Projet de Modélisation, optimisation, graphes, programmation linéaire

Cherchour Lièce He Xin

1 Répartition de patients dans les unités de soin

1.1 Programme linéaire pour des unités de soins fixés

$$minz = \frac{1}{\sum_{i\in I} v_i}\left(\sum_{i\in I} v_i \sum_{j\in J} x_{ij}d_{ij}\right)$$

$$s.c \begin{cases} \forall_{j\in J} \sum_{i\in I} v_i x_{ij} & \leq \gamma \\ \forall_{i\in I} \sum_{j\in J} x_{ij} & \geq 1 \\ \forall_{i\in I} \sum_{j\in J} x_{ij} & \leq 1 \end{cases}$$

$$x_{ij} \in 0,1,\ v_i, \gamma, n, k \in N$$

Nous avons x_{ij} qui permet de traduire l'affectation des agents d'une ville i à un centre de soins j. d_{ij} modélise la distance moyenne des habitants de la ville i pour se rendre au centre de soins j. v_i modélise tous simplement le nombre d'habitants dans la ville i. Nous cherchons donc ici à modéliser la distance moyenne d'affectation de chaque habitant (v_i) à un centre de soins. Un biais de notre programme ici est le poids des grandes villes qui leur permettrait d'avoir en général un centre de soins proche de chez eux. Nous avons des contraintes sur chaque centre j qui ne doit pas dépasser γ patients, avec :

$$\gamma = \frac{1+\alpha}{k}\sum_{i=1}^{n} v_i$$

Avec α, un paramètre strictement positif qui nous permettra de régler notre modèle dans le cas où il n'existe pas de solution (γ trop petit). Ces contraintes permettent de vérifier qu'on ne surcharge pas un centre de soins par rapport au autre. Ensuite, nous vérifions que toute ville i est relié à au plus à un centre de soins.

Il n'est pas possible de résoudre ce problème à l'aide d'un algorithme de flot maximum à coût minimum, car il faudrait contraindre les villes à envoyer leurs populations vers un seul centre de soins.

1.2 Résultat gurobi pour 3 et 4 villes fixées.

Pour le moment , nous fixons α = 0,2.

Les villes choisies pour les tests avec 3 villes fixées sont : Toulouse, Nice et Nantes. Nous obtenons : 272,748.

Si nous testons maintenant avec 4 villes fixées qui seront : Toulouse, Nice, Nantes et Montpellier. Nous obtenons : 250.758.

Maintenant si nous effectuons les mêmes tests avec α = 0,1.

Nous obtenons respectivement 273.012 pour k = 3 et 264.681 pour k = 4. Les solutions obtenues sont moins bonnes car la restrictions sur les centres de soins avec α = 0,1 est plus forte.

Plus il y à de villes, plus il y à de choix possibles pour répartir les habitants sans atteindre la limite γ pour un centre de soins précis, on arrive donc plus facilement à de meilleurs résultats car on est moins contraint par celle-ci.

2 Localisation optimale des unités de soin

2.1 Programme linéaire pour localiser des unités de soins non fixées

Les questions suivantes ont été réalisées avec $\alpha = 0{,}2$.

Nous allons maintenant reprendre le programme linéaire de la question 1 qui est :

$$minz = \frac{1}{\sum_{i\in I} v_i} \left(\sum_{i\in I} v_i \sum_{j\in J} x_{ij} d_{ij} \right)$$

$$s.c \begin{cases} \forall_{j\in J} \sum_{i\in I} v_i x_{ij} & \leq \gamma \\ \forall_{i\in I} \sum_{j\in J} x_{ij} & \geq 1 \\ \forall_{i\in I} \sum_{j\in J} x_{ij} & \leq 1 \end{cases}$$

$$x_{ij} \in 0, 1,\ v_i, \gamma, n, k \in N$$

Mais nous allons rajouter une variable y_j définie comme tel :

$y_j = 1$, si nous avons un centre de soins dans la ville j.
$y_j = 0$, si nous n'avons pas de centre de soins dans la ville j.

Nous allons donc ajouter quelque contraintes :

$$s.c \begin{cases} \forall_{j\in I} \sum_{i=1}^{n} x_{ij} & \leq n y_j \\ \sum_{j=1}^{n} y_j & \geq k \\ \sum_{j=1}^{n} y_j & \leq k \end{cases}$$

Nous avons en premier temps une contrainte sur toutes les villes, tel que nous ne pouvons pas avoir un patient i assigné au centre de soins j si il n'existe pas de centre dans la ville j. Les 2 dernières contraintes nous permettent de vérifier que nous n'avons pas plus de k centre de soins répartis dans les n villes. Cela nous a donc permis d'ajouter les centres de soins (y_j) dans les variables de décisions de notre modèle.

Nous avons donc maintenant le modèle suivant :

$$minz = \frac{1}{\sum_{i\in I} v_i} \left(\sum_{i\in I} v_i \sum_{j\in I} x_{ij} d_{ij} \right)$$

$$s.c \begin{cases} \forall_{j\in I} \sum_{i\in I} v_i x_{ij} & \leq \gamma \\ \forall_{i\in I} \sum_{j\in I} x_{ij} & \geq 1 \\ \forall_{i\in I} \sum_{j\in I} x_{ij} & \leq 1 \\ \forall_{j\in I} \sum_{i=1}^{n} x_{ij} & \leq n y_j \\ \sum_{j=1}^{n} y_j & \geq k \\ \sum_{j=1}^{n} y_j & \leq k \end{cases}$$

$$x_{ij}, y_j \in 0, 1,\ v_i, \gamma, n, k \in N$$

Nous allons donc maintenant comparer les solutions obtenues avec celles obtenues à la question 1 :

Pour k = 3, nous obtenons : ['Nantes', 'Montpellier', 'Dijon'] avec comme distance maximum d'un habitant à un centre de soins de 501.

Pour k = 4, nous obtenons : ['Toulouse', 'Nantes', 'Reims', 'Toulon'] avec comme distance maximum d'un habitant à un centre de soins de 398.

Pour k = 5, nous obtenons : ['Toulouse', 'Nice', 'Montpellier', 'Rennes', 'Reims'] avec comme distance maximum d'un habitant à un centre de soins de 347.

Il est compliqué de comparer nos résultats à ceux de la question 1, l'exercice 2 nous permet de trouver la combinaison idéale de centres de soins pour minimiser notre fonction objectif, alors que la question 1 doit essayer de la réduire sans changer la localisation des centres de soins. En énumérant les solutions dans la question 1 et en gardant la meilleur solution, on peut bien observer que celle-ci correspond à la solution donnée dans la question 2.1.

2.2 Programme linéaire pour localiser des unités de soins non fixées de façon équitable

Nous allons maintenant modifier la fonction objectif, nous souhaitons minimiser la plus grande distance d'un individu à un centre de soins:

$$minz = max_{i\in I} d(i, f(i)))$$

$$s.c \begin{cases} \forall_{j\in I} \sum_{i\in I} v_i x_{ij} & \leq \gamma \\ \forall_{i\in I} \sum_{j\in I} x_{ij} & \geq 1 \\ \forall_{i\in I} \sum_{j\in I} x_{ij} & \leq 1 \\ \forall_{j\in I} \sum_{i=1}^{n} x_{ij} & \leq n y_j \\ \sum_{j=1}^{n} y_j & \geq k \\ \sum_{j=1}^{n} y_j & \leq k \end{cases}$$

$$x_{ij}, y_j \in 0, 1,\ v_i, \gamma, n, k \in N$$

Nous pouvons réduire ce problème de **minmax** au programme linéaire suivant :

$$min\ z$$

$$s.c \begin{cases} \forall_{i\in I}, \sum_{j\in I} x_{ij} d_{ij} & \leq z \\ \forall_{j\in I}, \sum_{i\in I} v_i x_{ij} & \leq \gamma \\ \forall_{i\in I}, \sum_{j\in I} x_{ij} & \geq 1 \\ \forall_{i\in I}, \sum_{j\in I} x_{ij} & \leq 1 \\ \forall_{j\in I} \sum_{i=1}^{n} x_{ij} & \leq n y_j \\ \sum_{j=1}^{n} y_j & \geq k \\ \sum_{j=1}^{n} y_j & \leq k \end{cases}$$

$$x_{ij}, y_j \in 0, 1,\ v_i, \gamma, n, k \in N$$

Nous allons maintenant comparer nos résultats à celui de la question 2.1 :

Pour k = 3, nous obtenons : ['Montpellier', 'Bordeaux', 'Reims'] avec comme distance maximum d'un habitant à un centre de soins de 458.

Pour k = 4, nous obtenons : ['Nantes', 'Montpellier', 'Reims', 'Grenoble'] avec comme distance maximum d'un habitant à un centre de soins de 347.

Pour k = 5, nous obtenons : ['Toulouse', 'Reims', 'Toulon', 'Dijon', 'Angers'] avec comme distance maximum d'un habitant à un centre de soins de 332.

On observe bel et bien une amélioration, la distance maximum d'un individu à son centre de soins est plus faible, cela est précisément dû au fait que l'on ne prend plus en compte le poids des villes (par exemple, Toulouse possédant beaucoup d'habitants est assez avantagée) en se basant maintenant sur la distance parcourue par un habitant pour se rendre à un centre de soins. De plus, il existe plusieurs solutions optimal réalisable pour k = 4 et k = 5.

3 Équilibrage des charges des unités de soin

3.1 Modélisation sous la forme d'un problème de flot maximum de coût minimum et d'un problème d'un problème de transport

Il existe 2 façons de résoudre ce problème, nous allons ici le modéliser de 2 façons différentes en donnant un exemple de graphe pour la modélisation en problèmes de transport.

3.1.1 Première modélisation : Problème de flot maximum à coût minimum

Nous pouvons dans un premier temps voir ce problème comme celui du flot maximum à coût minimum.

Imaginons que nos centres de départ soient Toulouse, Nice, Nantes, Montpellier et Strasbourg, et que chacun possède respectivement 15, 150, 30, 60 et 20 patients nécessitant une prise en charge dans leur secteur.

Alors 15, 150, 30, 60, 20 représentent le flot allant de la source à chacune des villes de départ.

Nous souhaitons obtenir un équilibrage, cela revient à calculer

$$p\_equi = \frac{\sum_{i=1}^{n} pi}{n}$$

où n désigne le nombre de centre de soins dont nous disposons.

p_equi représente ici le flot venant des centres d'arrivées au puits.

Parfois, cela nécessite quelques ajustements: des centres qui devront accueillir plus de patients que d'autres car il n'est pas possible de répartir parfaitement à tous les coups.

Nous allons ensuite ajouter au graphe d'écart les distances séparant un centre i d'un centre j, modélisées par les distances qui nous sont données en annexe.

De cette façon, nous pouvons modéliser ce problème comme celui du flot maximum à coût minimum.

3.1.2 Deuxième modélisation : Problème de transport

Il est très simple de voir que la représentation vue plus haut peut être vue comme un problème de transport.

Voici la matrice associé :

Ville	Toulouse	Nice	Nantes	Montpellier	Strasbourg	Nombre de patients dans le centre
Toulouse	0	562	585	242	949	15
Nice	562	0	1143	326	790	150
Nantes	585	1143	0	824	860	30
Montpellier	242	326	824	0	791	60
Strasbourg	949	790	860	791	0	20
Nombre de patients qu'il souhaite	255/5	255/5	255/5	255/5	255/5	255
	55	55	55	55	55	//

Figure 1: Matrice associée à la représentation du problème de transport

On peut facilement voir un problème qui peut être résolu à l'aide de l'algorithme **dual-primal**, il faut par contre représenter les flots d'entrée par le nombre de patients dans un centre et le flot de sortie par le nombre de patients que chaque centre souhaité.

3.2 Résolution du problème à l'aide de l'algorithme de flot maximum de coût minimum sur différentes instances pi

Nous effectuons ces tests sur ['Toulouse', 'Reims', 'Toulon', 'Dijon', 'Angers']:

Pour p = [61, 312, 107, 14, 3] et p_equi = [100, 100, 99, 99, 99], Nous obtenons 95438

Pour p = [223, 157, 29, 46, 15] et p_equi = [94, 94, 94, 94, 94], Nous obtenons 229919

Pour p = [390, 10, 8, 14, 16] et p_equi = [88, 88, 88, 87, 87], Nous obtenons 192953

Pour p = [20, 47, 25, 3, 25] et p_equi = [24, 24, 24, 24, 24], Nous obtenons 8986

4 Organisations du code

Notre code est réparti dans 4 fichiers. Les fichiers graph_function, linear_programming_1_2, utils contiennent des fonctions commentées qui seront utilisées dans le fichier principal projet.py, il est possible de lancer le fichier projet.py directement comme un script depuis la ligne de commande (par exemple **python projet.py**). Il peut être nécessaire d'installer **gurobi** et **networkx** qui ne sont pas des librairies inclus dans la librairie standard de python.

Le fichier utils contient une fonction permettant de récupérer les élèments intéressants de notre fichier csv (nombre de villes, population pour chacune d'entre elles, ...). Elle contient également des fonctions permettant de :

- récupérer le nom de villes à l'aide de leur index
- extraire une partie de la table des distances initiales
- Deux fonctions pour générer le nombre de patients du secteur j nécessitant une prise en charge pour la question 3.2 et de calculer le nombre de patients par ville pour être convenablement réparti

Le fichier linear_programming_1_2 contient principalement l'implantation des programmes linéaires demandés aux questions 1.2, 2.1 et 2.2. La fonction définissant le programme linéaire de l'exercice 2.1 renvoie également la distance maximum d'un individu à son centre de soins et les centres de soins assignés pour simplifier la comparaison avec les différents exercices. Le programme linéaire de l'exercice 2.2 nous renvoie la combinaison de ville qui permet de minimiser sa fonction objectif. Dans l'exercice 2.2, plusieurs solution optimal sont possible, une partie du code commenté utilise GRB.Param.PoolSolutions et GRB.Param.PoolSearchMode pour trouvers ces solutions.

Le fichier graph_function contiens deux fonctions permettant de résoudre l'exercice 3.2.

- Une fonction permettant la construction du graphe biparti nécessaire pour résoudre la question
- Une fonction permettant d'effectuer notre algorithme du flot maximum à coût minimum sur un graphe passé en paramètre.

Le fichier projet.py contient une exécution des fonctions telle qu'il permet dérouler le projet exercice par exercice. La résolution avec gurobi donne un affichage peu lisible mais il n'est pas réellement possible d'améliorer cet aspect là. Au début du fichier se trouvent les différents réglages pour chaque exercice (Par exemple α, k et la liste des indices des villes pour l'exercice 1.2), l'exécution du code est commentée pas à pas pour bien comprendre le déroulement de celui-ci.