Master M2 TET, ENPC 2024-25
Recherche Opérationnelle

Projet : TSP et programmation linéaire

1. Formulation DFJ du TSP symétrique

On considère le problème du voyageur de commerce (TSP) *symétrique* c'est-à-dire avec des distances entre villes qui sont les mêmes quelque soit le sens du parcours. Dans ce cas, on peut considérer un graphe non-orienté associé au TSP. Les sommets représentent les villes et les arêtes *non-orientées* correspondent à un parcours emprunté entre deux villes dans la tournée. Les variables de décision sont associées directement aux arêtes du graphe (parcours entre les villes) et non plus aux villes reliées entre elles (avec le sens de parcours). Cette formulation par programmation linéaire (PL) est due à Dantzig, Fulkerson, Johnson [1].

On notera V l'ensemble des sommets du graphe (les villes) et $A = \{e = \{i, j\};\ i, j \in V \text{ avec } i \neq j\}$ l'ensemble des arêtes non-orientées. On considère ainsi le graphe *non-orienté* $G = (V, A)$ associé au TSP symétrique. On introduit alors les variables

$$x_e = \begin{cases} 1 & \text{si l'arête } e \text{ est parcourue} \\ 0 & \text{sinon} \end{cases}$$

et on note $m = |A|$ le nombre d'arêtes, $n = |V|$ le nombre de villes. Si le graphe est complet (les sommets sont tous reliés entre eux), on a $m = n(n-1)/2$. Pour simplifier on note $V = \{1, \cdots, n\}$ et $A = \{1, \cdots, m\}$. Les distances étant symétriques, on dispose des distances $(d_e)_{1 \leq e \leq m}$ associées aux arêtes du graphe. La formulation DFJ s'écrit

$$\min \sum_{e=1}^{m} d_e x_e \tag{1}$$

$$\sum_{e \in \delta(i)} x_e = 2,\ \forall i \in [\![1, n]\!] \tag{2}$$

$$\sum_{e \in \delta(W)} x_e \geq 2,\ \forall W \subsetneq V,\ |W| \geq 2 \tag{3}$$

$$x_e \in \{0, 1\},\ \forall e \in [\![1, m]\!], \tag{4}$$

où $\delta(i)$ désigne l'ensemble des arêtes ayant i comme sommet et $\delta(W)$ l'ensemble des arêtes allant d'un sommet de $W \subsetneq V$ à un sommet de $V \setminus W$ (cf. Figure 1 ci-dessous). On note $|W|$ le nombre de sommets du sous-ensemble W.

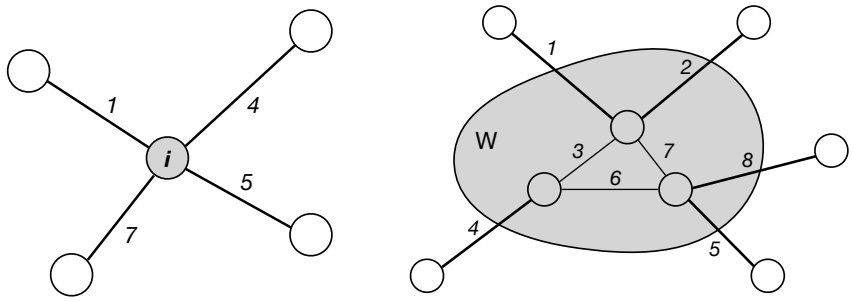


Figure 1 – Ensembles d'arêtes $\delta(i) = \{1, 4, 5, 7\}$ (à gauche) ; $\delta(W) = \{1, 2, 4, 5, 8\}$ et $A_W = \{3, 6, 7\}$ (à droite).

1. Dantzig, G.B., Fulkerson D.R. and Johnson S.M. *Solutions of a large scale travelling salesman problem*, Ops. Res., 2, 393-410 (1954).

Sans la contrainte (3), des *sous-circuits* peuvent apparaître (chemin non-connexe avec plusieurs composantes connexes dont les deux sommets extrémités sont identiques). La contrainte (3) permet précisément de supprimer les sous-circuits. Dans l'exemple de la Figure 2, il y a 2 sous-circuits $W_1 = \{1, 2, 4\}$ et $W_2 = \{3, 5, 6, 7\}$. On a $\delta(W_1) = \emptyset$ car il n'y a aucune arête reliant les sommets de W_1 aux autres sommets restants de W_2. On a donc $x_e = 0$ pour toute arête $e \in \delta(W_1)$, la contrainte (3) n'est donc pas respectée dans cet exemple.

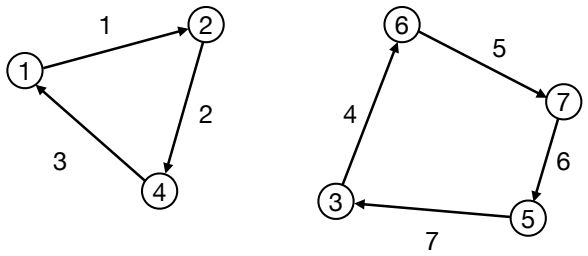


FIGURE 2 – Un exemple de 2 sous-circuits.

La contrainte (3) peut s'exprimer de façon équivalente sous une forme plus simple à utiliser dans la pratique.

Proposition 1 *Soit $W \subset V$. On note A_W l'ensemble de toutes les arêtes contenues dans W i.e. $A_W = \{e \in A,\ e = \{i, j\}$ avec $i, j \in W\}$. Sous la contrainte* (2)*, la contrainte* (3) *est équivalente à :*

$$\sum_{e \in A_W} x_e \leq |W| - 1, \quad \forall W \subsetneq V,\ |W| \geq 2 \tag{5}$$

Dans l'exemple de la Figure 2, on a $A_{W_1} = \{1, 2, 3\}$ et $\sum_{e \in A_{W_1}} x_e = x_1 + x_2 + x_3 = 3$. La contrainte (5) n'est donc pas respectée car $|W| - 1 = 2$.

2. Méthode de résolution

Il y a un nombre exponentiel de contraintes [2]. Dans la pratique, on commence par résoudre (1),(2),(4) sans la contrainte (3) (ou (5)). Quand on rencontre un sous-circuit, on ajoute les contraintes (5) sur chacune des composantes connexes (ayant au moins 2 sommets) du graphe correspondant. On résout alors ce nouveau problème et on recommence de façon itérative, jusqu'à ne plus rencontrer de sous-circuit.

Ce n'est pas exactement la contrainte (5) qui est utilisée mais une variante où la somme porte uniquement sur les arêtes constituant le sous-circuit et non pas sur toutes les arêtes de la composante connexe. A l'itération k, supposons que l'on rencontre un sous-circuit $W \subsetneq V$ constitué d'un ensemble d'arêtes A_k composant le sous-chemin, c'est-à-dire avec

$$A_k = \{e \in W,\ x_e = 1\}, \tag{6}$$

où $\{x_e\}_{1 \leq e \leq m}$ est la solution obtenue (présentant un sous-circuit). On considère alors la contrainte suivante à la place de (5) :

$$\sum_{e \in A_k} x_e \leq |W| - 1. \tag{7}$$

L'algorithme correspondant est donné ci-dessous.

2. il y a exactement $2^n - (n + 2)$ contraintes pour (5) et n contraintes pour (2), soit un total de $2^n - 2$ contraintes.

Algorithme 1 : Résolution de la formulation DFJ avec élimination itérative des sous-circuits

• *Initialisation.* Résoudre

$$(P_0)\begin{cases} \min\limits_{x_e\in\{0,1\}} \sum\limits_{e=1}^{m} d_e x_e \\ \sum\limits_{e\in\delta(i)} x_e = 2,\ \forall i \in [\![1,n]\!] \end{cases}$$

$k \leftarrow 1$

while il existe un sous-circuit **do**

Soit n_{cc} le nombre de composantes connexes ($n_{cc} > 1$)

for $i = 1 : n_{cc}$ **do**

• Ajouter au problème (P_{k-1}) la contrainte pour le sous-circuit $A_{k,i}$

$\sum\limits_{e\in A_{k,i}} x_e \leq N_{k,i} - 1$ où $N_{k,i}$ est le nombre de sommets dans la composante connexe i.

end for

$\rightarrow$ on obtient le problème (P_k)

• Résoudre (P_k)

$k \leftarrow k + 1$

end while

3. Forme matricielle de la formulation

On note les variables $\mathbf{x} = (x_1, \cdots, x_m)^\top \in \{0,1\}^m$ et le vecteur des distances $\mathbf{d} = (d_1, \cdots, d_m)^\top$. A l'itération k, en présence d'un sous-circuit A_k, le problème (P_k) à résoudre s'écrit matriciellement

$$\min_{\mathbf{x}} \mathbf{d}^\top \mathbf{x} \tag{8}$$

$$A\mathbf{x} = \mathbf{2} \tag{9}$$

$$B\mathbf{x} \leq N_k - 1 \tag{10}$$

$$\mathbf{x} \in \{0,1\}^m \tag{11}$$

— La matrice $A = (a_{ij})_{\substack{1\leq i\leq n \\ 1\leq j\leq m}}$ est la matrice d'*incidence* sommets/arêtes du graphe G,

$$a_{ij} = \begin{cases} 1 & \text{si le sommet } i \text{ est une extrémité de l'arête } j, \\ 0 & \text{sinon.} \end{cases} \tag{12}$$

— La matrice B est de taille (n_{cc}, m) où n_{cc} est le nombre de composantes connexes du graphe associé à la solution $\mathbf{x}$ précédente du problème (P_{k-1}). Le graphe associé à une solution $\mathbf{x}$ d'un PL est défini de la façon suivante : une arête e du graphe G associé à la solution $\mathbf{x} \in \{0,1\}^m$ d'un PL existe dans G si et seulement si $x_e = 1$.

Chaque ligne de B correspond donc à une composante connexe (sous-circuit) du graphe associé G. La ligne associée à un sous-circuit A_k est le vecteur $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_m)^\top$ donné par

$$\alpha_e = \begin{cases} 1 & \text{si l'arête } e \in A_k, \\ 0 & \text{sinon.} \end{cases} \tag{13}$$

4. Module `networkx` de `python` pour les graphes

Le module `networkx` permet de créer et de gérer des graphes sous `python`.

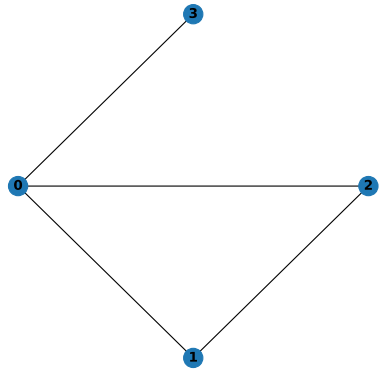
• Construction d'un graphe.

Les instructions suivantes permettent de créer un graphe à partir d'arêtes. Les arêtes sont définies par une liste ou un tableau `numpy` contenant les numéros des 2 sommets extrémités.

```
import networkx as nx
```

```
# Initialisation du graphe
G = nx.Graph()
# Définition des arêtes avec 4 sommets (numérotés de 0 à 3)
id_edges=[[0,1],[0,2],[0,3],[1,2]]
# Construction du graphe
G.add_edges_from(id_edges)

# Affichage du graphe
pos = nx.shell_layout(G)
nx.draw(G, pos=pos, with_labels=True, font_weight='bold')
```



— Pour construire un graphe complet comportant n sommets (numérotés de 0 à $n-1$), vous pourrez utiliser la fonction `edges` dans le fichier `edges.py` qui vous est fourni [3]. Cette fonction construit la liste des arêtes suivantes (un tableau `numpy` en fait) :

`id_edges = [[0,1], ... ,[0,n-1], ..., [i,j], ... [n-2,n-1]]` avec $i < j$.

— Pour construire un graphe associé à la solution `x` d'un PL où l'arête e existe si et seulement si $x_e = 1$, pensez à utiliser la fonction `where` du module `numpy`.

```
import numpy as np

# Initialisation du graphe
G = nx.Graph()
# Construction du graphe associé à la solution xsol (un vecteur/liste)
esol=np.where(np.array(xsol)==1)[0]
G.add_edges_from(idxs[esol,:])
```

Enfin, une fois le graphe construit, on a accès aux arêtes d'un graphe `G` avec l'instruction `G.edges` (et `G.nodes` pour les numéros des sommets) :

```
for e in G.edges:
   print(e)
```

• Composantes connexes.

La fonction `number_connected_components` de `networkx` retourne le nombre de composantes connexes d'un graphe `G` : `ncc=nx.number_connected_components(G)`

Pour accéder aux composantes connexes d'un graphe vous pouvez utiliser la fonction `connected_components` de `networkx`, en récupérant les sous-graphes avec l'instruction suivante :

```
Gcc = [G.subgraph(c).copy() for c in nx.connected_components(G)]
```

• Matrices.

La fonction `incidence_matrix` de `networkx` fournit la matrice d'incidence d'un graphe : `A=nx.incidence_matrix(G)`. La matrice A a une structure creuse c'est-à-dire que seuls les éléments

3. Pour construire un graphe complet vous pouvez aussi utiliser l'instruction `G = nx.complete_graph(n)`.

non-nuls de A sont stockés. L'instruction `A.todense()` permet de convertir la matrice A en un format "plein" (pour pouvoir être affichée avec un `print` par exemple).

Pour construire la matrice B dans la contrainte (10), vous pouvez aussi utiliser une matrice creuse pour B avec le module `scipy` :

```
import scipy as sp

B = sp.lil_array((ncc, m), dtype=int)
```

Pour construire les lignes de B avec (13), vous pourrez utiliser la fonction `nodes_edges` qui vous est fournie dans le fichier `edges.py`. L'instruction `E=nodes_edges(id_edges)` retourne un tableau `numpy` qui permet de retrouver l'arête e associée aux sommets (i, j) avec `E[i,j] = e`

5. Solveur PL.

Le solveur de PL à utiliser est Gurobi avec son interface Python, le module `gurobipy`. Il y a deux façons d'installer Gurobi/`gurobipy`.

— Se rendre sur le site de Gurobi (*Gurobi for Academics*) et demander une licence académique (vous devez fournir une adresse académique d'une université ou d'une école et être connecté.e - via éventuellement un vpn - à cette institution). Vous recevrez un fichier de licence et il vous faudra suivre les instructions pour installer Gurobi.

`https://www.gurobi.com/features/academic-named-user-license/`

— Utiliser une version "bridée" (au plus 2000 contraintes et 200 variables) avec une installation directe sans demande de licence. Exécutez la commande [4]

`pip install gurobipy`

dans un environnement de type `Visual Studio Code`, `Anaconda` ou directement en ligne de commande dans une console système.

Travail demandé.

1. *Partie théorique (subsidiaire).* L'objectif de cette question est d'établir la Proposition 1 en montrant la contrainte (5).
   (a) Montrer que
   $$2 \sum_{k \in A_W} x_k = 2|W| - \sum_{k \in \delta(W)} x_k \tag{14}$$
   Indication. Scinder en deux la somme apparaissant dans la contrainte (3), avec des arêtes $k \in A_W$ et des arêtes $k \notin A_W$. Sommer ensuite sur les sommets $i \in W$.
   (b) En déduire (5) en utilisant (3).
2. Deux fichiers sont à récupérer à l'adresse

   `https://scheid.perso.math.cnrs.fr/master-m2-tet-enpc.html`

   Il y a un fichier utilitaire `edges.py` à ne pas modifier et un squelette de script `tspsym.py` qu'il faut compléter en construisant la matrice B puis résoudre le PL correspondant à chaque itération.

   — Tester votre code avec `n=9` et les distances

   `dist=[2, 5, 3, 3, 1, 2, 1, 5, 1, 4, 3, 1, 1, 2, 2, 1, 4, 1, 4, 1, 5, 5, 1, 2, 4, 5, 4, 3, 2, 2, 3, 2, 5, 3, 5, 5]`.

   Vous devez obtenir une convergence en 2 itérations avec une distance minimale $d_{min} = 13$.

4. PIP est un gestionnaire de paquets standard pour Python. Il s'agit d'un outil de ligne de commande qui vous aide à installer et désinstaller des packages. Pensez à installer `pip` si vous êtes sous windows

— Tester sur d'autres données en engendrant aléatoirement des vecteurs distances et en augmentant progressivement le nombre n de sommets.

Le fichier de script complété `tspsym.py` est à retourner par mail
`jean-francois.scheid@univ-lorraine.fr`
avec un compte rendu succinct présentant les résultats obtenus.