Rapport intermédiaire du projet sur la résolution exacte du SSCFLP et ses simplifications

Jocelin Cailloux

December 29, 2016

Contents

1 Introduction

Ce projet pédagogique réalisé dans le cadre de l'unité d'enseignement de la programmation à variables entières à l'Université de Nantes a pour objectif de nous faire implémenter un algorithme de branch & bound sur une variante du problème de locations de facilités. J'ai choisi la variante SSCFLP (Single Source Capacited Facility Location Problem). La plus grande difficulté rencontrée à été de débugger mon algorithme de branch & bound. En effet, même les instances les plus simples sont très longues (vraiment très longues) à résoudre et j'ai longtemps pensé que mon algorithme cyclait sans réussir à comprendre pourquoi.

2 Les problèmes

Le Facility Location Problem est un problème qui consiste à chercher où installer un certain type de facilités afin d'en minimiser le coût d'installation et de maintenance. Il existe une variante du problème pour laquelle on cherche à couvrir un maximum de personnes à moindre coût. Ici, le problème est simplifié, on doit couvrir tout le monde et on cherche le coût minimum. De plus, les facilités ne peuvent pas être installées à n'impote quel emplacement mais sur une liste d'emplacements définie, ce qui rend le problème discret.

Pour les modélisations, nous utilisons les variables suivantes :

- W_j la quantité nécessitée par la demande j
- S_i la quantité maximale que peut fournir la facilité i
- F_j le coût de construction d'une facilité à l'emplacement j
- C_{ij} le coût de liaison entre la demande i et la facilité j
- $X_i = \begin{cases} 1, & \text{si une facilité est installée à l'emplacement i} \\ 0, & \text{sinon} \end{cases}$
- $Y_{ij} = \begin{cases} 1, & \text{si la demande j est assignée à la facilité i} \\ 0, & \text{sinon} \end{cases}$

2.1 UFLP

La variante UFLP du problème est la plus simple traitée ici. Dans cette variante, les facilités peuvent satisfaire tous les besoins d'autant de demandes que nécessaire.

Nous obtenons donc la modélisation suivante :

$$min(z) = \sum_{i \in I}(X_i * F_i + \sum_{j \in J}(Y_{ij} * C_{ij})) \tag{1}$$

Sous la contrainte suivante :

- Une demande doit être liée à une et une seule facilité

$$\sum_{i \in I} X_i * Y_{ij} = 1, \quad j \in J \tag{2}$$

2.2 CFLP

La vriante CFLP est la plus permissive. Dans cette variante, une demande peut-être complétée par plusieurs facilitées.

Pour ce faire, la variable Y_{ij} ne stocke plus 0 ou 1 mais une proportion (entre 0 et 1) de la demande W_j. De plus, nous n'avons plus la contrainte limitant à 1 le nombre de facilités reliées. Cette contrainte est remplacée par la contrainte vérifiant que la demande est entièrement complétée. La fonction objectif est la même.

Nous optenons donc la modélisation suivante :

$$min(z) = \sum_{i \in I}(X_i * F_i + \sum_{j \in J}(Y_{ij} * C_{ij})) \tag{3}$$

- Une demande doit être entièrement complétée par les facilités auxquelles elle est reliée

$$\sum_{i \in I} X_i * Y_{ij} = 1, \quad j \in J \tag{4}$$

- Une facilité ne peut pas fournir plus que sa capacité maximale

$$\sum_{j \in J} X_i * Y_{ij} * W_j \leq S_i, \quad i \in I \tag{5}$$

2.3 SSCFLP

Cette variante est la plus compliquée des variantes traitées ici. Dans cette variante, une demande doit être liée à une et une seule facilité. Cependant, contrairement à la variante UFLP, chaque facilité à une quantité maximale qu'elle peut fournir.

Pour ce faire, nous reprenons les variables telles qu'elles sont décrites au début du chapitre. Nous n'avons plus besoin de vérifier qu'une demande est entièrement complétée par les facilités auxquelles elle est reliée. En effet, il n'y a qu'une seule facilité liée à une demande, nous savons donc que la facilité complète toute la demande. Il faut Toutefois vérifier que la facilité a la capacité de fournir toutes les demandes auxquelles elle est liée. La fonction objectif reste la même.

Nous optenons donc la modélisation suivante :

$$min(z) = \sum_{i \in I}(X_i * F_i + \sum_{j \in J}(Y_{ij} * C_{ij})) \tag{6}$$

- Une demande doit être liée à une et une seule facilité

$$\sum_{i \in I} X_i * Y_{ij} = 1, \quad j \in J \tag{7}$$

- Une facilité ne peut pas fournir plus que sa capacité maximale

$$\sum_{j \in J} X_i * Y_{ij} * W_j \leq S_i, \quad i \in I \tag{8}$$

3 Les instances

Les instances que nous utiliserons au cours de ce projet ont été trouvé à l'adresse suivante : `http://www-eio.upc.es/~elena/sscplp/index.html`

Nous utiliserons 4 petites instances de taille 20X10, les instances p1, p3, p4 et p5. Ces petites instances permettront de tester les algorithmes. Ils permettront de dérouler les algorithmes de manière relativement simple et ainsi aideront au débogage. Ils permettront ainsi aussi de tester et comparer certaines astuces qui pourront être tentées pour améliorer l'efficacité de la résolution. Les instances p1, p3, p4 et p5 semblent pour cela correspondre à une bonne combinaison d'instances. En effet, elles ont toutes des caractéristiques particulières. p1 possède des coûts de fabrication des facilités relativement faibles, p3 possède des capacités de facilités plutôt faibles, p4 possède des coûts de fabrication élevés et p5 a tendance à avoir des facilités dont les capacités sont elevées.

Ensuite, nous utiliserons des instances un peu plus grande, 30X15, ces instances permettront d'observer comment ralentit l'algorithme en fonction de la taille des données et d'améliorer ce qui prend du temps. Afin d'avoir des comparaisons plus pertinentes, nous devrions peut-être utiliser des instances dont les capacités des facilités sont plus ou moins grands. Je pense en effet que cette caractéristique d'une instance est intéressante à étudier. C'est pourquoi nous utiliserons les instances p11 et p14 qui ont respectivement des capacités faibles et élevées.

Nous utiliserons seulement une instance de chaque taille pour les plus grandes. Il s'agît surtout de tester le comportement de l'algorithme en augmentant de plus en plus la taille des données. Les instances p25, p32, p35 et p45 sont choisies de manière arbitraire. Elles sont respectivement de tailles 40X20, 50X20, 60X30 et 75X30.

Une fois l'algorithme stabilisé, ou pour tester le comportement dans un contexte spécifique, d'autres instances pourront être utilisées. Il est possible que ces instances choisies soient amenées à changer si les tests des premières instances indiquent que la variation de certaines caractéristiques sont plus pertinentes à tester que prévu.

4 Les solutions et les bornes des instances

4.1 Solution de p1 :

- Lien ver l'instance
- Temps de résolution par GLPK : 8 minutes et 55 secondes
- Temps de résolution par le Branch & Bound : 1 heure 38 minutes et 49 secondes
- Valeur de la fonction objectif : 2014
- Valeur de la borne primale : 2518
- Valeur de la borne duale : 1811

4.2 Solution de p3 :

- Lien vers l'instance
- Temps de résolution par GLPK : 15 minutes et 16 secondes
- Temps de résolution par le Branch & Bound : 5 minutes et 10 secondes
- Valeur de la fonction objectif : 6051
- Valeur de la borne primale : 7541
- Valeur de la borne duale : 5985

4.3 Solution de p4 :

- Lien vers l'instance
- Temps de résolution par GLPK : 19 secondes
- Temps de résolution par le Branch & Bound : Non résolu
- Valeur de la fonction objectif : 7168
- Valeur de la borne primale : 10068
- Valeur de la borne duale : 6984

4.4 Solution de p5 :

- Lien vers l'instance :
- Temps de résolution par GLPK : 22 secondes
- Temps de résolution par le Branch & Bound : 1 heure 28 minutes et 49 secondes
- Valeur de la fonction objectif : 4551
- Valeur de la borne primale : 5009
- Valeur de la borne duale : 4418

4.5 Solution de p11 :

- Lien vers l'instance :
- Temps de résolution par GLPK : Non résolu
- Temps de résolution par le Branch & Bound : Non résolu
- Valeur de la fonction objectif : Inconnue
- Valeur de la borne primale : 4241
- Valeur de la borne duale : 3310

4.6 Solution de p14 :

- Lien vers l'instance :
- Temps de résolution par GLPK : 3 minutes et 39 secondes
- Temps de résolution par le Branch & Bound : Non résolu
- Valeur de la fonction objectif : 5965
- Valeur de la borne primale : 7124
- Valeur de la borne duale : 5793

4.7 Solution de p25 :

- Lien vers l'instance :
- Temps de résolution par GLPK : 73 millisecondes
- Temps de résolution par le Branch & Bound : Non résolu
- Valeur de la fonction objectif : 8947
- Valeur de la borne primale : 12408
- Valeur de la borne duale : 8943

4.8 Solution de p32 :

- Lien vers l'instance :
- Temps de résolution par GLPK : 9 secondes
- Temps de résolution par le Branch & Bound : 1 minute et 48 secondes
- Valeur de la fonction objectif : 9881
- Valeur de la borne primale : 13380
- Valeur de la borne duale : 9856

4.9 Solution de p35 :

- Lien vers l'instance :
- Temps de résolution par GLPK : 12 minutes et 38 secondes
- Temps de résolution par le Branch & Bound : Non résolu
- Valeur de la fonction objectif : 5456
- Valeur de la borne primale : 10024
- Valeur de la borne duale : 5432

4.10 Solution de p45 :

- Lien vers l'instance :
- Temps de résolution par GLPK : 3 secondes
- Temps de résolution par le Branch & Bound : Non résolu
- Valeur de la fonction objectif : 17676
- Valeur de la borne primale : 27473
- Valeur de la borne duale : 17664

5 Branch & Bound

J'ai choisi d'effectuer mon branch & bound uniquement sur les variables définissant les liaisons entre une facilité et un client. En effet, si une liaison existe entre la facilité i et le client j, alors la facilité i est systématiquement ouverte et toutes les autres liaisons avec j sont fermées. Effectuer un branch & bound sur l'ouverture des facilités ne semble pas nécessaire. Néanmoins, je pense qu'il est possible d'améliorer grandement la rapidité de l'algorithme en effectuant des relaxations avec un branch & bound sur les ouvertures de facilités puis un branch & bound sur les liaisons. Ceci est une bonne piste d'amélioration pour mon algorithme.

5.1 L'algorithme

La profondeur vaut 0 au début. Tant que la profondeur de l'arbre n'est pas négative, à chaque noeud du branch & bound :

1. Si la modélisation n'a pas de solution ou si la solution relâchée est moins bonne que la borne primale, la profondeur diminue de 1.
2. Sinon si la solution est entière, alors elle est la meilleure solution trouvée jusqu'à maintenant et est enregistrée. la borne primale est mise à jour.
3. Sinon, la colonne à modifier est sélectionnée.
   (a) Si la colonne est libre, alors elle est fixée à 1 et la profondeur de l'arbre augmente de 1. Une relaxation linéaire est effectuée..
   (b) Sinon si la colonne est fixée à 1, alors elle est fixée à 0 et la profondeur de l'arbre augmente de 1. Une relaxation linéaire est effectuée..
   (c) Sinon si la colonne est fixée à 0, alors il n'y a pas de solution dans cette direction, elle est libérée et la profondeur de l'arbre diminue de 1..

5.2 Les astuces d'optimisation

Tout d'abord, une optimisation simple et évident concerne la modélisation. En effet, je n'ajoute pas de contrainte à la modélisation, j'indique simplement à GLPK les bornes des variables. Ainsi, je peux fixée une variable à 1, 0 ou la libérer en O(1) tout comme obtenir ses bornes courantes est en O(1). Je n'ai donc pas de structure pour la remodélisation ni pour retenir les bornes puisque GLPK le permet déjà.
Ensuite, une seconde optimisation moins naturelle est de fixer l'ouverture des facilités si celle-ci ont au moins un client relié par le branch & bound ou de fixer la fermeture d'une

facilité si celle-ci a toutes ses variables de liaison fixée à 0. Cette amélioration qui semble anodine vu le fonctionnement du simplexe permet pourtant un gain de vitesse de presque 4%.

6 Conclusion

Au terme de cet exercice, j'ai réussi à implémenter un algorithme de branch & bound fonctionnel qui résout le SSCFLP. Toutefois, nous remarquons qu'un branch & bound classique est loin de suffir pour ce type de problème. En effet, les instances sont très longues à résoudre. Je pense qu'il faudrait travailler plus finement un branch and bound sur les ouvertures de facilité et le faire fonctionner en coopération avec le branch & bound sur les liaisons avec les facilités. Ce projet m'a permis de prendre conscience de la difficulté de stopper rapidement le parcours d'une mauvaise branche et cela se ressent sur le caractère imprévisible de la durée d'exécution de la résolution en fonction de l'instance. En effet, si souvent d'excellentes solutions sont trouvées rapidement, parfois la meilleure, le branch & bound est très souvent lent à terminer.