Exercici 2
Q1 2022-23

Integrants:
Miquel Muñoz García-Ramos
Cristian Sánchez Estapé

Índex

1. Apartat 1

1.1 Comentaris previs a la implementació

En primer lloc, la implementació feta comprèn dos arxius: *Floyd-Warshall.dat*, que seria un fitxer que emmagatzema les dades de:

- **N**, el conjunt de nodes del graf amb què treballem
- **links**, el conjunt d'arcs que tenim al graf
- **cost**, el cost que té cada arc de la xarxa. Aquest paràmetre, a diferència dels anteriors, ha estat generat mitjançant el script de python *generateCosts.py*.

D'altra banda, el fitxer *Floyd-Warshall.mod*, que seria el fitxer que genera la taula mateixa. Ara bé, per tal que es faci correctament se li ha d'introduir el *path* absolut fins al fitxer *Floyd-Warshall.dat* (vegeu la negreta de la següent subsecció).

En segon lloc, el fitxer *.mod* mostra per consola la taula generada. Tot i això, la visualització que es fa per defecte correspon a les dades d'entrada del problema del MTZ. És per aquest motiu que, si es vol, el codi inclou també la visualització de la taula amb la matriu de predecessors inclosa (també indicada, en negreta, a la següent subsecció).

En tercer i darrer lloc, cal esmentar que l'execució del codi d'aquest apartat només requereix la càrrega del fitxer *.mod* (és el mateix script qui carrega les dades).

1.2 Implementació del problema a AMPL

Aquí adjuntem el codi (comentat) que s'ha implementat per la generació de la taula de *Floyd-Warshall*:

```
# nodes
set N;
# arcs
set links within {N, N};
# costs
param cost {links} default 0;

# carreguem les dades del problema
#data /path/fins/el/fitxer/Floyd-Warshall.dat
data  /home/cristian/Escriptori/Uni/IO/Ex2/P2/ap1.dat;

# Matriu de distancies
param D {N,N} default Infinity;
# Matriu de predecessors
param P {N,N} default 0;

for {(i,j) in links} {
  let P[i,j] := i;
  let D[i,j] := cost[i,j];
}
for {i in N} {
      let D[i,i] := 0;
}
for {l in N}
      for {i in N}
            for {j in N}
                if i != j and D[i,j] > D[i,l] + D[l,j] then {
                  let D[i,j] := D[i,l] + D[l,j];
                  let P[i,j] := P[l,j];
                   }

# Mostrem les matrius
printf "\nMatriu de distancies mínimes:\n";
#printf "Origen\tDestí\tDistancia\tPredecessor\n";
printf "Origen\tDestí\tDistancia\n";
for {i in  N, j in N}
      #printf "%d\t%d\t%s\t\t%d\n",i,j,round(D[i,j],2),P[i,j];
      printf "%d\t%d\t%s\n",i,j,round(D[i,j],2);

# Com que els costs amb què treballem sempre seran positius, ignorem els
potencials cicles de cost negatiu.
```

Figura 1: codi corresponent a la implementació de la generació de la taula Floyd-Warshall

1.3 Resultats de les execucions

Posat que les taules resultats tenen al voltant de 580 línies de text, adjuntem dos fitxers, *OutputNormal.txt* i *OutputAmbPredecessors.txt*, en els quals trobem les dues visualitzacions possibles de la taula de Floyd-Warshall. Aquí deixem, però, una mostra de com es veuen els resultats:

Matriu de distàncies mínimes:

Origen	Destí	Distancia
1	1	0
1	2	96
1	3	192
1	4	168
1	5	336
1	6	360
1	7	289
1	8	432
1	9	336
1	10	432

Figura 2: taula Floyd-Warshall sense predecessors (fitxer OutputNormal.txt)

Matriu de distàncies mínimes:

Origen	Destí	Distancia	Predecessor
1	1	0	0
1	2	96	1
1	3	192	2
1	4	168	1
1	5	336	4
1	6	360	3
1	7	289	3
1	8	432	9
1	9	336	4
1	10	432	5

Figura 3: taula Floyd-Warshall amb predecessors (fitxer OutputAmbPredecessors.txt)

2. Apartat 2

2.1 Comentaris previs a la implementació

En primer lloc, posat que ambdós membres del grup tenim DNIs que acaben en nombres tant senar com parell, hem triat el nus A com a *depot* i el B com a destinació amb demanda.

En segon lloc, esmentem que el codi, de nou, es divideix en dos fitxers: *MTZ.dat*, que correspon a la definició de les dades d'entrada del problema, on:

- **N** és el conjunt sencer de nodes del problema
- **O** és el paràmetre que defineix el node d'origen (en el nostre cas és l'11)
- **K** és el paràmetre que defineix el nombre de camions amb què treballem
- **C** és la capacitat màxima d'un camió
- **a** és la demanda de cada node/ciutat
- **D** és el cost de viatjar des d'una ciutat a una altra; aquest és el paràmetre que correspon a la taula de Floyd-Warshall, generada a l'apartat anterior

D'altra banda, trobem *MTZ.mod*, que correspon al codi que implementa les restriccions del problema.

En tercer lloc, cal esmentar que el model fa ús de variables binàries i, per tal que AMPL resolgui correctament el problema amb el nostre codi, hem de garantir que el *solver* emprat tingui en consideració la naturalesa de les variables. És per això que demanem que, per exemple, s'utilitzi el *solver* **CPLEX**, i s'evitin d'altres com **MINOS**, que no respecten les variables binàries.

2.2 Implementació del problema a AMPL

A continuació, s'adjunta el codi (comentat) que implementa la formulació MTZ del *Vehicle Routing Problem* (*VRP*):

```
# nodes (inclos el node origen O)
set N;

# costs segons distancia (taula generada per Floyd-Warshall)
param D{(i,j) in N cross N};
# capacitat dels camions
param C;
# nombre de camions
param K;
# node origen
param O;
# demanda individual de cada node
param a {N};

# variable binaria x
var x {(i,j) in N cross N} binary, default 0;
# variable de capacitat
var u {N} <= C;

subject to Degree_Out {j in N: j!= O}:
       sum {i in N: i != j} x[i,j] = 1;

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

subject to Depot_departures:
       sum {j in N: j != O} x[O,j] = K;

subject to Depot_arrivals:
       sum {i in N: i != O} x[i,O] = K;

subject to subtour_elimination {(i,j) in (N diff {O}) cross (N diff {O}): i !=
j}:
       u[j] >= u[i] + a[j] - C * (1 - x[i,j]);

subject to Capacity_Constraint {i in (N diff {O})}:
       a[i] <= u[i] <= C;

minimize total_cost:
       sum {i in N, j in N} D[i,j] * x[i,j];
```

Figura 4: codi corresponent a la implementació de la formulació MTZ del VRP

2.3 Resultats de les execucions

A continuació, i partint del fet que la variable x actua com una matriu d'adjacències, mostrem el resultat de les execucions per una $K = \{3, 4, 5\}$, juntament amb el graf associat a la informació continguda a x, que ens mostrarà quins han estat els trajectes realitzats per cada circuit.

Finalment, recollint l'output que la comanda ***display x;*** produeix a AMPL en un fitxer anomenat *TaulesK.txt*, depurant cadascuna de les matrius per deixar la informació referent a la matriu d'adjacències (és a dir, sense els *headers* de les files i les columnes) en fitxers de la forma *Taula*$\{3, 4, 5\}$*.txt*, i mitjançant el script de R *GraphGeneration.R*[1], s'han generat els graphs que presentem a continuació. També tinguis en compte que els nodes marcats en daurat corresponen als clients, és a dir, aquells que tenen demanda > 0.

```
CPLEX 20.1.0.0: optimal integer solution within mipgap or absmipgap; objective 3014.75
180886 MIP simplex iterations
14818 branch-and-bound nodes
absmipgap = 0.3, relmipgap = 9.95107e-05
```

Figura 5: resultat de l'execució del MTZ per $K = 3$

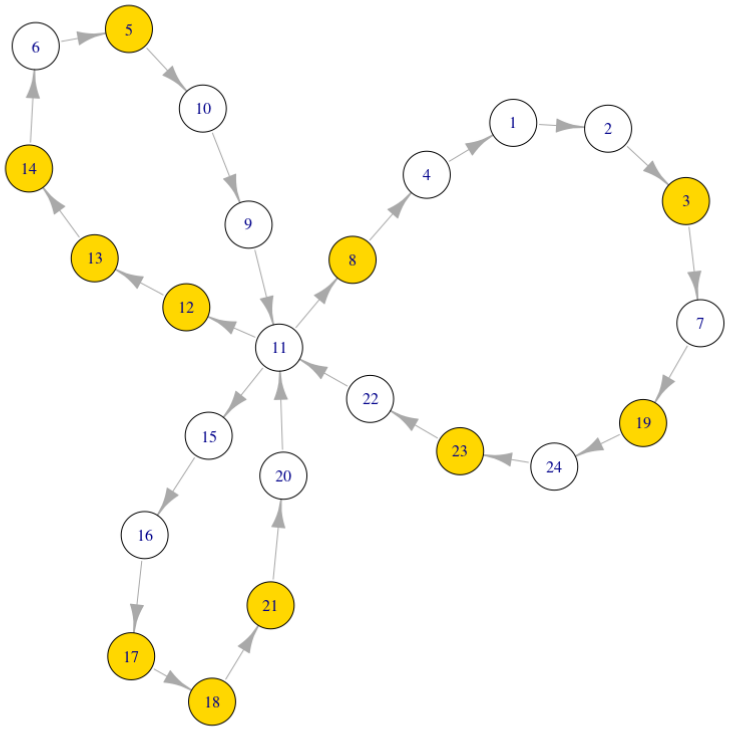

Figura 6: graf dels subcircuits òptims per $K = 3$

[1] Tingui en compte que, per tal que funcioni correctament, la 3a línia de codi del script requereix que se li introdueixi el *path* absolut fins on es troben els fitxers *.txt*.

```
CPLEX 20.1.0.0: optimal integer solution within mipgap or absmipgap; objective 3190.75
31661 MIP simplex iterations
2717 branch-and-bound nodes
absmipgap = 0.0746429, relmipgap = 2.33935e-05
```

Figura 7: resultat de l'execució del MTZ per $K = 4$

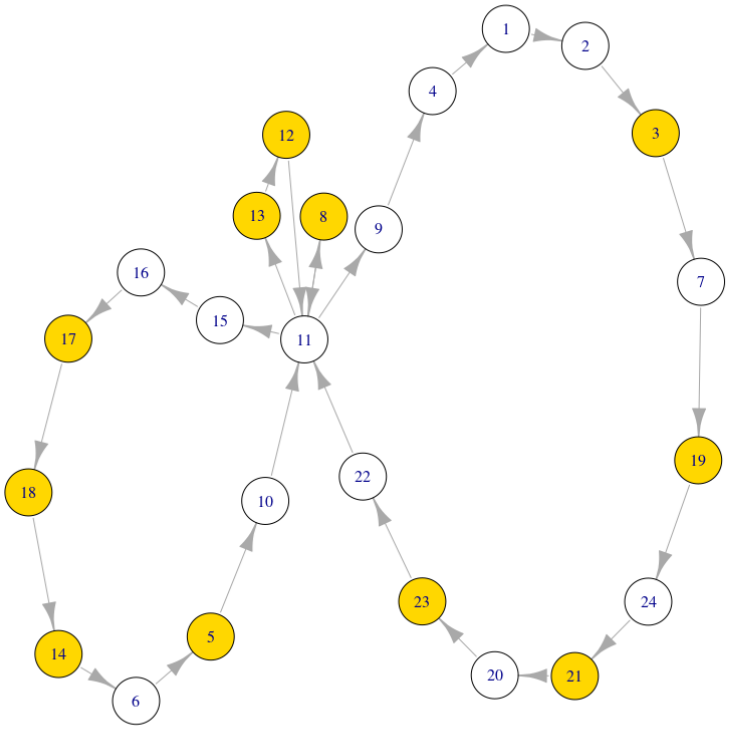

Figura 8: graf dels subcircuits òptims per $K = 4$

```
CPLEX 20.1.0.0: optimal integer solution within mipgap or absmipgap; objective 3382.75
15641 MIP simplex iterations
1670 branch-and-bound nodes
absmipgap = 0.25, relmipgap = 7.39044e-05
```

Figura 9: resultat de l'execució del MTZ per $K = 5$

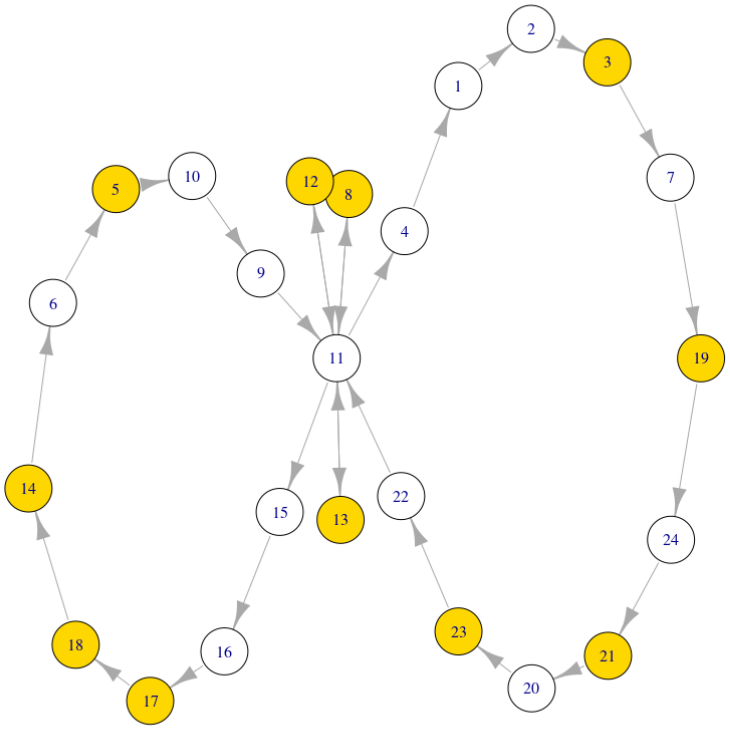

Figura 10: graf dels subcircuits òptims per $K = 5$

Tal i com es pot apreciar, a priori podem validar la correctesa de la solució, ja que en tots els casos i per tots els subcircuits es tenen, com a màxim, 4 clients, a més a més del fet que tenim tants subcircuits com camions disponibles. En definitiva, podem afirmar que la nostra implementació aconsegueix donar una solució factible al problema plantejat.