 3
        18 4
        19 5
        20 3
        21 4
        22 1
        23 3
        24 4
;
```

```
# yc: y coordinates of nodes
param ycoord:=
  1 1
        2 1
        3 1
        4 2
        5 2
        6 2
        7 2
        8 3
        9 3
        10 3
        11 4
        12 4
        13 4
        14 4
        15 5
        16 6
        17 6
        18 6
        19 6
        20 7
        21 7
        22 8
        23 8
        24 8
;

#right hand side at nodes
param demand :=
  3      -110 /* Valor modificable */
  5      10
  8      10
  11     10
  12     10
  13     10
  14     10
  17     10
  18     10
  19     10
  21     10
  23     10
;

param C := 40;
param K := 5; # 3, 4 o 5
```