PICHARD Thomas
SCION Gaël

Projet de programmation linéaire en nombres entiers

« Cluster »

2ème année Polytech Paris-Saclay - Cycle Ingénieur

Année 2019-2020

Enseignant encadrant :
M. Nicolas DUPIN

Polytech Paris-Saclay
Maison de l'ingénieur - Bâtiment 620, Rue Louis de Broglie, 91405 Orsay, France
Téléphone : +33 (0)1 69 33 86 00 - fax : +33 (0)1 69 41 99 58 - www.polytech.u-psud.fr

I. Introduction

Le but de ce projet est d'appliquer les techniques exactes et heuristiques abordées durant le cours de Programmation Linéaire en Nombres Entiers sur un problème de clustering.

Ce projet exploratoire a pour but de réaliser un générateur de jeux de données permettant d'analyser les différences de résultats empiriques entre méthodes exactes et heuristiques en faisant varier un certain nombre de paramètres tels que la dimension, le nombre de clusters naturellement formés ou encore le nombre de points répartis dans les différents clusters naturels.

II. Générateur de données

Dans le cadre de ce projet, nous avons développé un générateur de données permettant de générer des jeux de données au format OPL avec l'aide du langage de programmation Java.

Afin de générer des données les plus réalistes possibles pour le problème de clustering, nous nous sommes inspirés d'un exemple concret en se demandant : comment des datacenters doivent-ils être répartis sur un territoire pour communiquer entre eux le plus efficacement possible ? En d'autres termes, il s'agit de minimiser la latence entre les sites et de déterminer dans quel datacenter doit se situer le serveur « arbitre » principal qui sera en charge de traiter les requêtes entrantes des clients.

Implantation des serveurs de Google Cloud à travers le monde en 2019

En étudiant la carte d'implantation des datacenters de Google Cloud à travers le monde, nous avons par exemple pu remarquer que, pour chaque région du monde où les services de l'entreprise sont proposés, les datacenters sont présents « en grappe » tout en étant quand même répartis sur le territoire à l'échelle d'un continent. Nous avons également remarqué qu'un ou plusieurs datacenters sont souvent plus centrés par rapport aux autres.

Ainsi, pour implémenter notre générateur, nous avons choisi de générer nos différents cluster en désignant pour chacun d'eux un point « arbitre » (pressenti comme étant un centre de clusters) dont les coordonnées sont choisies aléatoirement dans un environnement borné dans l'intervalle [0 ; 1000]. Les autres points sont ensuite placés aléatoirement autour d'un des points centraux en respectant des contraintes de distances maximale et minimale bornées dans l'intervalle [10 ; 100] et modélisées sous forme de rectangles. De cette manière, pour un point d'un cluster désigné, nous pouvons garantir que celui-ci ne sera ni trop près, ni trop loin de son centre de cluster pressenti.

Le respect de l'ensemble de ces contraintes nous permet de garantir que la génération du jeu de données de clusters soit différent pour chaque génération grâce aux composantes aléatoires mais aussi réalistes pour la problématique d'optimisation qui sera ensuite traitée par le solveur CPLEX d'une part et nos métaheuristiques d'autre part.

Conformément au cahier des charges, il est possible de paramétrer le générateur en précisant les paramètres :

- ***dimension*** : la dimension dans laquelle les points doivent être générés.
- ***nbPoints*** : le nombre total de points à générer incluant les centres de cluster.
- ***nbCluster*** : le nombre de cluster désirés.

Note : Pour chaque jeu de données généré, la liste des coordonnées des centres de cluster choisis pour la génération est également fournie sous forme de commentaire en fin de fichier.

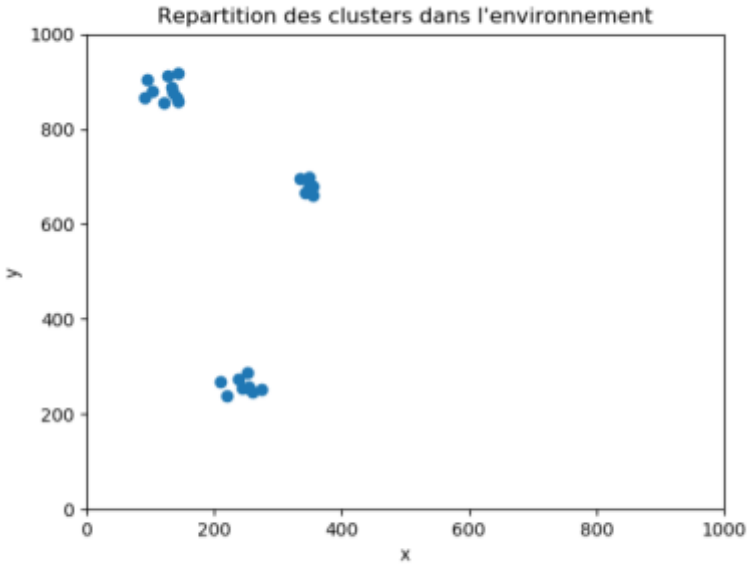
III. Affichage graphique des jeux de données

Lors de la phase de développement du générateur, nous nous sommes rendus compte qu'il n'était pas directement possible de vérifier la qualité des jeux de données générés par l'intermédiaire de notre générateur. Nous avons donc réalisé un script python dédié à l'affichage des jeux de données fourni par notre générateur à l'aide du module *pyplot* de la librairie *matplotlib*.

L'affichage des clusters sur un graphe simple nous a permis de visualiser rapidement la répartition des clusters et des points associés dans l'environnement et donc, de faire évoluer notre générateur en conséquence au cours du projet.

Cependant, par souci de simplicité et à cause des contraintes de temps imposées, nous n'avons traité que l'affichage des jeux de données pour la dimension 2.

L'appel au script d'affichage, dénommé « plotter », permet de naviguer de manière interactive dans l'environnement en 2D (des déplacements et des zooms sont possibles) mais aussi de générer un fichier image au format .png pour pouvoir visualiser l'implantation des clusters dans un jeu de données sans devoir ré-exécuter le programme. Ci-dessous, vous trouverez un exemple d'affichage pour le jeu de test *test09* dont les caractéristiques seront expliquées dans la suite du rapport.



Affichage du jeu de test *test09*

IV. Implémentation du problème

Afin de résoudre notre problématique qui se rapporte au choix des centres de clusters et à l'association des points restants à un cluster en minimisant le résultat de la fonction objectif, nous avons utilisé d'une part l'outil d'optimisation CPLEX, qui utilise des méthodes directes de résolutions, et d'autre part, nous avons développé notre propre solveur en JAVA qui utilise différentes méta-heuristiques.

A. Implémentation sous CPLEX

Concernant l'implémentation des différents modes de résolution sous CPLEX, nous avons créé différents modèles en prenant en compte la fonction objectif et les contraintes pour les problèmes de p-médian, k-medoids, p-centre discret et min-sum-k-radii.

```
7  int dimension = ...;
8  int nbPoints = ...;
9  int nbCluster = ...;
10
11 range pts = 1..nbPoints;
12 range dim = 1..dimension;
13
14 float points[pts][dim] = ...;
15
16 dvar boolean zn[pts][pts];
17 dvar boolean yn[pts];
18
19 float dist[p1 in pts][p2 in pts] = sqrt(sum(d in dim) pow(points[p1][d] - points[p2][d], 2));
20
21 minimize
22   sum(p1 in pts, p2 in pts) dist[p1][p2] * zn[p1][p2];
23
24 subject to{
25     forall(p1 in pts)
26       ctAffect:
27         sum(p2 in pts)
28           zn[p1][p2] >= 1;
29       ctRepre:
30         sum(p2 in pts)
31           yn[p2] == nbCluster;
32     forall(p1 in pts, p2 in pts)
33       ct:
34         zn[p1][p2] <= yn[p2];
35 }
```

Implémentation de la problématique p-median sous CPLEX

Ci-dessus, vous pouvez voir l'implémentation de p-median sous CPLEX. Cette dernière, tout comme l'ensemble des différents modes de résolution que nous avons implémentés, nous donne des résultats tout à fait cohérents et attendus dans des temps raisonnables.

B. Implémentation des méta-heuristiques

Afin de résoudre les problématiques du cas cluster avec une autre méthode que la méthode directe de CPLEX, nous avons implémenté différentes méta-heuristiques dans notre propre solveur en JAVA.

Ce dernier possède tout d'abord un parser qui va lire en entrée un fichier .dat de données du même format que CPLEX afin de stocker en mémoire la dimension des points, le nombre de points, le nombre de cluster et l'ensemble des points à traiter. Toutes ces informations sont stockées dans une structure objet permettant une manipulation simple des données afin de réaliser les calculs nécessaires à la recherche de solution.

Notre solveur possède une approche constructive et quatre approches perturbatives.

Pour l'approche constructive, il s'agit de la première étape de notre algorithme de résolution. En effet, lors de la phase de lancement, nous utilisons un algorithme glouton randomisé (GRASP) afin de générer de manière aléatoire une liste de boolean nous indiquant quels sont les points désignés comme centre de cluster.

Exemple de sortie de l'approche constructive

Pour ce qui est des approches perturbatives, nous avons implémenté quatre approches différentes :

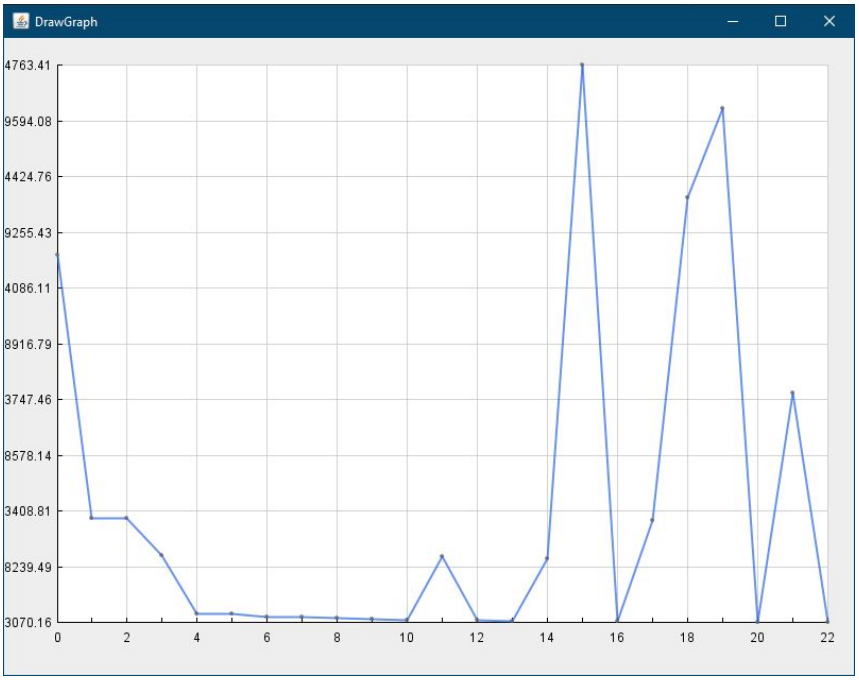
- Une méthode utilisant un algorithme aléatoire réalisant une diversification forte en tirant une toute nouvelle liste de centre de cluster.
- Une méthode d'intensification modifiant de manière aléatoire un seul centre de cluster parmi une liste de centres de cluster passée en entrée.
- Une méthode d'intensification modifiant de manière aléatoire la moitié des centres de cluster parmi une liste de centres de cluster passée en entrée.
- Une méthode d'intensification tabou remplaçant chaque centre de cluster par le point qui est le plus proche de ce dernier.

Afin de calculer une solution en utilisant notre approche constructive et nos différentes approches perturbatives, nous avons implémenté la méthode de Hill-Climbing avec comme fonction objectif la somme des distances au sein des clusters qui correspond à la problématique du p-médian.

Notre solveur est complètement ajustable dans le choix de son fonctionnement. En effet, vous avez la possibilité d'utiliser uniquement une de nos quatre méthodes perturbatives sur

l'entièreté du temps d'exécution (un algo glouton GRASP sur la totalité de la recherche) ou de choisir une méthode d'intensification et une méthode de diversification qui sera appelée de manière périodique durant la recherche de solution (un algo glouton + ILS).

Il est aussi possible d'afficher un graphique représentant l'état de la fonction objectif toutes les 10 000 itérations de l'algorithme et à chaque fois que l'algorithme trouve un meilleur choix dans les centres de cluster, afin de voir l'évolution au cours du temps de la progression de la recherche de solution.



Exemple de graphique que peut produire le solveur

V. Résolution du problème de cluster

A. Jeux de données utilisés

Nom du fichier	Dimension	NbPoints	NbCluster
clustering	2	49	3
test01	2	100	4
test02	2	250	12
test03	2	500	12
test04	3	200	6
test05	4	100	10
test06	4	100	5
test07	2	2000	100
test08	2	5000	200
test09	2	25	3
test10	3	25	3

Afin de comprendre et d'analyser au mieux le fonctionnement de nos métaheuristiques vis-à-vis de celui de la méthode exacte implémentée sur CPLEX, nous avons essayé de réaliser des jeux de données les plus variés possibles afin de couvrir une grande diversité de scénarios possibles.

Dans cette optique, nous avons générés des jeux de test dans différentes dimensions (2D, 3D et 4D) en faisant varier la densité de points dans l'environnement mais aussi le nombre de clusters.

Les fichiers *test01*, *test02* et *test03* nous permettent par exemple d'étudier l'incidence de l'augmentation du nombre de points et de clusters dans un environnement d'une dimension fixée (en l'occurrence 2D ici) tandis que les fichiers *test07* et *test08* permettent quant à eux d'effectuer un test de charge en définissant un nombre élevé de points et de clusters.
Les fichiers *test09* et *test10* ont vocation à analyser l'impact d'un changement de dimension des points et permettent, par conséquent, de se faire une idée de la performance des méthodes de résolution testées avec un nombre de points et de clusters identiques entre les deux fichiers.

B. Résultats et comparaison

1. Comparaison des résultats entre méthodes exactes

Nom du fichier / Mode de résolution sous CPLEX	Méthode directe CPLEX p-médian		Méthode directe CPLEX k-médoids		Méthode directe CPLEX p-centre discret		Méthode directe CPLEX min-sum-k-radii	
	Valeur fonction objectif	Temps (s)	Valeur fonction objectif	Temps (s)	Valeur fonction objectif	Temps (s)	Valeur fonction objectif	Temps (s)
clustering	2058.092534765	0.49	419990.640565828	0.49	291.139710256335	0.72	2058.09253476487	0.44
test01	2009.845472631	0.93	54392.1982338689	1.15	46.7452253745497	1.96	2009.84547263099	0.98
test02	4925.343360805	8.14	132393.401835385	9.21	55.7044813697172	32.80	4925.34336080539	10.24
test03	10300.408636385	84.52	280370.501782053	112.90	52.2725111367388	26 mins et 16.44s	10300.4086363848	62.72
test04	5254.934981534	4.61	169879.85158979	5.41	55.101898801591	9.83	5254.93498153428	5.12
test05	2737.931409276	0.80	96391.5531196287	1.02	60.1504160453988	1.97	2737.93140927635	0.84
test06	2890.179780345	0.94	97524.4478265781	0.86	54.0043021370355	1.70	2890.17978034511	0.91
test07	OUT_OF_MEMORY	N/A	OUT_OF_MEMORY	N/A	OUT_OF_MEMORY	N/A	OUT_OF_MEMORY	N/A
test08	OUT_OF_MEMORY	N/A	OUT_OF_MEMORY	N/A	OUT_OF_MEMORY	N/A	OUT_OF_MEMORY	N/A
test09	539.014871034	0.18	16122.2482828829	0.28	47.3587815516458	0.22	539.014871033624	0.17
test10	655.482429569	0.22	22777.145107586	0.19	50.52355219903	0.21	655.4824295569025	0.17

Etant donné que les différentes méthodes de résolution (dont les résultats sont présents dans le tableau ci-dessus) possèdent des fonctions objectifs différentes, il n'est pas possible de les comparer directement par leur valeur. C'est pourquoi nous nous proposons de comparer leur temps de convergence.

Ainsi, pour les différents cas considérés dans notre jeu de test, nous pouvons remarquer qu'il y a deux méthodes qui convergent plus rapidement parmi les quatre testées. Il s'agit de p-médian pour les tests *test01*, *test02*, *test04* et *test05* et min-sum-k-radii pour les tests *clustering*, *test03*, *test09* et *test10*.

En calculant le temps de convergence moyen de ces deux dernières méthodes et de k-médian pour notre jeu de test, nous obtenons les résultats suivants :

- p-médian : moy. 11.2033 s
- k-médian : moy. 14.6122 s
- min-sum-k-radii : moy. 9.0655 s

Sur ce jeu de test, c'est donc le modèle min-sum-k-radii qui semble être celui qui converge les plus rapidement pour trouver une solution satisfaisante mais avec une faible avance sur le modèle p-médian dont la moyenne est relativement proche.

2. Comparaison des résultats entre résolution directe par p-médian et les méta-heuristiques

Nom du fichier / Algo de résolution	Méthode directe CPLEX p-médian		Algo Glouton GRASP			Algo Glouton + ILS		
	Valeur fonction objectif	Temps (s)	Valeur fonction objectif	différence avec CPLEX	Temps (s)	Valeur fonction objectif	différence avec CPLEX	Temps (s)
clustering	2058.092534765	0.49	2058.333964789218	0.01%	0.673	2058.912497521501	0.04%	0.177
test01	2009.845472631	0.93	2014.036920624424	0.21%	60	2052.072695214837	2.10%	0.495
test02	4925.343360805	8.14	6178.685979976608	25.45 %	60	4950.207522949075	0.50%	36.681
test03	10300.408636385	84.52	14018.447571295603	36.10%	60	10733.50206571232	4.2%	60
test04	5254.934981534	4.61	5632.897093938999	7.19%	60	5271.63231829 95075	0.32%	2.157
test05	2737.931409276	0.80	3005.99387517 36636	9.79%	60	2767.05713310 00356	1.06%	0.701
test06	2890.17	0.94	3009.4	4.13%	60	2986.57	3.34%	0.870

	9780345		86700766483			1062634076		
test07	OUT_OF_MEMORY	N/A	62043.966971930924	N/A	60	58516.80590440102	N/A	60
test08	OUT_OF_MEMORY	N/A	128591.08636246089	N/A	60	130075.5882563744	N/A	60
test09	539.014871034	0.18	539.0148710336243	0%	0.106	539.0148710336243	0%	0.111
test10	655.482429569	0.22	655.482429569025	0%	0.217	655.482429569025	0%	0.102

Le tableau ci-dessus permet de comparer les différentes valeurs de la fonction objectif trouvée en solution pour le problème de cluster avec p-médian, déterminé par CPLEX, un algorithme glouton GRASP et un algorithme glouton + ILS pour les jeux de données que nous avons générés.

Nous pouvons constater que dans la majorité des cas, en comparaison de la méthode directe de CPLEX, c'est l'algorithme glouton + ILS qui trouve une solution qui est la plus proche de la meilleure solution trouvée par CPLEX. Dans certains cas, comme pour le fichier clustering, *test04*, *test09* et *test10*, cet algorithme trouve une solution presque voire complètement identique dans un temps plus court.

Il arrive aussi parfois que l'algorithme glouton GRASP trouve une meilleure solution par rapport au glouton + ILS comme, par exemple, avec le *test01* et le *test08*. Etant donné qu'il s'agit d'un algorithme ne réalisant que de la diversification très importante avec de l'aléatoire, ces résultats sont dus au hasard du déplacement sur la fonction objectif.

Plus globalement, nous avons également remarqué grâce au *test07 et test08* que plus la densité de points dans le jeu de test est élevée, plus le temps d'exécution devra être important pour converger vers une solution satisfaisante du fait du grand nombre de possibilités et de tests de combinaison à effectuer. La complexité des jeux de données en nombre de points et de clusters a donc un impact sur le niveau de difficulté pour résoudre le problème.

VI. Conclusion

Durant ce projet, nous avons pu constater qu'utiliser les méta-heuristiques nécessite un certain ajustement dans la quantité d'intensification ou de diversification afin de ne pas trop s'éloigner de la "meilleure solution" (celle fournie par CPLEX), mais que malgré tout, un simple algorithme glouton + ILS nous donne des résultats tout à fait satisfaisants voire même permet de fournir une solution que CPLEX ne peut pas fournir faute de mémoire suffisante sur la machine.

En conclusion, ce projet nous aura permis de compléter notre découverte de l'univers de la programmation par contraintes au travers d'un cas concret qu'est le problème de clustering. Nous avons pu nous familiariser avec CPLEX, qui est un des outils d'optimisation disponibles sur le marché et implémenter des méta-heuristiques dans notre propre solveur et de se familiariser avec leur fonctionnement.