INFO-F103 – Algorithmique 1
Projet 1 : *Facility location problem*

Jérôme De Boeck

Année académique 2020-2021

Facility Location Problem

Le problème de l'emplacement d'installations, ou *Facility Location Problem* (FLP), est un problème d'optimisation qui consiste à déterminer quelles emplacements choisir pour ouvrir des centres d'approvisionnement à moindre coût, en considérant des coûts de construction ainsi que des coûts de déplacement des clients jusqu'aux centres ouverts. Ce problème peut par exemple s'appliquer dans le cas du choix de centres de vaccination à ouvrir pour vacciner un ensemble de famille dans le cas de la campagne de vaccination en cours.
Un ensemble de sites J est disponible pour la construction de centres de vaccination. La construction d'un centre $j \in J$ a un coût fixe f_j. Chaque famille $i \in I$ doit être affecté à un centre $j \in J$. Un coût de trajet t_{ij} est à considérer dans le cas où la famille i est affectée au centre j. Nous considérons que chaque famille peut se déplacer jusqu'à chaque centre si celui-ci est construit. Le coût total d'un choix de sites pour la construction de centre de vaccination dépend donc des coûts fixes de construction des centres ainsi que des coûts de déplacement des familles.
Chaque famille à un demande d_i de doses de vaccin qui doit être satisfaite par le centre auquel elle est affectée et chaque site de construction disposera d'une capacité maximale de doses c_j (qui peut être considérée comme égale à 0 si aucun centre n'est construit dessus). La demande totale des familles affectés à un centre ne peut dépasser sa capacité. Les familles ne peuvent donc pas nécessairement être toutes affectées au centre ouvert le plus proche.
Si les sites choisis pour la construction de centre de vaccination sont notés $J' \subset J$ et les affectations sont représentées par une liste a où a_i est le site auquel la famille i est affectée, le coût total se calcul par

$$\sum_{j \in J'} f_j + \sum_{i \in I} t_{ia_i}.$$

Le FLP consiste donc à déterminer les ensembles J' et la liste a minimisant cette expression.
Pour prendre un exemple concret considérons les données suivantes pour 7 familles $I = \{0, ..., 6\}$ et 4 sites de construction $J = \{0, ..., 3\}$:

- Coûts fixes f_j : 17, 12, 10, 9
- Capacité c_j : 45, 30, 25, 20
- Demandes d_i : 8, 13, 8, 9, 11, 9, 7

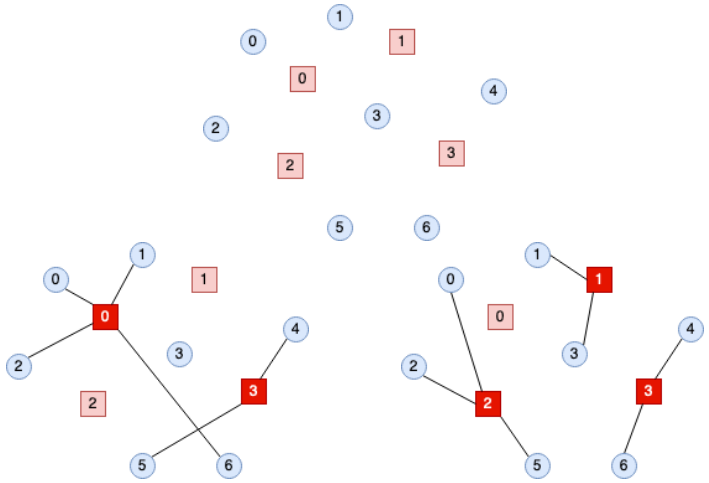

FIGURE 1 – default

— Coût de trajet :

t_{ij}	0	1	2	3
0	1	3	3	6
1	2	1	3	5
2	2	4	1	6
3	2	1	2	2
4	5	3	6	2
5	4	6	2	4
6	5	6	4	2

La Figure 1 illustre cet exemple. Les familles I sont représentées par des cercles et les sites J où des centres de vaccination peuvent potentiellement être construits sont représentés par des carrés dans la figure du haut. Un exemple de solution où deux centres sont ouverts avec les affectations correspondantes est donné en bas à gauche avec $J' = \{0,3\}$ et $a = [0,0,0,0,3,3,0]$ et un coût de 44 (17 + 9 pour les coûts fixes et 1+2+2+2+2+4+5 pour les coûts de déplacement de chaque famille), cette solution est optimale si l'on force l'utilisation des sites 0 et 3. Une autre solution en bas à droite avec trois centres, $J' = \{1,2,3\}$ et $a = [2,1,2,1,3,2,3]$ à un coût de 43 et est optimale.

Dans toutes les notations qui suivent, les ensembles de familles et de sites I et J sont numérotés à partir de 0 comme dans l'exemple ci-dessus pour qu'une famille ou un site puisse correspondre à un indice donné d'une liste dans l'implémentation.

Implémentations

Les deux fonctions suivantes sont à implémenter :

— `def minimum_travel_cost(demand,capacity,travel_cost): return (cost,assignments)` Cette fonction a pour but de trouver une affectation des fa-

milles aux centres de vaccination minimisant les coûts de déplacements en considérant que ceux-ci sont déjà construits. Cette fonction ne tient compte d'aucun coût de construction. Le paramètre `travel_cost` est un tableau tel que `travel_cost[i][j]` = t_{ij}, `demand` et `capacity` sont des listes tels que `demand[i]` = d_i et `capacity[j]` = c_j pour $i \in I, j \in J$. Dans le cas où un centre n'est pas construit sur un site j, alors `capacity[j]`=0. La valeur de retour est un tuple où `cost` est le coût minimum et `assigments` est un liste où `assigments[i]` est le site auquel est assigné la famille j (la liste a dans la section précédente). Cette affectation doit minimiser les coûts de transport de telle sorte que la demande de chaque famille soit satisfaite et la capacité de chaque site respectée. Si aucune affectation ne permet de satisfaire les demandes, le tuple (-1,[]) est retourné.

Sur l'exemple précédent dans la solution de gauche, le paramètre `capacity` serait égal à [45,0,0,20] et la valeur de retour de cette fonction serait (18,[0,0,0,0,3,3,0]).

— `def facility_location(opening_cost,demand,capacity,travel_cost): return (cost,locations,assignments)` Cette fonction résout le FLP. Les paramètres `travel_cost`, `demand` et `capacity` sont identique à la fonction ci-dessus, `opening_cost` est une liste où `opening_cost[j]` = f_j. Un tuple contenant dans l'ordre le coût de la solution optimale du FLP `cost`, les sites ouverts `locations` contenu dans une liste (les ensemble J' de l'exemple) et les affectations `assignments` tel que défini dans la fonction précédentes est à retourner. Une liste peut également être retournée contenant ces trois éléments dans le même ordre mais ne retournez pas trois variables séparées. Si le problème ne possède pas de solution, le tuple (-1,[],[]) est retourné.

Sur l'exemple précédent la valeur de retour de cette fonction serait (43,[1,2,3],[2,1,2,1,3,2,3]).

Il est évidemment fortement encouragé d'implémenter d'autres fonctions. Les deux fonctions demandées ci-dessus peuvent ne contenir que des appels à d'autres fonctions permettant de retourner les valeurs demandées.

Votre code doit avant tout trouver une solution optimale. Des éléments permettant d'améliorer le temps de résolution du problème sont attendus mais pas au détriment de l'optimalité de la solution.

Quelques consignes pour l'ensemble de votre code :

— Les deux fonctions demandées sont bien des fonctions et non des méthodes de classe. Les noms ainsi que la fonction de chaque attribut doivent être respectés tel que décrite. Aucun attribut supplémentaire n'est autorisé. Celles-ci doivent pouvoir être appelées à partir d'autres fichiers importants votre fichier.

— Aucun code ne doit être exécuté lors de l'import de votre fichier. Si vous écrivez du code en plus de vos fonctions pour le tester, placez-le sous la condition `if __name__ == "__main__":`

— L'utilisation des variables globales est autorisée à condition que leur nom commence par `glob_` pour ne pas causer de problème lors de l'import de votre fichier et que les deux fonctions demandées puissent être directement appelées. N'hésitez pas à ajouter plusieurs attributs aux fonctions que vous créez si besoin, même s'il y en a beaucoup, pour éviter des variables globales inutiles. L'utilisation de classes est déconseillée mais

autorisée.
- L'ensemble de votre code doit être commenté, en particulier chaque fonction doit détailler ses attributs et sa valeur de retour. Vos différents choix d'implémentation pour résoudre le problème d'optimisation doivent également être commenté afin de rendre le code lisible.
- Seul la librairies `sys` et `copy` peuvent être utilisées. Les copies de listes doivent être utilisées seulement si celles-ci s'avèrent pertinentes.
- Les `print` sont autorisés dans le code remise mais seulement s'ils sont sont pertinents pour informer sur l'exécution du code. Évitez à tout prix les suites illisibles de lignes à l'écran pendant la recherche d'une solution dans le code que vous remettez.
- Indiquez votre nom, prénom et numéro de matricule en commentaire au début de votre code.

Consignes générales

Tout votre code doit être contenu dans un seul fichier **projet1.py**. Veuillez respecter le nom, la casse (pas de majuscule), le format des fonctions et le nom du fichier. Un exemple de fichier est donné contenant la fonction lisant un fichier de donné passé en ligne de commande et passe ses données à la fonction `facility_location` afin que vous puissiez tester votre code.
Un répertoire contenant des instances du problème est disponible sur l'UV. Pour tester une instance, placez le fichier correspondant dans le même répertoire que votre fichier projet1.py, par exemple `FLP-7-4.txt` qui correspond à l'exemple, et lancez la commande

```
python3 projet1.py FLP-7-4.txt
```

Vous pouvez tester ceci avant toute implémentation, vous devriez voir les données de l'instance apparaître, effacez ces lignes ensuite. Dans le fichier `FLP-I-J-n.txt`, I correspond au nombre de familles, J au nombre de site où un centre de vaccination peut être construit, et n au numéro de l'instance.
Lors de la correction, des tests sur votre code seront effectués à partir d'un fichier important vos fonctions avec les commandes

```
from projet1 import facility_location
from projet1 import minimum_travel_cost
```

pour importer les deux fonctions demandée de votre fichier, aucun code ne doit être exécuté lors de l'import comme mentionné précédemment. Vos fonctions seront testées sur diverses instances, veillez donc à respecter les noms et attributs.
Un fichier `correction.py` est disponible sur l'UV qui teste votre code sur l'instance `FLP-7-4.txt` et ressemble au script qui sera utilisé lors de la correction. Pour l'utiliser, placez-le dans le même dossier que votre fichier `projet1.py` et l'instance `FLP-7-4.txt` et lancez la commande

```
python3 correction.py
```

Si vous fonctions sont correctement définies et que tous les fichiers sont placés dans le même répertoire, un message indiquera si chacune de vos fonctions a retourné le résultat attendu.

Seront évalués la qualité de votre code, la mise en pratique de la matière vue en cours et les optimisations mises en place. Veillez à ce que votre code soit commenté de manière concise pour mettre en valeur ses fonctionnalités et les optimisations appliquées. Dans le cas où l'exécution de votre code en ligne de commande produit une erreur, une note nulle sera reportée pour la partie exécution, veillez donc à tester votre code via le fichier `correction.py` avant la remise.
Pour remettre votre projet sur l'UV, nous vous demandons de

- créer localement sur votre machine un répertoire de la forme NOM_Prenom (exemple :DUPONT_Jean) dans lequel vous mettez le fichier `projet1.py` à soumettre (sans y inclure de fichier de données)
- compresser ce répertoire via un utilitaire d'archivage produisant un .zip (aucun autre format de compression n'est accepté)
- de soumettre le fichier archive .zip, et uniquement ce fichier, sur l'UV

Le projet est à remettre pour le 14 mars 2021 à 12h sur l'UV. Tout manquement aux consignes ou retard sera sanctionné directement d'un **0/10**.