

Résolution du SSCFLP

Dumez Dorian

December 3, 2016

Introduction

Nous allons ici nous intéresser aux problèmes de FLP. Le but étant de résoudre de manière exacte, grâce à un algorithme de branch & bound, le problème de SSCFLP. Le problème de FLP, facility location problem, consiste à trouver où ouvrir des services pour couvrir tous ses clients à moindre frais. La version qui nous intéresse, la single source avec capacité, rajoute des contraintes de capacité et de condition de livraison pour les clients. Le but de cette étude est d'utiliser la version sans capacité et celle sans contrainte de source unique comme relaxation de notre problème pour obtenir des bornes pour notre algorithme.

1 Problèmes étudiés

La première, et plus simple, version de notre problème est le UFLP. C'est à dire la localisation de service sans capacité. On se permet alors de supposer qu'un dépôt peut alimenter un nombre infini de clients, on ne quantifie donc pas la demande des clients.

Le modèle de programmation linéaire est :

$$\begin{aligned} \min z &= \sum_{j \in J} f_j x_j + \sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij} \\ \text{s.c.} \end{aligned}$$

$$\forall i \in I : \forall j \in J : y_{ij} - x_j \leq 0 \quad (1)$$

$$\forall i \in I : \sum_{j \in J} y_{ij} \geq 1 \quad (2)$$

$$\forall j \in J : x_j \in \{0; 1\} \wedge \forall i \in I : y_{ij} \in \{0; 1\}$$

Les x_j représentent le fait que la facilité j est ouverte et les y_{ij} le fait que le client i est appareillé avec le dépôt j . Les données sont les coûts d'ouvertures f_j et les coûts d'association client-dépôts c_{ij} .

La contrainte 1 exprime le fait que le dépôt j peut alimenter le client i seulement si il est ouvert. La deuxième exprime le fait que tous les clients doivent être couverts.

Dans une deuxième version du problème, CFLP, on prend en compte des capacités et des demandes. C'est à dire que les dépôts ont une capacité limitée et que les clients ont une demande quantifiée. De plus il faut remarquer que tous les dépôts n'ont pas forcément

la même capacité et les clients des quantités de demandes différentes, même si tout cela concerne toujours le même produit. Enfin il faut remarquer que l'on n'interdit pas qu'un client ait sa demande fragmenté entre plusieurs facilités.

Le modèle de programmation linéaire est :

$$\begin{aligned} \min z &= \sum_{j \in J} f_j x_j + \sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij} \\ \text{s.c.} \end{aligned}$$

$$\forall j \in J : \sum_{i \in I} y_{ij} \leq s_j x_j \quad (1)$$

$$\forall i \in I : \sum_{j \in J} y_{ij} = w_i \quad (2)$$

$$\forall j \in J : x_j \in \{0; 1\} \wedge \forall i \in I : y_{ij} \in \mathbb{R}^+$$

Maintenant que des quantités entre en jeux, les variables y_{ij} expriment la quantité fournie au client i par le dépôts j . Les données supplémentaires sont les capacités s_j des dépôts et les demandes w_i des clients. Il faut aussi noter que les coûts d'associations sont maintenant exprimé en prix par unité transporté.

La contrainte 1 représente alors le fait qu'un dépôts ne peut fournir plus que sa capacité, si il n'est pas ouvert sa capacité est nécessairement de 0. Et la seconde force la satisfaction complète de la demande de tous les clients.

La dernière version du problème, celle à laquelle on s'intéresse, le SSCFLP, ajoute la contrainte qu'un client ne peut être servi que par un unique dépôts. Le modèle de programmation linéaire est :

$$\begin{aligned} \min z &= \sum_{j \in J} f_j x_j + \sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij} \\ \text{s.c.} \end{aligned}$$

$$\forall j \in J : \sum_{i \in I} y_{ij} w_{ij} \leq s_j x_j \quad (1)$$

$$\forall i \in I : \sum_{j \in J} y_{ij} = 1 \quad (2)$$

$$\forall j \in J : x_j \in \{0; 1\} \wedge \forall i \in I : y_{ij} \in \{0; 1\}$$

Maintenant que les quantités demandés par les clients ne peuvent plus être fractionnés des booléens suffisent de nouveau pour les y_{ij} et exprime de nouveau l'association du client i à la facilité j . De plus aucune donnée supplémentaire n'est nécessaire.

La contrainte 1 exprime toujours le fait qu'un dépôts ne peut livrer plus que sa capacité. Et la contrainte 2 impose le fait que chaque client doit être livré par un unique dépôts.

On remarque alors que le programme de l'UFLP peut être re-écrit si les coûts d'association sont non-négatif :

$$\min z = \sum_{j \in J} f_j x_j + \sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij}$$

s.c.

$$\forall i \in I : \forall j \in J : y_{ij} - x_j \leq 0 \quad (1)$$

$$\forall i \in I : \sum_{j \in J} y_{ij} = 1 \quad (2)$$

$$\forall j \in J : x_j \in \{0; 1\} \wedge \forall i \in I : y_{ij} \in \{0; 1\}$$

En effet, la fonction d'objectif étant en minimisation, la solution optimale ne couvrira jamais, sauf dégénérescence causée par coûts d'appareillages nul, un client 2 fois. On peut alors dire que la contrainte est satisfaite si et seulement si chaque client est couvert une unique fois.

De plus, on peut re-écrire la contrainte qui force un dépôt utilisé à être ouvert pour faire apparaître plus clairement la relaxation du SSCFLP en UFLP :

$$\min z = \sum_{j \in J} f_j x_j + \sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij}$$

s.c.

$$\forall j \in J : \sum_{i \in I} y_{ij} w_{ij} \leq M x_j \quad (1)$$

$$\forall i \in I : \sum_{j \in J} y_{ij} = 1 \quad (2)$$

$$\forall j \in J : x_j \in \{0; 1\} \wedge \forall i \in I : y_{ij} \in \{0; 1\}$$

En effet l'usage d'un M (une valeur devant laquelle toute demande est négligeable) permet d'oublier la contrainte de capacité. La relaxation du SSCFLP en UFLP s'effectue donc en relaxant la première contrainte. La solution obtenue avec ce programme respectera la condition de source unique mais pas celles des capacités.

On remarque que on peut aussi écrire CFLP sous une autre forme, qui rend évident le fait que le CFLP est une relaxation du SSCFLP :

$$\min z = \sum_{j \in J} f_j x_j + \sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij} w_{ij}$$

s.c.

$$\forall j \in J : \sum_{i \in I} y_{ij} w_{ij} \leq s_j x_j \quad (1)$$

$$\forall i \in I : \sum_{j \in J} y_{ij} = 1 \quad (2)$$

$$\forall j \in J : x_j \in \{0; 1\} \wedge \forall i \in I : y_{ij} \in [0; 1]$$

Les y_{ij} représente alors la part de la demande de i qui est assumé par la facilité j . On remarque alors que si l'on prend le programme linéaire du SSCFLP et qu'on y exprime les coûts d'associations en prix par unité alors on a exactement le même programme à l'exception de la relaxation linéaire des y_{ij} . La solution obtenue avec ce programme respectera les contraintes de capacités mais pas celle de source unique.

2 Instances choisies

Toutes les instances viennent du site <http://www-eio.upc.es/elena/sscplp/index.html>.

La première instance, p1, est une petite instance normale, elle ne comprend que 20 clients et 10 possibilités de facilités. Les coûts d'ouverture sont relativement grand par rapport aux coûts de connexions et les capacités sont aussi grandes par rapport aux demandes.

L'instance p2, petite, possède des coût de connexions élevé par rapport aux coûts d'ouvertures.

L'instance p3 est aussi de petite taille mais présente des facilités avec des coûts d'ouvertures très élevés.

L'instance p6, toujours petite, mais nous propose des facilités peu chère mais avec de petites capacités par rapport aux demandes.

Suivant le même schéma on effectue une montée en charge avec les instances p7(normale), p9(petite et pas chère), p10(facilité chères) et p13(coûts de connexions élevé). En effet celles ci on 30 clients pour 15 possibilités de facilités.

Toujours selon le même principe on passe à des instances à 50 clients et 20 sites possibles. On utilise les instances p26(normale), p33(coûts d'ouvertures élevés), p31(facilités peux chères) et p30 (coûts de connexions élevés).

3 Résolution avec des solveur génériques

Premièrement il faut noter que j'ai opté pour d'autres solveurs car GLPK n'y arrivait pas. En effet des tests sur de toutes petites instances on montré que le modèle et l'appel était correct mais GLPK ne parvenait pas à résoudre le problème. Sur la première des instances les plus petites il y parvenais en 25 minutes et ne parvenait pas à résoudre la seconde à cause d'une trop grande empreinte mémoire.

Je me suis tourné vers les 2 autres solveurs interfacé avec JuMP qui propose la résolution de problèmes MIP : CPLEX et Mosek.

nbClient	nbDepos	normale	++connexions	++ouverture	- - capacités
20	10	0.22	0.14	0.91	0.23
30	15	1.08	1.44	1.28	0.40
50	20	2.24	3.25	9.78	5.25

Table 1: temps de résolution par CPLEX

nbClient	nbDepos	normale	++connexions	++ouverture	- - capacités
20	10	2.02	2.61	10.6	0.85
30	15	10.14	3.52	23.31	5.53
50	20	43.32	130.37	long	long

Table 2: temps de résolution par Mosek

On prend comme étalon les instances dites normale, ne présentant à priori aucune particularité. On peut alors supposer que les instances avec des grand coûts de connexions ou d'ouverture sont plus compliqué. Par contre on ne peut rien dire sur les instance concernant les petits dépôts pas cher.

Je n'ai pas inclus les solutions fournie dans le dossier de rendu car CPLEX et Mosek donnent parfois des valeurs différente de 0 ou 1 à des variables binaires (elle le sont bien selon l'affichage de la modélisation). Mais ces imprécisions sont de l'ordre de 10^{-10} ou telles que des -0 .

4 Bornes

4.1 Borne primale

Pour obtenir une borne primale nous avons utilisé l'heuristique de construction de Delmaire. Cette dernière permet d'obtenir une solution réalisable non triviale. Elle se base sur l'idée que les services à ouvrir doivent avoir le coût par client le plus faible et que les clients doivent êtres associés au facilitées avec lequel ils coûtent le moins chers.

instance	type	% de coût supplémentaire	temps de calcul
p1	normale	6.65	0.001
p2	++connexion	11.03	0.001
p3	++ouverture	14.95	0.001
p6	- - capacité	14.85	0.001
p7	normale	6.18	0.009
p13	++connexion	13.51	0.006
p10	++ouverture	11.60	0.007
p9	- - capacité	10.20	0.007
p26	normale	13.24	0.028
p30	++connexion	8.85	0.030
p33	++ouverture	15.31	0.031
p31	- - capacité	8.28	0.027

Table 3: Tests de l'heuristique de Delmaire

Premièrement on observe que les temps d'exécution sont petit, et n'augmente pas trop avec la taille des instances. De plus le type d'instance n'impacte que très peu sur le temps de calcul. On peu donc envisager d'utiliser cette heuristique de construction de

manière très agressive durant notre branch and bound.

De manière grossière l'heuristique semble mieux fonctionner sur les instances normales et moins sur celle à coûts d'ouvertures élevés. Mais dans tous les cas la solution fournit nous donne une borne primale correcte.

4.2 Borne duale

Comme relaxation continue nous avons utilise la modélisation du CFLP auquel on a relâché la contrainte de type sur les variables désignant l'ouverture des dépôts. C'est à dire que l'on peu ouvrir un dépôts à moitié.

instance	type	% de coût inférieur	temps GLPK	temps Mosek	temps CPLEX
p1	normale	10.95	0.0008	0.0025	0.0010
p2	++connexion	6.18	0.0009	0.0030	0.0008
p3	++ouverture	6.47	0.0009	0.0024	0.0007
p6	- - capacité	8.86	0.0009	0.0020	0.0008
p7	normale	5.47	0.0016	0.0036	0.0013
p13	++connexion	5.24	0.0020	0.0028	0.0012
p10	++ouverture	1.26	0.0022	0.0032	0.0013
p9	- - capacité	7.49	0.0024	0.0035	0.0012
p26	normale	5.66	0.0056	0.0053	0.0022
p30	++connexion	5.46	0.0060	0.0052	0.0022
p33	++ouverture	2.28	0.0061	0.0062	0.0023
p31	- - capacité	6.80	0.0056	0.0054	0.0021

Table 4: Expérimentations relaxation

Premièrement on peu encore noté que les temps d'exécutions restent petit donc on pourra aussi utiliser ce procédé de manière fréquente dans le branch and bound.

La qualité de la relaxation diffère significativement entre les types d'instances donc on peu les étudiés séparément :

- les instances normales font parties de celle sur lequel la relaxation est de moins bonne qualité. Quand on observe les solutions trouve on remarque que la solution relâché a tendance a ne pas ouvrir un dépôts au profit d'autre partiellement ouvert. Et au contraire la relaxation profite peu de la distribution de la demande sur deux dépôts. Au vu de ces résultat la possibilité de relâcher de SSCFLP en CFLP pourrai corriger ce problème si il ne se déplace pas vers la distribution des demande entre les dépôts.
- sur les instances à fort coût de connexion la relaxation est de qualité moyenne. Mais en réalité si on observe les résultats, ils ont la même forme que pour les instances normales. Sauf qu'ici la part des coûts engendrés par les connexions est plus importante donc, vu que l'on compare relativement, la relaxation semble de meilleure qualité.

- sur les instances présentant de grands coûts d'ouvertures la relaxation est généralement de très bonne qualité. La solution proposé utilise alors plus la distribution de la demande entre plusieurs dépôts, et les services ouvert sont à peu près les même mais pas forcément ouvert en entier. Mais étant donne que la plupart des coûts des générés par l'ouverture des dépôts ce doit surtout être la relaxation sur l'ouverture des dépôts qui diminue la qualité. Mais la différence relative reste faible car les solutions optimales restent très chères.
- enfin les instances avec des facilités à faibles capacités sont mal approximé par relaxation. Pour ces instances on retrouve les même problème qu'avec les instances normales.

Au vu de ces remarque j'ai expérimenté la relaxation en CFLP :

instance	type	% de coût inférieur	temps GLPK	temps Mosek	temps CPLEX
p1	normale	10.08	0.0019	0.0174	0.0078
p2	++connexion	4.48	0.0046	0.0114	0.0337
p3	++ouverture	1.10	0.0051	0.0227	0.0318
p6	- - capacité	4.24	0.0131	0.0347	0.0131
p7	normale	4.13	0.0063	0.0645	0.0696
p13	++connexion	1.59	0.0505	0.0815	0.0397
p10	++ouverture	0.50	0.0056	0.0571	0.0175
p9	- - capacité	5.01	0.0165	0.0581	0.0371
p26	normale	2.71	0.0743	0.1034	0.0779
p30	++connexion	3.50	0.1169	0.2074	0.1007
p33	++ouverture	0.55	0.0191	0.0571	0.0222
p31	- - capacité	5.03	0.0354	0.0616	0.0478

Table 5: Expérimentations CFLP

On obtient effectivement des bornes d'une très bonne qualité, mais en un temps significativement plus long, ce qui limite l'agressivité avec laquelle on pourra utiliser cette relaxation.

J'ai donc essaye la relaxation en UFLP pour avoir un compromis entre qualité et temps de calcul. Mais la qualité de la relaxation devient pitoyable, de l'ordre de 60-80%.

Dans l'article de Kaj Holmberg & co il est fait référence de l'amélioration de la qualité de la borne duale obtenue par relaxation linéaire en ajoutant des contraintes redondantes. J'ai donc testé la relaxation du SSCFLP en cette modélisation, ou la contrainte 3 a été ajouté :

$$\min z = \sum_{j \in J} f_j x_j + \sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij}$$

S.C.

$$\forall j \in J : \sum_{i \in I} y_{ij} w_{ij} \leq s_j x_j \quad (1)$$

$$\forall i \in I : \sum_{j \in J} y_{ij} = 1 \quad (2)$$

$$\begin{aligned} \forall i \in I : \forall j \in J : y_{ij} - x_j &\leq 0 \\ \forall j \in J : x_j \in [0; 1] \wedge \forall i \in I : y_{ij} &\in [0; 1] \end{aligned} \quad (3)$$

instance	type	% de coût inférieur	temps GLPK	temps Mosek	temps CPLEX
p1	normale	10.52	0.0027	0.0079	0.0011
p2	++connexion	6.01	0.0025	0.0086	0.0010
p3	++ouverture	6.35	0.0028	0.0092	0.0010
p6	- - capacité	8.16	0.0025	0.0077	0.0009
p7	normale	4.94	0.0069	0.0150	0.0021
p13	++connexion	4.85	0.0088	0.0136	0.0024
p10	++ouverture	1.25	0.0060	0.0150	0.0019
p9	- - capacité	7.29	0.0070	0.0111	0.0016
p26	normale	5.29	0.0246	0.0255	0.0043
p30	++connexion	5.30	0.0257	0.0240	0.0038
p33	++ouverture	2.27	0.0229	0.0286	0.0046
p31	- - capacité	5.58	0.0280	0.0454	0.0052

Table 6: Expérimentations relaxation linéaire sur-contraint

La qualité est toujours meilleure mais généralement la différence est très petite. Par contre la résolution est plus lente mais cela reste raisonnable donc on peut envisager l'utilisation de ces contraintes.

Au contraire l'ajout de la contrainte propose par sune, disant que la somme des capacité doit être supérieure à la demande totale, n'améliore rien et ralenti tout.

Bibliographie

- Sources des instances : <http://www-eio.upc.es/elena/sscplp/index.html>.
- Pour l'heuristique de Delmaire : "Comparing New Heuristics for the Pure Integer Capacitated Plant Location Problem" par Hugues Delmaire, Maruja Ortega, Juan A. Diaz, Elena Fernandez
- "An exact algorithm for the capacitated facility location problems with single sourcing" par Kaj Holmberg, Mikael Ronnqvist, Di Yuan