MOGPL
Résolution de problèmes d'affectation et de localisation

Projet

Antonin ARBERET 3407709
Madina TRAORÉ 3412847

Décembre 2018

Introduction

Le code Python utilisé pour ce projet est joint à ce rapport sous forme d'un notebook dans le fichier `mogpl_projet.ipynb` mis en page accompagné des sorties des tests effectués, ou dans le fichier `mogpl_projet.py` sous forme de code brut.

Les notations utilisées dans ce rapport sont les mêmes que celle du sujet. k est donc le nombre de villes contenant une ressource, I l'ensemble des villes et J l'ensemble des villes contenant une ressource. De nouvelles variables seront définies si nécéssaire. Les termes "ville i", "ville j" et "ressource j" qui sont parfois utilisés ci-après pour alléger les phrases désignent la ville d'indice i dans I et la ville ou ressource d'indice j dans J.
Les lettres entourées comme "ⓐ" dans les programmes linéaires permettent de noter un ensemble de contraintes de même type détaillés juste après. Elle sont aussi indiqué dans les commentaires du code à leur construction : "#type (a)".

Le chargement des fichiers est assuré par les fonction `parse_villes`, `parse_populations`, `parse_coordonees`, `parse_distances`.
Les fonctions `trace_reseau` et `save_reseau` permettent l'affichage et l'enregistrement des représentations graphiques des solutions.
La fonction `get_pop_tot` retourne la population totale nécessaire dans le calcul de γ.

Partie 1

Dans cette partie, nous souhaitons minimiser la fonction $f(x)$ sous les contraintes définies dans l'énoncé dans le cas où k villes se sont vues attribuer une ressource arbitrairement. Nous utilisons pour cela le programme linéaire suivant :

$$min\ z = \sum_{i=1}^{n} \sum_{j=1}^{k}\ d_{ij}x_{ij}$$

$$\begin{cases} ⓐ\ \forall i \in I,\ \sum_{j=1}^{k} x_{ij} = 1 \\ ⓑ\ \forall j \in J,\ \sum_{i=1}^{n} x_{ij}v_i \leq \gamma \end{cases} \tag{1}$$

$$\forall i,\ \forall j,\ x_{ij} \in \{0,1\}$$

- x_{ij} : ces variables sont binaires, de valeur 1 si la ville i dépend de la ressource j, 0 sinon. Il y a nk variables de ce type.
- ⓐ : Les contraintes de ce type assurent que pour chaque ville une seule des k ressources lui est allouée en fixant la somme pour une ville i des x_{ij} à 1. Il y a n contraintes de ce type.
- ⓑ : Les contraintes de ce type assurent que pour chaque ressource, la contrainte de sécurité n'est pas dépassée. Il y a k contraintes de ce type.
- z : La fonction objectif à minimiser est bien f(x).

Notre programme linéaire comporte n^2 variables et $n + k$ contraintes.
La fonction `optimisation_Q1` construit et résout ce programme linéaire.
La fonction `satisfaction` calcule la satisfaction moyenne et la satisfaction minimum comme étant respectivement $\frac{1}{d_{moy}}$ et $\frac{1}{d_{max}}$.

Partie 2

Dans cette partie, nous souhaitons minimiser la fonction $g(x)$ sous les contraintes définies dans l'énoncé dans le cas où k villes se sont vues attribuer une ressource arbitrairement. Nous utilisons pour cela le programme linéaire suivant :

$$min\ z = \sum_{i=1}^{n} \sum_{j=1}^{k} d_{ij}m_{ij} + \sum_{i=1}^{n} \sum_{j=1}^{k}\ \epsilon d_{ij}x_{ij}$$

$$\left\{ \begin{aligned} &\text{ⓐ}\ \forall i \in I,\ \sum_{j=1}^{k} x_{ij} = 1 \\ &\text{ⓑ}\ \forall j \in J,\ \sum_{i=1}^{n} x_{ij}v_i \leq \gamma \\ &\text{ⓒ}\ \forall i \in I,\ \forall j \in J,\ -d_{ij}x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{k} x_{ij}d_{ij}m_{ij} \geq 0 \\ &\text{ⓓ}\ \sum_{i=1}^{n}\sum_{j=1}^{k}\ m_{ij} = 1 \end{aligned} \right. \tag{2}$$

$$\forall i,\ \forall j,\ x_{ij} \in \{0,1\}, m_{ij} \in \{0,1\}$$

- x_{ij} : ces variables x_{ij} sont identiques à celles du modèle précédent. Nous avons nk variables de ce type.
- m_{ij} : ces variables sont binaires. Elles traduisent le fait que la distance d_{ij} est la distance maximum entre une ville i et la ressource j qui lui est alloué. Si c'est le cas la m_{ij} est à 1, 0 sinon. Nous avons nk variables de ce type.
- ⓐ et ⓑ : ces contraintes assurent les même rôles que dans le modèle précédent. Les nouvelles variables sont de coefficients nuls. Nous avons respectivement n et k contraintes de ces types.
- ⓒ : ces contraintes assurent que la distance d_{ij} quand $m_{ij} = 1$ est bien supérieure ou égale à tout autre distance entre une ville et sa ressource car leur différence est positive. Nous avons nk contraintes de ce type.
- ⓓ : cette contrainte assure qu'il n'existe qu'un unique couple (i, j) tel que $m_{ij} = 1$.
- z : La fonction objectif à minimiser est donc bien g(x) car grâce aux contraintes : m_{ij}=1 uniquement pour la distance maximum entre une ville et sa ressource.

Notre programme linéaire comporte $2nk$ variables et $nk + n + k + 1$ contraintes.
La fonction `optimisation_Q2` construit et résout ce programme linéaire.
Le fonction `PE` calcul le prix de l'équité comme défini dans l'énoncé.

Tests : Comparaison de $f(x)$ et $g(x)$

Les tests suivant sont réuinis par instance du problème puis par valeurs de α pour étudier à chaque fois le prix de l'équité.

Instance #1

Dans cette instance du problème on fixe k=3, les villes ressources sont : Courbevoie, Garches, Nanterre.

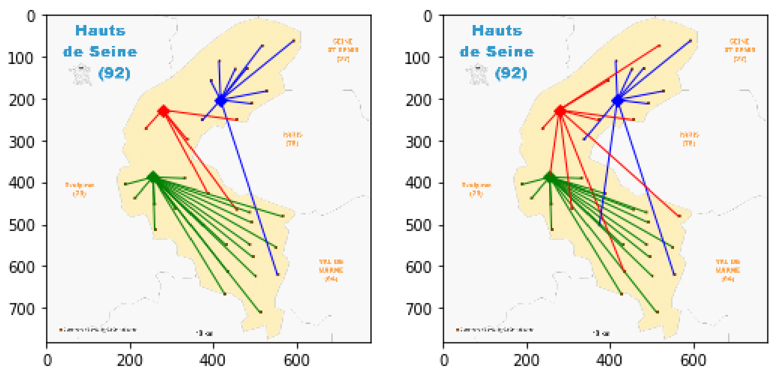

FIGURE 1 – Instance 1, $\alpha = 0.1$. Modèle 1 (gauche) et 2 (droite)

Prix de l'équité : 0.081825

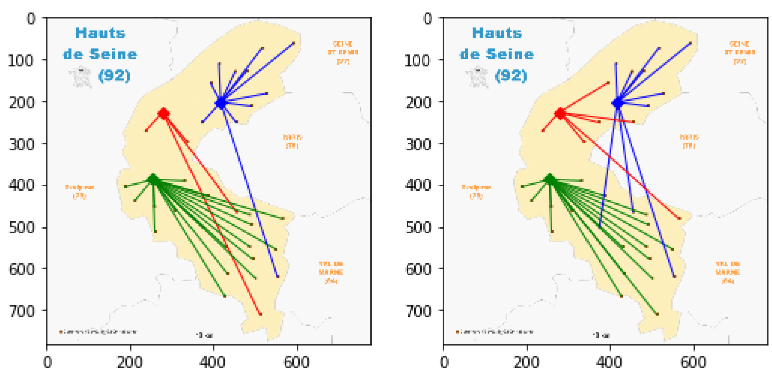

FIGURE 2 – Instance 1, $\alpha = 0.2$. Modèle 1 (gauche) et 2 (droite)

Prix de l'équité : 0.045684

Instance #2

Dans cette instance du problème on fixe k=4, les villes ressources sont : Courbevoie, Garches, Le Plessis-Robinson, Nanterre.

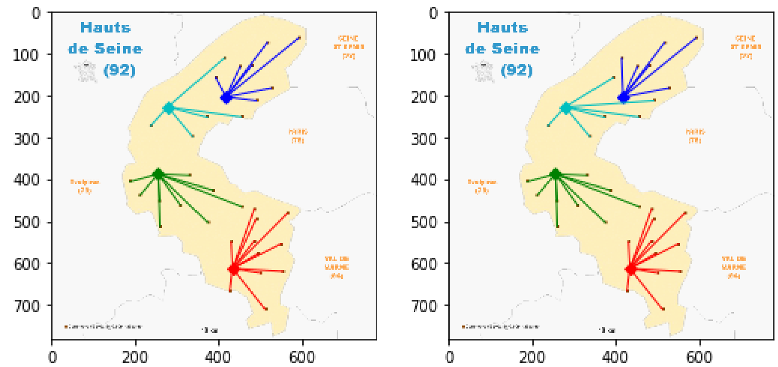

FIGURE 3 – Instance 2, $\alpha = 0.1$. Modèle 1 (gauche) et 2 (droite)

Prix de l'équité : 0.023015

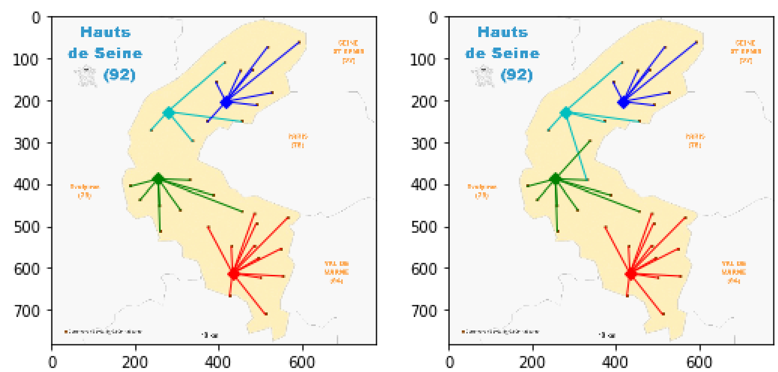

FIGURE 4 – Instance 2, $\alpha = 0.2$. Modèle 1 (gauche) et 2 (droite)

Prix de l'équité : 0.056842

Instance #3

Dans cette instance du problème on fixe k=5, les villes ressources sont : Courbevoie, Garches, Le Plessis-Robinson, Nanterre, Sevres.

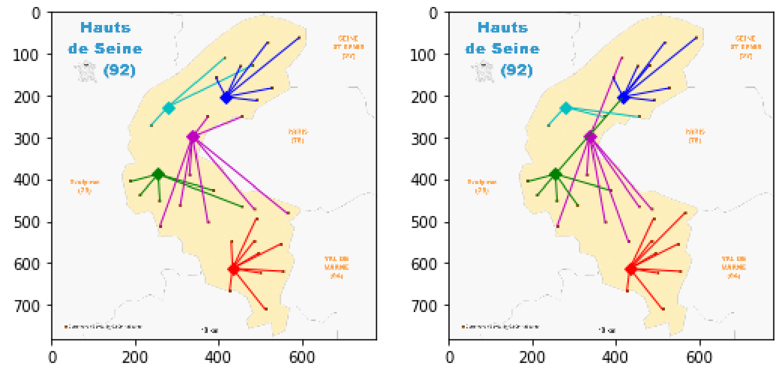

FIGURE 5 – Instance 3, $\alpha = 0.1$. Modèle 1 (gauche) et 2 (droite)

Prix de l'équité : 0.056129

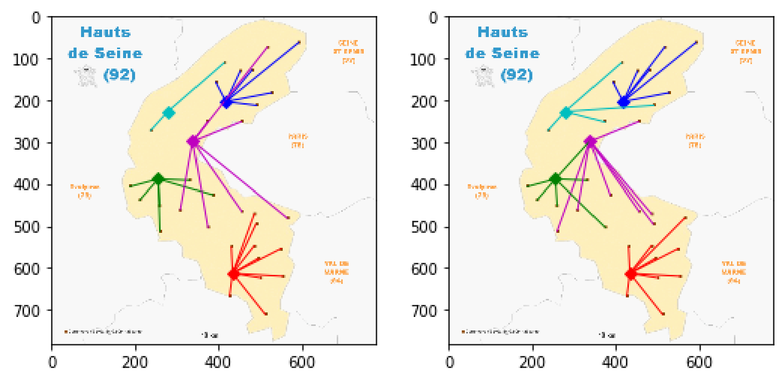

FIGURE 6 – Instance 3, $\alpha = 0.2$. Modèle 1 (gauche) et 2 (droite)

Prix de l'équité : 0.21113131194651338

Observations

On remarque que la première méthode consistant à minimiser $f(x)$ retourne des solutions optimales souvent fortement handicapantes pour certaines villes au profit de la collectivité. Le fait de relâcher la contrainte de sécurité en augmentant α ne semble pas impacter significativement ce phénomène, ce qui justifie l'utilisations de la deuxième méthode.
La minimisation de $g(x)$ permet de diminuer ce phénomène en donnant un poids élevé à la distance maximum entre une ville et sa contrainte dans z, rendant forcément la distance moyenne d'une ville à sa ressource plus grande. Ce compromis est exprimé par le prix de l'équité donné ici pour chaque instance.
On peut aussi remarquer graphiquement que l'instance #2 offre une solution qui semble efficace et sans grande distance d'une ville à sa ressource. En effet c'est l'instance dans laquelle les villes sont les mieux reparties géographiquement, et cela semble plus impactant que les deux autres paramètres sur lesquels nous jouons. Ce qui nous amène à nous demander comment trouver des positions optimales pour placer nos ressources, ce que nous allons faire dans la troisième partie.

Partie 3

Dans cette partie, nous souhaitons minimiser la fonction $g(x)$ sous les contraintes de l'énoncé sans fixer les positions de ressources et déterminer quelles sont les k villes optimales pour placer une ressource. Nous considérons ici J comme l'ensemble des villes potentielles pour placer une ressource, donc $J = I$. Nous utilisons le programme linéaire suivant :

$$min\ z = \textstyle\sum_{i=1}^{n} \sum_{j=1}^{k} d_{ij} m_{ij} + \sum_{i=1}^{n} \sum_{j=1}^{k} \epsilon d_{ij} x_{ij}$$

$$\begin{cases} ⓐ\ \forall i \in I,\ \displaystyle\sum_{j=1}^{k} x_{ij} = 1 \\ ⓔ\ \forall i \in I,\ \forall j \in J,\ x_{ij} - r_j \leq 0 \\ ⓑ\ \forall j \in J,\ \displaystyle\sum_{i=1}^{n} x_{ij} v_i \leq \gamma \\ ⓒ\ \forall i \in I,\ \forall j \in J,\ -d_{ij} x_{ij} + \displaystyle\sum_{i=1}^{n} \sum_{j=1}^{k} x_{ij} d_{ij} m_{ij} \geq 0 \\ ⓓ\ \displaystyle\sum_{i=1}^{n} \sum_{j=1}^{k} m_{ij} = 1 \\ ⓕ\ \displaystyle\sum_{j=1}^{k} r_j = 1 \end{cases} \tag{3}$$

$$\forall i,\ \forall j,\ x_{ij} \in \{0,1\}, m_{ij} \in \{0,1\}, r_{ij} \in \{0,1\}$$

- x_{ij} et m_{ij} : ces variables x_{ij} sont identiques à celles des modèles précédents. Nous avons n^2 variables de chaque type.
- r_j : ces variables sont binaires. Elles traduisent le fait que la ville d'indice j comporte une ressource. Si c'est le cas la variable r_j est à 1, 0 sinon. Nous avons n variables de ce type.
- ⓐ, ⓑ, ⓒ, ⓓ : ces contraintes assurent les même rôles que dans le modèle précédent. Les nouvelles variables sont de coefficients nuls. Nous avons respectivement n, n, n^2 et 1 contraintes pour ces types.
- ⓔ Ces contraintes assurent que la ville i ne peut obtenir de ressource que d'une ville j qui en possède une car $x_{ij} - r_j$ est négatif. Nous avons n^2 contraintes de ce type.
- ⓕ Cette contrainte assure que le nombre de villes possédant une ressources est k.
- z : La fonction objectif à minimiser est donc bien g(x) comme dans le modèle précédant.

Notre programme linéaire comporte $2n^2 + n$ variables et $2n^2 + 2n + 2$ contraintes.
La fonction `optimisation_Q3` construit et résout ce programme linéaire.

Tests : determination des villes optimales

k=3

Dans cette instance du problème on fixe k=3 et on cherche les villes optimales pour placer les ressources :

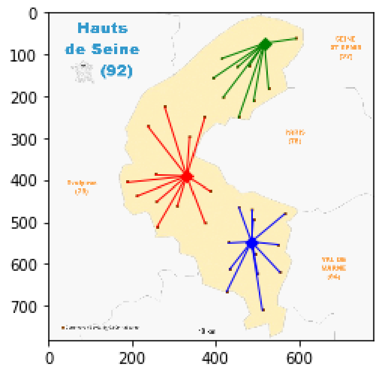

FIGURE 7 – Determination des villes optimales avec k=3

Les villes optimales pour k=3 sont Chatillon, Gennevilliers et Saint-Cloud.

k=4

Dans cette instance du problème on fixe k=4 et on cherche les villes optimales pour placer les ressources :

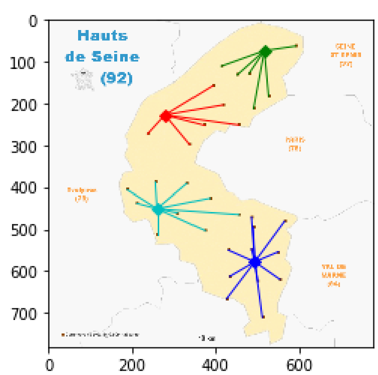

FIGURE 8 – Determination des villes optimales avec k=4

Les villes optimales pour k=4 sont Fontenay-aux-Roses, Gennevilliers, Nanterre, Ville-d'Avray.

k=5

Dans cette instance du problème on fixe k=5 et on cherche les villes optimales pour placer les ressources :

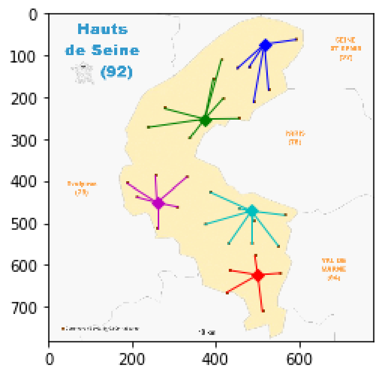

FIGURE 9 – Determination des villes optimales avec k=5

Les villes optimales pour k=3 sont Gennevilliers, Puteaux, Sceaux, Vanves, Ville-d'Avray.

Observations

Les solutions optimales sont donc des solutions dans lesquelles les villes sont bien réparties sur la carte. On voit aussi que, contrairement aux autres modèles, il n'y a ici aucune ressource surchargée, ni délaissée, elle sont toutes affectées à un nombre à peu près égal de villes.
Nous avons donc déterminé les villes optimales pour le placement des ressources selon les méthodes proposés. Devant le cas réel on pourrait penser à augmenter le modèle en prenant en compte le nombre d'habitant de chaque ville et en pondérant la satisfaction des villes en fonction par exemple.