Optimisation du placement de points relais sur Manhattan utilisant un programme linéaire

Tristan Bilot, Nora Delfau, Romaric Faivre, Madushan Thambithurai, Enzar Salemi

February 13, 2021

1 Abstract

Il nous a été demandé, dans le contexte d'un projet de programmation linéaire, de minimiser la distance de manhattan entre un nombre donné de points relais et de potentiels clients voulant utiliser ces relais de la façon la plus optimisée possible. Autrement dit, nous souhaitons pour chaque client (a_l, b_l) minimiser la distance avec de potentiels points relais (x_i, y_i):

$$\sum_{i=1}^{p} \sum_{l=1}^{m} d((x_i, y_i), (a_l, b_l))$$

Dans un premier temps, nous avons décidé de nous focaliser sur la résolution du problème n'utilisant qu'un seul point relais à positionner pour deux clients. Trouver une approximation de la solution pour une paire de clients nous permettra ensuite d'envisager une solution plus poussée permettant de résoudre le problème pour n clients. Pour faciliter notre apprentissage de la programmation linéaire et afin de mieux comprendre et visualiser son utilisation, nous avons utilisé un solver en ligne permettant la résolution de programmes linéaires. C'est ainsi que nous avons implémenté pour la première fois une version minimisant la distance entre deux paires de clients et un point relais. Pour faciliter notre apprentissage de la programmation linéaire et afin de mieux comprendre et visualiser son utilisation, nous avons utilisé un solver en ligne permettant la résolution de programmes linéaires.

2 Solution pour un seul point relais

L'objectif premier fût de traduire le programme linéaire sous forme exploitable par l'ordinateur. C'est grâce à des matrices de coefficients que nous avons pu exprimer l'équation à minimiser ainsi que les inéquations représentant les contraintes. Il est important de noter qu'il est nécessaire d'appliquer les contraintes pour chaque paire de point, nous arrivons donc vite à des matrices de taille très grande. La matrice de contraintes aura un nombre de colonnes égale au nombre de variables total utilisé par le programme linéaire et un nombre de lignes égale au nombre de combinaisons de paires possibles multiplié par le nombre de contraintes établi pour chaque paire de point.

Une fois ces matrices construites, il est possible de les donner en input à un solver de programmes linéaires, nous avons utilisé scipy.optimize.linprog() de Python dans notre cas.

3 Solution générale pour p points relais

Nous arrivons à présent à résoudre le problème d'optimisation pour 1 point relais et 2 clients. Mais que ce passerait-il si nous utilisions cette implémentation pour n clients? Notre algorithme a pour but de trouver la distance minimale entre des paires de points, donc si nous appliquons cet algorithme sur les mêmes points, cela ne devrait-il pas donner le même résultat à chaque fois? En effet, si. C'est pour cela que nous devons réfléchir à une solution permettant de subdiviser le problème initial en sous-problèmes permettant de trouver plusieurs points relais qui seraient optimisés pour certaines zones de clients. Heureusement, il existe déjà des algorithmes performant pour cela. Nous avons utilisé la méthode de clusering des Kmeans souvent utilisée pour classifier des données en groupes. Cette méthode nous permettra de travailler non plus sur tout un nuage de points mais sur plusieurs sous-groupes sur lesquels appliquer notre algorithme d'optimisation. Nous pouvons synthétiser cette résolution en deux étapes:

- Traduction du problème linéaire en matrices d'équations et inéquations afin d'utiliser un solver.
- Subdiviser la population de clients M en sous-groupes (clusters) afin de pouvoir utiliser l'algorithme sur chacun de ces sous-groupes.

Une simulation 2D est proposée afin de visualiser la position des relais par rapport à la population.

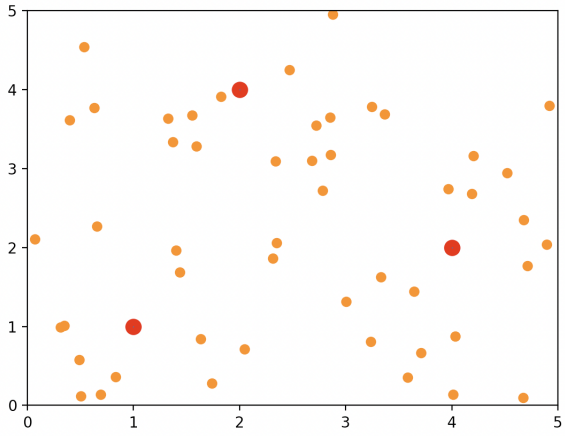

Figure 1: Simulation avec 50 clients et 3 relais

4 Possibles améliorations

Il serait intéressant d'utiliser un solver custom qui permettrait de limiter la taille des matrices de contraintes à construire. Par exemple donner en input une règle générale pour mettre en relation des contraintes à appliquer pour chaque paire de points. Ainsi on aurait une complexité spatiale constante. Un autre inconvénient rencontré est le fait qu'on ne peut pas avoir plus de points relais que de clients en utilisant l'algorithme simplex utilisé par linprog(). Encore une fois, cette limitation pourrait être contournée en implémentant soi-même une version du solver.