BENSLIMANE Amine (28716642)
GIANG Phuong-Thu, Cécile (3530406)
MOGPL - Groupe 1

RAPPORT DE PROJET - MOGPL

2021-01-02

1 Répartition de patients dans les unités de soin

1.1 Formalisation de l'exercice

Notons $I = \{1, ..., n\}$ un ensemble de n villes situées dans un territoire de référence, et v_i la population de la ville i pour tout $i \in \mathbb{N}$. Dans cette partie, k unités spéciales de traitement sont implantées dans k villes, avec $k < n$. On note $J \subset I$ l'ensemble des villes où sont implantées les secteurs.
Nous notons X la matrice $n \times k$ telle que x_{ij} vaut 1 si les patients habitant dans la ville i sont traités dans la ville-secteur j, et 0 sinon. Notons aussi d_{ij} la distance moyenne qui sépare la ville i de la ville j.

Avec la formalisation précédente, les trois contraintes données par l'énoncé se réécrivent ainsi :

1ère contrainte : La population totale des villes composant un secteur j ne doit pas dépasser γ :

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

$$\sum_{i=1}^{n} v_i x_{ij} \leq \gamma \quad \text{pour un secteur } j \text{ fixé}$$

2ème contrainte : Les secteurs $j \in J$ forment une partition des n villes :

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

3ème contrainte : Une ville i n'appartient qu'à un et un seul secteur :

$$\sum_{j=1}^{k} x_{ij} = 1 \quad \text{pour une ville } i \text{ fixée}$$

Le programme linéaire qui détermine les secteurs de service des k unités de soin de manière à minimiser la distance moyenne de chaque habitant à l'unité de soin dont il dépend s'écrit alors ainsi :

$$min \quad \frac{1}{\sum_{i=1}^{n} v_i} \sum\nolimits_{i=1}^{n} \sum\nolimits_{j=1}^{k} d_{ij} \times v_i \times x_{ij}$$

$$s.c \quad \begin{cases} \sum_{i=1}^{n} v_i x_{ij} \leq \gamma & \text{pour tout secteur } j \in J \\ \sum_{i=1}^{n} \sum_{j=1}^{k} x_{ij} = n & \\ \sum_{j=1}^{k} x_{ij} = 1 & \text{pour toute ville } i \in I \end{cases}$$

$$x_{ij} \in \{0, 1\}, \quad i \in I, \quad j \in J$$

Ce problème ne peut pas être modélisé comme un problème de flot maximum à coût minimum.

En effet, cela signifierait pouvoir résoudre ou bien un problème de transport ou bien un problème d'affectation. Dans les deux cas, c'est impossible d'y modéliser les deux premières contraintes du problème en même temps.

1.2 Implémentation et résultats

Pour l'implémentation de ce programme linéaire sous Python, nous commençons par lire le fichier `villes.csv` afin de stocker les données numériques qu'il contient. Nous créons un vecteur `v` qui contient les populations de chaque ville, ainsi que `d`, la matrice $n \times n$ qui contient la distance entre deux villes. Elle est initialement triangulaire inférieure mais nous choisissons de la rendre symétrique pour faciliter nos calculs (en partant de `d` triangulaire inférieure, `d += d.T`).

Nous créons également la liste de toutes villes `I` ainsi que la liste des villes-secteurs `J`, et déclarons $n \times k$ variables de décision stockées dans `x`, telles que `x[i,j]` vaut 1 si les patients de la ville `i` doivent se faire soigner dans la ville `j`.

La fonction objectif s'écrit alors :

```
obj = LinExpr();

for i in range(n):
    for j in range(k):
        obj += d[i,J[j]]* v[i] * x[i,j]
obj /= np.sum(v)

# definition de l'objectif
m.setObjective(obj,GRB.MINIMIZE)
```

Listing 1 – Définition de l'objectif

Après avoir codé les trois contraintes *(cf. code source)*, nous testons comme demandé notre PL en utilisant les 15 villes données en annexe.

Choisissons 3 villes dans lesquelles on supposera qu'une unité de soin spéciale a été implantée, par exemple Nice, Le Havre et Dijon (ce qui nous donne la liste de secteurs `J = [1, 11, 13]`) et calculons leurs secteurs de service. Nous obtenons le résultat suivant :

```
Valeur de la fonction objectif : 305.2985967176322

Solution optimale:
 [[1 0 0]
 [1 0 0]
 [0 1 0]
 [1 0 0]
 [0 0 1]
 [0 0 1]
 [0 1 0]
 [0 1 0]
 [0 0 1]
 [0 0 1]
 [1 0 0]
 [0 1 0]
 [0 0 1]
 [0 0 1]
 [0 1 0]]

Les secteurs sont:
    Secteur Nice :  ['Toulouse', 'Nice', 'Montpellier', 'Toulon']
    Secteur LeHavre :  ['Nantes', 'Lille', 'Rennes', 'LeHavre', 'Angers']
    Secteur Dijon :  ['Strasbourg', 'Bordeaux', 'Reims', 'Saint-Étienne', 'Grenoble',
'Dijon']
```

FIGURE 1 – Resultat obtenus pour les 15 villes

Ainsi, pour une la liste de villes-secteurs `J = [1, 11, 13]` (i.e. Nice, Le Havre et Dijon) et pour $\alpha = 0.1$, l'affectation de villes en secteurs est :

- secteur Nice : Toulouse, Nice, Montpellier, Toulon
- secteur Le Havre : Nantes, Lille, Rennes, Le Havre, Angers
- secteur Dijon : Strasbourg, Bordeaux, Reims, Saint Etienne, Grenoble, Dijon

Cette affectation, optimale pour un tel J, nous permet d'obtenir une distance moyenne habitant-secteur de $305,3$. Nous obtenons la même affection pour $\alpha = 0.2$ et $\alpha = 0.3$.
Nous avions pris exprès 3 villes assez bien réparties dans la France entière. Si maintenant nous prenons 3 secteurs très proches : par exemple Nantes, Rennes, Angers et donc `J = [2, 7, 14]`. Nous obtenons, pour $\alpha = 0.1$:

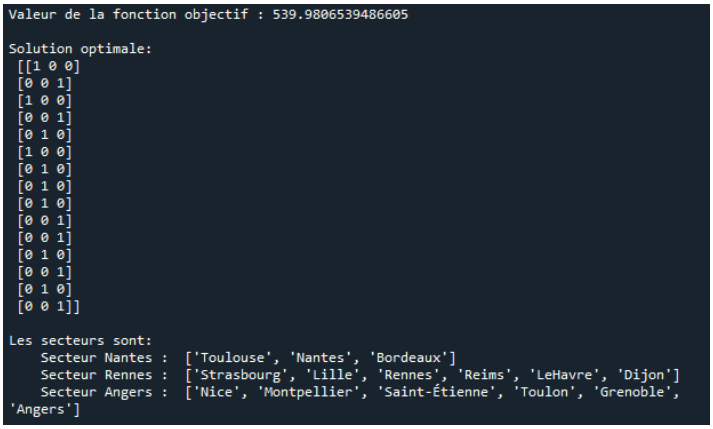

FIGURE 2 – Resultat obtenus pour Nantes, Rennes et Angers

Cette fois-ci l'affectation optimale nous renvoie une distance moyenne plus grande $(539,98)$, comme nous nous y attendions. En augmentant α :

- $\alpha = 0.2 : z = 537.19$
- $\alpha = 0.3 : z = 535.9$
- $\alpha = 0.8 : z = 525.9$
- $\alpha \geq 0.8 : z = 525.9$

Plus α augmente, plus la fonction objectif est faible.

Maintenant pour J contenant quatre villes. Comme avant, prenons deux cas : un J contenant 4 villes bien réparties dans la France entière, et un J contenant 4 villes très proches les unes des autres.

Cas 1 (quatre villes bien réparties) :

Prenons les villes-secteurs Nice, Strasbourg, Bordeaux et Rennes : `J = [1, 4, 5, 7]`

- $\alpha \leq 0.002276898$: Pas de solutions satisfiable
- $\alpha = 0.002276899 : z = 472.85$
- $\alpha = 0.003 : z = 472.85$
- $\alpha = 0.01 : z = 339.19$
- $\alpha = 0.05 : z = 241.13$
- $\alpha = 0.1 : z = 208.84$
- $\alpha = 0.3 : z = 206.91$
- $\alpha \geq 0.3 : z = 206.91$

Cas 1 (quatre villes très proches les unes des autres) :

Prenons les villes-secteurs Nice, Montpellier, Toulon, Grenoble : `J = [1, 3, 10, 12]`

- $\alpha \leq 0.002276898$: Pas de solution satisfiable
- $\alpha = 0.002276899 : z = 542.56$
- $\alpha = 0.003 : z = 542.56$
- $\alpha = 0.05 : z = 490.21$
- $\alpha = 0.1 : z = 485.56$
- $\alpha = 0.8 : z = 404.28$
- $\alpha = 1 : z = 401.19$
- $\alpha \geq 1.281857203 : z = 397.89$

De manière générale, plus on augmente le nombre de secteurs, plus notre fonction objectif est minimisée.

En effet, la distance moyenne qu'un patient doit parcourir diminue ! Il vaut également mieux choisir des secteurs bien réparties dans le territoire, plutôt que des secteurs très proches les uns de autres, afin d'obtenir un résultat optimal.

Au fur et à mesure que α augmente, la fonction objectif est minimisée : c'est compréhensible car en augmentant α, on autorise nos secteurs à accueillir plus de patients (augmentation de γ), ce qui permet aux secteurs

d'accueillir même les villes proches de faible population, qui auraient été discriminées si α était trop faible (car notre modélisation favorise les villes à forte population).

Attention cependant : il vaut mieux avoir beaucoup de secteurs, mais il faut faire attention à bien augmenter la quantité α en concordance : par exemple pour un secteur `J = [0,1,2,3,4,5,6,7]` contenant 8 villes, garder $\alpha = 0.1$ mène à une solution qui n'est pas à satisfaisante car toutes les villes ne sont pas réparties dans un secteur ; elle n'est donc pas optimale :

FIGURE 3 – Resultat obtenu non optimal

En augmentant α à 0.8 cependant, nous obtenons bien une solution réalisable et optimale, et la distance moyenne parcourue par un patient est alors de 77.9 seulement !

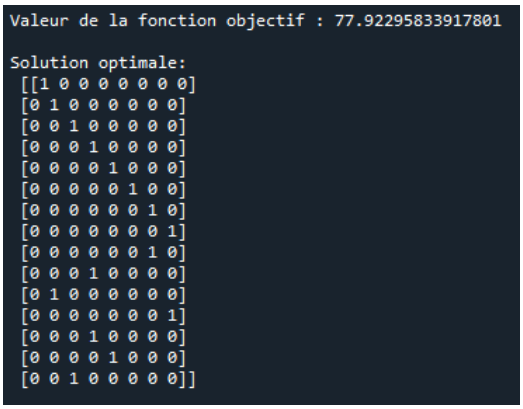

FIGURE 4 – Resultat obtenu optimal

Cependant il faut veiller à ne pas prendre trop de secteurs avec des α trop grands, car cela reviendrait seulement à affecter les habitants d'une ville à leur secteur le plus proche, ce qui n'est plus intéressant. Ainsi prendre une liste de secteurs `J` telle que `I = J` reviendrait seulement à affecter les habitants d'une ville à leur même ville (qui est un secteur). Ainsi, le PL avec `J = I` et $\alpha = 2$ rend la solution optimale qui à chaque ville i affecte le secteur i *(matrice identité)* et la fonction objectif vaut alors 0.0.

2 Localisation optimale des unités de soin

On considère maintenant le scénario dans lequel la localisation des unités de soin n'est pas encore décidée et que seul le nombre k de telles unités est connu. On cherche alors à déterminer la localisation optimale des unités de soin et à définir leurs secteurs de service.

2.1 Premier programme

On cherche à déterminer, pour un k donné, la localisation optimale des unités de soin et à définir leurs secteurs de service, sans énumérer explicitement les sous-ensembles de k villes parmi n pour appliquer le modèle de la partie précédente.

On se propose donc de modifier le programme linéaire précédent pour intégrer le fait que les positions des unités de soin font maintenant partie des variables de décision. Rappelons que notre fonction objectif était :

$$min \qquad \frac{1}{\sum_{i=1}^{n} v_i} \sum_{i=1}^{n} \sum_{j=1}^{k} d_{ij} \times v_i \times x_{ij}$$

Simplement considérer nos villes-secteurs j comme des variables de décision n'est pas une solution pour deux raisons :

- on ne peut indexer `d` et `x` que sur des entiers, et non des variables de décisions Gurobi
- en passant nos villes-secteurs j comme variables de décision, notre fonction objectif ne serait plus linéaire

Introduisons donc n nouvelles variables $z_0, z_1, ..., z_{n-1}$ telles que z_j vaut 1 si la ville j est un secteur, 0 sinon. En multipliant nos x_{ij} par z_j, nous pouvons annuler les x_{ij} où j n'est pas un secteur : les variables z_j agissent comme une fonction indicatrice !

N'oublions pas de rajouter de nouvelles contraintes aux contraintes précédentes et de modifier la 2ème contrainte :

4ème contrainte : Les variables z_j valent 0 ou 1 :

$$z_i \leq 1 \quad \text{pour toute ville i}$$

5ème contrainte : Il y a en tout k variables z_i :

$$\sum_{i=1}^{n} z_i = k$$

2ème contrainte : La contrainte de partition des n villes dans les k secteurs devient :

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \times z_j = n$$

Nous pouvons ainsi réécrire notre programme linéaire ainsi :

$$min \qquad \frac{1}{\sum_{i=1}^{n} v_i} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \times v_i \times x_{ij} \times z_j$$

$$s.c \quad \begin{cases} \sum_{i=1}^{n} v_i x_{ij} \leq \gamma & \text{pour tout secteur } j \\ \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \times z_j = n & \\ \sum_{j=1}^{n} x_{ij} = 1 & \text{pour toute ville } v_i \\ z_i \leq 1 & \text{pour toute ville i} \\ \sum_{i=1}^{n} z_i = k & \end{cases}$$

$$x_{ij} \in \{0,1\}, \quad z_j \in \{0,1\}, \quad i \in I, \quad j \in J$$

Pour l'implémentation, nous rajoutons les variables de décision correspondant à nos z_j et modifions l'objectif :

```
# Variables xij qui valent 1 si les patients de la ville i sont affect s au
# secteur j, 0 sinon
x = m.addVars(n, n, vtype=GRB.BINARY, lb=0)

# Variables zj qui valent 1 si la ville j est un secteur, 0 sinon
z = m.addVars(n, vtype=GRB.BINARY, lb=0)

# maj du modele pour integrer les nouvelles variables
m.update()

# definition de l'objectif
obj = LinExpr();
for i in range(n):
    for j in range(n):
        obj += d[i,j]* v[i] * x[i,j] * z[j]
obj /= np.sum(v)

m.setObjective(obj,GRB.MINIMIZE)
```

Listing 2 – Définition de l'objectif

C'est maintenant à l'utilisateur de rentrer lui-même la valeur de k. Nous rajoutons les nouvelles contraintes et testons notre programme. Nous testons notre programme pour $k = 1$ (secteurs : Saint-Etienne, $z = 498.7$) et $k = 2$ (secteurs : Montpellier et Angers, $z = 299.2$) et obtenons bien les secteurs optimaux à chaque fois (nous avons testé pour toutes les possibilités de secteurs).
Comme demandé, testons également pour $k = 3, 4$ et 5.

Tests pour k = 3 : La fonction n'a pas de solution satifiable pour $\alpha = 0.1$. Nous augmentons donc α. Le programme fonctionne pour $\alpha = 0.3$ mais la valeur de la fonction objectif est plus petite pour $\alpha = 0.5$. Prenons donc $\alpha = 0.5$. Nous obtenons les résultats suivants :

```
Valeur de la fonction objectif : 208.62693365128348

Solution optimale:
 [[0 1 0]
 [0 1 0]
 [1 0 0]
 [0 1 0]
 [0 0 1]
 [1 0 0]
 [0 0 1]
 [1 0 0]
 [0 0 1]
 [0 1 0]
 [0 1 0]
 [0 0 1]
 [0 1 0]
 [0 0 1]
 [1 0 0]]

Les secteurs sont:
    Secteur Nantes :  ['Nantes', 'Bordeaux', 'Rennes', 'Angers']
    Secteur Montpellier :  ['Toulouse', 'Nice', 'Montpellier', 'Saint-Étienne',
'Toulon', 'Grenoble']
    Secteur Reims :  ['Strasbourg', 'Lille', 'Reims', 'LeHavre', 'Dijon']
```

FIGURE 5 – Resultat obtenu pour $k = 3$, $\alpha = 0.1$

Pour $k = 3$ et $\alpha = 0.5$, les secteurs optimaux sont donc : Nantes, Montpellier et Reims, et la fonction objectif vaut 208.6.

Tests pour k = 4 : La fonction n'a pas de solution satifiable pour $\alpha = 0.1$. Prenons $\alpha = 0.3$. Nous obtenons les résultats suivants :

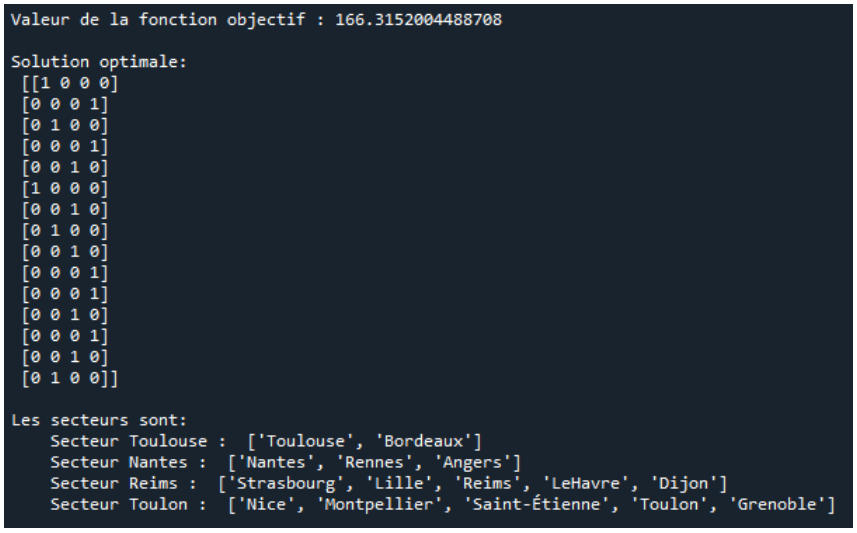

FIGURE 6 – Resultat obtenu , $k = 4$, $\alpha = 0.3$

Pour $k = 4$ et $\alpha = 0.3$, les secteurs optimaux sont donc : Toulouse, Nantes, Reims et Toulon, et la fonction objectif, c'est à dire la distance moyenne parcourue par un habitant avec cette liste optimale de secteurs, vaut 166.3.

Tests pour k = 5 : La fonction n'a pas de solution satifiable pour $\alpha = 0.1$. Prenons $\alpha = 0.5$. Nous obtenons les résultats suivants :

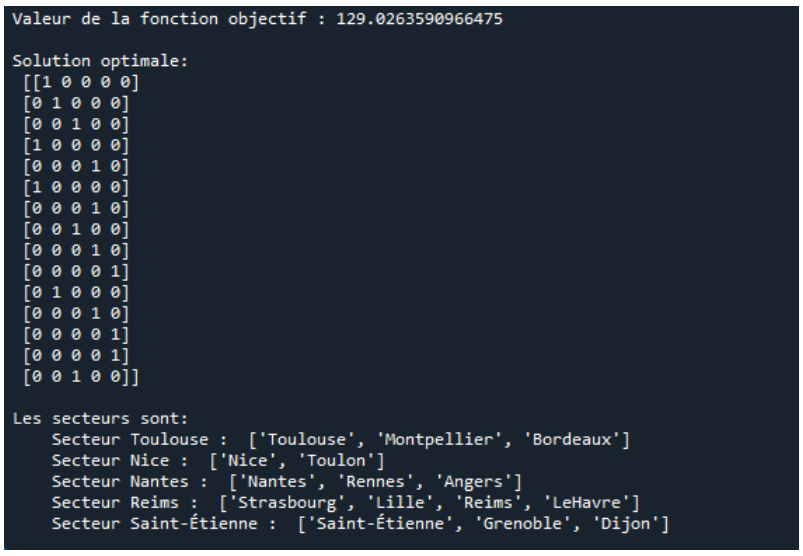

FIGURE 7 – Resultat obtenu, $k = 5$, $\alpha = 0.5$

Pour $k = 5$ et $\alpha = 0.5$, les secteurs optimaux sont donc : Toulouse, Nice, Nantes, Reims et Saint-Etienne, et la fonction objectif vaut 129.02.

Après avoir testé le programme de la première partie avec différentes listes de villes-secteurs `J` de taille $k = 3, 4$ et 5, nous n'en trouvons aucune qui rende une distance (valeur de la fonction objectif) plus petite que pour les secteurs déterminés par notre programme courant. Le programme de la première partie rend également la même valeur d'objectif sur nos listes de villes-secteurs optimales. Nous remarquons également que, contrairement à la partie 1 où, en choisissant à la main, on risquait parfois de se retrouver avec des villes-secteurs qui n'étaient même pas dans leur propre secteur (cas des villes à faible population proche de villes à forte population, avec un α petit), comme la répartition des secteurs est maintenant optimisée ce n'est plus un risque.

2.2 Deuxième programme

La solution précédente avantage les habitants de grandes villes qui ont plus de poids et ne permet pas l'équité de l'accès au soin (distance de chaque individu à son unité de soin). On note maintenant $f(i)$ le numéro de la ville-secteur à laquelle est affiliée la ville i. Nous voulons désormais trouver la localisation optimale des secteurs qui minimise la distance maximale que doit parcourir un patient pour aller jusqu'à sa ville-secteur. On cherche donc à minimiser la quantité $\max_{i \in I} d(i, f(i))$.

Comme pour le programme précédent, un programme avec une telle fonction objectif ne serait pas linéaire. Pour linéariser la recherche de la quantité $\max_{i \in I} d(i, f(i))$, nous introduisons une nouvelle variable de décision `d_max`, qui représentera la distance maximale entre une ville i et sa ville-secteur pour un certain k. La fonction objectif est alors :

$$min \quad d_{max}$$

Pour s'assurer que d_{max} est bien la distance maximale entre une ville et une ville-secteur, il faut que cette distance soit supérieure à toutes les autres distances pour des secteurs donnés. On rajoute la contrainte suivante :

6ème contrainte : d est la distance maximale entre un habitant d'une ville i et une ville-secteur j :

$$d_{max} \geq d_{ij} \times x_{ij} \times z_j \text{ pour toute ville } i \text{ et pour tout secteur } j$$

Nous pouvons ainsi réécrire notre programme linéaire ainsi :

$$min \quad d_{max}$$

$$s.c \quad \begin{cases} \sum_{i=1}^{n} v_i x_{ij} \leq \gamma & \text{pour tout secteur } j \\ \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \times z_j = n & \\ \sum_{j=1}^{n} x_{ij} = 1 & \text{pour toute ville } v_i \\ z_i \leq 1 & \text{pour toute ville i} \\ \sum_{i=1}^{n} z_i = k & \\ d_{max} \geq d_{ij} \times x_{ij} \times z_j & \text{pour toute ville } i \text{ et pour tout secteur } j \end{cases}$$

$$x_{ij} \in \{0,1\}, \quad z_j \in \{0,1\}, \quad d_{max} \geq 0, \quad i \in I, \quad j \in J$$

Pour l'implémentation, nous rajoutons la variable de décision correspondant à d_{max} et modifions l'objectif :

```
# Variables xij qui valent 1 si les patients de la ville i sont affect s au
# secteur j, 0 sinon
x = m.addVars(n, n, vtype=GRB.BINARY, lb=0)

# Variables zj qui valent 1 si la ville j est un secteur, 0 sinon
z = m.addVars(n, vtype=GRB.BINARY, lb=0)

# Variable d qui est la distance maximale d'un habitant    son secteur pour un
# certain k, et qu'il faut minimiser
d_max = m.addVar(vtype=GRB.CONTINUOUS, lb=0)


# maj du modele pour integrer les nouvelles variables
m.update()


# On veut minimiser la distance maximale entre les villes et leur secteur
obj = LinExpr();
obj += d_max


# definition de l'objectif
m.setObjective(obj,GRB.MINIMIZE)
```

Listing 3 – Définition de l'objectif

Comme pour le premier programme d'optimisation des localisation des villes-secteurs, on demande d'abord à l'utilisateur de rentrer un nombre k de secteurs. Nous rajoutons la 6ème contrainte et testons notre programme. A partir de maintenant, lorsque nous parlerons de programme $P1$ il s'agira du programme de la question (2.1) qui cherche la localisation optimale des secteurs minimisant la distance moyenne, et nous parlerons de programme $P2$ pour désigner le programme de la question (2.2) qui cherche à minimiser la distance maximale parcourue.

Tests pour k = 3 : Nous testons $P1$ et $P2$ avec $\alpha = 0.5$, et obtenons exactement les mêmes secteurs (Nantes, Monpellier et Grenoble) et donc la même distance maximale parcourue, qui est de 347. Naturellement, la distance moyenne parcourue est la même et vaut 208.62.

Tests pour k = 4 : Nous testons $P1$ avec $\alpha = 0.5$ et obtenons les villes secteurs : Toulouse, Nantes, Reims et Toulon. La distance maximale parcourue est de 398 et la distance moyenne parcourue vaut 166.32.

```
Valeur de la fonction objectif pour alpha = 0.5 : 166.3152004488708

Solution optimale:
 [[1 0 0 0]
 [0 0 0 1]
 [0 1 0 0]
 [0 0 0 1]
 [0 0 1 0]
 [1 0 0 0]
 [0 0 1 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]
 [0 0 0 1]
 [0 0 1 0]
 [0 0 0 1]
 [0 0 1 0]
 [0 1 0 0]]

Les secteurs sont:
    Secteur Toulouse :  ['Toulouse', 'Bordeaux']
    Secteur Nantes :  ['Nantes', 'Rennes', 'Angers']
    Secteur Reims :  ['Strasbourg', 'Lille', 'Reims', 'LeHavre', 'Dijon']
    Secteur Toulon :  ['Nice', 'Montpellier', 'Saint-Étienne', 'Toulon', 'Grenoble']

 None
Distance maximale parcourue: 398
```

FIGURE 8 – Resultat obtenu $p1$, $k = 4$, $\alpha = 0.5$

En testant cette fois-ci avec $P2$ avec $\alpha = 0.5$, nous obtenons cette fois-ci les villes secteurs : Nantes, Montpellier, Reims et Toulon. La distance maximale parcourue *(i.e. la valeur de la fonction objectif pour $P2$)* est cette fois de 347, ce qui est inférieur à ce qu'on a trouvé pour $P2$ (398). Par contre la distance moyenne parcourue, elle, est supérieure au résultat de $P2$ et vaut 180.50 (contre 166.32).

```
Valeur de la fonction objectif pour alpha = 0.5 : 347.0

Solution optimale:
 [[0 1 0 0]
 [0 0 0 1]
 [1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [1 0 0 0]
 [0 0 1 0]
 [1 0 0 0]
 [0 0 1 0]
 [0 1 0 0]
 [0 0 0 1]
 [0 0 1 0]
 [0 1 0 0]
 [0 0 1 0]
 [1 0 0 0]]

Les secteurs sont:
    Secteur Nantes :  ['Nantes', 'Bordeaux', 'Rennes', 'Angers']
    Secteur Montpellier :  ['Toulouse', 'Montpellier', 'Saint-Étienne', 'Grenoble']
    Secteur Reims :  ['Strasbourg', 'Lille', 'Reims', 'LeHavre', 'Dijon']
    Secteur Toulon :  ['Nice', 'Toulon']
```

FIGURE 9 – Resultat obtenu $p2$, $k = 4$, $\alpha = 0.5$

Tests pour k = 5 : Nous testons $P1$ avec $\alpha = 0.5$ et obtenons les villes secteurs : Toulouse, Nice, Nantes, Reims et Saint-Etienne. La distance maximale parcourue est de 347 et la distance moyenne parcourue vaut 129.02.

En testant cette fois-ci avec $P2$ avec $\alpha = 0.5$, nous obtenons cette fois-ci les villes secteurs : Toulouse, Montpellier, Strasbourg, Rennes et Reims.

La distance maximale parcourue *(i.e. la valeur de la fonction objectif pour $P2$)* est cette fois de 326, ce qui est inférieur à ce qu'on a trouvé pour $P2$ (347). Par contre la distance moyenne parcourue, elle, est supérieure au résultat de $P2$ et 154.47 (contre 129.02).

3 Equilibrage des charges des unités de soin

On considère maintenant le scénario où 5 unités de soin ont été construites. Pour se fixer les idées et sans perte de généralité, on reprend les cinq unités de soins obtenues en 2.2 , c-à-d Toulouse, Montpellier, Strasbourg, Rennes et Reims.
Les solutions précédentes ne tenaient pas compte du facteur de surcharge d'un secteur, en pratique c'est une contrainte indispensable.

On considère donc le vecteur $p = (p_1, .., p_5)$ où p_j est le nombre de patients à traiter dans le secteur j et on suppose que

$$\sum_{i=1}^{5} p_i \leq 500$$

Montrons que ce problème peut être formulé comme un problème de transport.

3.1 Formalisation comme problème de transport

On impose avoir certains $P_i > 100$, en effet si $\forall i, P_i \leq 100$ les secteurs n'auront pas à transporter leur patients.

En considérant chaque secteur i où $p_i \geq 100$ comme producteur et les secteurs j où $p_j < 100$ comme consommateur, on obtient une partition $L \cup R$ des villes-secteurs formalisant un problème de transport. *(Un ensemble de secteurs à ressources et un ensemble de secteurs en demande)*

Puisque l'on impose des p_i supérieurs à 100, on est sûr de pouvoir atteindre ce formalisme pour n'importe quel vecteur p.

Considérons *(sans perte de généralité)* le vecteur $P = (140, 60, 120, 106, 74)$ pour Toulouse, Montpellier Strasbourg, Rennes et Reims respectivement.

Notons un surplus de 40 patients pour le secteur Toulouse, 20 pour Strasbourg et 6 pour Rennes. On traduit cela comme des ressources disponibles.
Tandis que pour Montpellier et Reims, on a 40 et 26 places disponibles que l'on considère comme des besoins. *(Pour ainsi réaliser l'équilibrage)*

Il ne reste qu'à déduire les quantité de ressources et besoins en comparant à 100. Cela est résumé dans le tableau ci-dessous :

Secteurs/ Unités	Montp	Reims	Qte dispo
Toulouse	242	812	40
Strasbourg	791	347	20
Rennes	894	483	6
Besoins	40	26	

FIGURE 10 – Tableau des coûts d'acheminement

La matrice b_{ij} indique le coût d'acheminement d'un patient d'un secteur i vers une unité j, ici les coûts sont caractérisés par la distance entre chaque ville-secteur.
On remarque que

$$\sum_{u \in L} |b_u| = \sum_{u \in R} |b_u| = 66 \qquad (*)$$

C'est donc bel est bien un problème de transport.

Il ne reste plus qu'à résoudre ce problème en utilisant l'un des algorithmes connus de la litérrature.

3.2 Résolution du problème de transport par la programmation linéaire

Nous avons choisi de résoudre le problème à l'aide de la programmation linéaire.
En effet, il s'agit de minimiser la quantité x_{ij} acheminée du secteur i vers l'unité j. C'est donc nos variables de décisions.

Le tableau ci-dessous clarifie la notation x_{ij} dans notre cas d'étude :

Secteurs/ Unités	*Montp*	*Reims*	*Qte dispo*
Toulouse	X_{11}	X_{12}	*40*
Strasbourg	X_{21}	X_{22}	*20*
Rennes	X_{31}	X_{32}	*6*
Besoins	*40*	*26*	

FIGURE 11 – Tableau des variables de décision

La fonction objectif à minimiser serait donc

$$\sum_{i=1}^{3}\sum_{j=1}^{2} b_{ij}x_{ij}$$

(avec b_{ij} la matrice des coûts associées)

Les contraites de notre programme linéaire ne sont qu'une traduction du tableau des besoins et quantités disponibles.

C'est à dire que les patients transférés du secteur Toulouse ne peuvent pas dépasser 40, ainsi :

$$x_{11} + x_{12} \leq 40$$

Par analogie pour les secteurs Strasbourg et Rennes respectivement :

$$x_{21} + x_{22} \leq 20$$

$$x_{31} + x_{32} \leq 6$$

Il ne reste plus qu'à contraindre le système pour ne pas transférer vers des secteurs jusqu'à leur surcharge (on obtiendrait à nouveau le même problème).
Ainsi, le secteur Montpellier ne pas recevoir plus de 40 patients, cependant nous savons par l'équation (*) que le problème admet une solution optimale et donc il existe une configuration où l'on peut transférer tout les patients.

Et donc le secteur Montpellier doit recevoir exactement 40 patients :

$$x_{11} + x_{21} + x_{31} = 40$$

Cas analogue pour Reims :

$$x_{12} + x_{22} + x_{32} = 26$$

Ainsi, nous présentons le programme linéaire obtenu :

$$min \quad 242x_{11} + 812x_{12} + 791x_{21} + 347x_{22} + 894x_{31} + 483x_{32}$$

$$s.c \quad \begin{cases} x_{11} + x_{12} \leq 40 \\ x_{21} + x_{22} \leq 20 \\ x_{31} + x_{32} \leq 6 \\ x_{11} + x_{21} + x_{31} = 40 \\ x_{12} + x_{22} + x_{32} = 26 \end{cases}$$

$$x_{ij} \geq 0, \quad i \in \{1,2,3\}, \quad j \in \{1,2\}$$

Nous avons ensuite procédé à une implémentation classique d'un problème de minimisation sous Gurobi, que voici ci-dessous :

Nous déclarons les variables de décisions (pour faciliter l'implémentation, nous utilisons les notations $x_1, x_2, .., x_6$

```
# declaration variables de decision
x = []
for i in colonnes:
    x.append(m.addVar(vtype=GRB.CONTINUOUS, lb=0, name="x%d" % (i+1)))

# maj du modele pour integrer les nouvelles variables
m.update()
```

FIGURE 12 – Déclaration des variables de décision

Ensuite, nous définissons la fonction objectif et les contraintes associées :

```
obj = LinExpr();
obj =0
for j in colonnes:
    obj += c[j] * x[j]

# definition de l'objectif
m.setObjective(obj,GRB.MINIMIZE)

# Definition des contraintes
for i in lignes:
    if i==3 or i==4:
        m.addConstr(quicksum(a[i][j]*x[j] for j in colonnes) == b[i], "Contrainte%d" % i)
    else:
        m.addConstr(quicksum(a[i][j]*x[j] for j in colonnes) <= b[i], "Contrainte%d" % i)
```

FIGURE 13 – Définition de la fonction objectif et des contraintes

Pour le vecteur P précédent, nous obtenons les résultats suivants :

```
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (win64)
Thread count: 2 physical cores, 2 logical processors, using up to 2 threads
Optimize a model with 5 rows, 6 columns and 12 nonzeros
Model fingerprint: 0x00be1e3e
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e+02, 9e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+00, 4e+01]
Presolve removed 1 rows and 1 columns
Presolve time: 0.02s
Presolved: 4 rows, 5 columns, 10 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.8702000e+04   5.989500e+00   0.000000e+00      0s
       2    1.9518000e+04   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.03 seconds
Optimal objective  1.951800000e+04

Solution optimale:
x1 = 40.0
x2 = 0.0
x3 = 0.0
x4 = 20.0
x5 = 0.0
x6 = 6.0

Valeur de la fonction objectif : 19518.0
```

FIGURE 14 – Resultat obtenus

Enfin, nous interprétons les résultats comme 40 patients à acheminer du secteur Toulouse à Montpellier, 20 patients de Strasbourg à Reims et 6 patients de Rennes à Reims. Le tout pour un coût de 19518 km parcourus

Notons que pour n'importe quel vecteur P donné, et en particulier dans notre cas, le résultat final sera l'équilibrage parfait des secteurs de soins. C-à-d :

$$\forall i \in [1..5], p_i = 100 si : \sum_{i=1}^{5} p_i = 500$$

Et un équilibrage admissible pour $\sum_{i=1}^{5} p_i \leq 500$.

Nous avons atteint notre objectif.