Luc LAPORTE

Titre

Encadrant :
Zacharie ALES

Table des matières

Chapitre 1

Introduction

1.1 Problème de set-cover

1.1.1 Définition du problème

Soit un ensemble U, un sous-ensemble S de l'ensemble des parties de U et un entier k. On souhaite savoir s'il existe un sous-ensemble T de S de taille k tel que tous les éléments de U sont couverts par l'union des sous-ensembles de T, c'est-à-dire :

$$\forall e \in U,\ e \in \bigcap_{t \in T} t$$

1.1.2 Modélisation sous la forme d'un programme linéaire

Résoudre ce problème revient à résoudre le programme linéaire suivant :

$$\begin{cases} min & \sum_{s \in S} x_s \\ tq & \sum_{s,\ e \in S} x_s \geq 1,\ e \in U \\ & x_s \in \{0,1\},\ s \in S \end{cases}$$

La fonction coût à optimiser correspond à la taille du sous-ensemble T que l'on cherche à minimiser. La première contrainte impose que tous les éléments de U soient couverts par l'union des sous-ensembles de T.

1.1.3 Application du problème de set-cover

Le problème de set-cover trouve une application dans la résolution du problème de p-centres. Ce problème est le suivant : on considère un ensemble de points, et un ensemble de positions candidates. On cherche à placer p-centres sur ces positions candidates afin de minimiser la plus grande distance entre un point et son centre le plus proche. Ce problème correspond par exemple

au choix d'un terrain pour la mise en place d'entrepôts ou de magasins afin d'être le plus proche possible des clients.

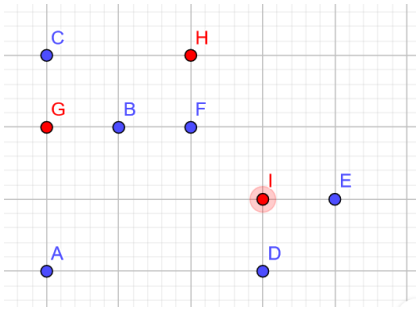

Fig. 1.1 – Exemple de problème de 2-centres

La figure ci-dessus est un exemple de problème de 2-centres. Les points bleus correspondent aux clients, tandis que les points rouges correspondent aux positions possibles ou placer les deux centres.
Afin de résoudre ce problème, on exprime les distances à chacun des positions candidates sous la forme d'un tableau (voir exemple en table 1)

	G	H	I
A	2	$\sqrt{13}$	$\sqrt{10}$
B	1	$\sqrt{2}$	$\sqrt{5}$
C	1	2	$\sqrt{13}$
D	$\sqrt{13}$	$\sqrt{10}$	1
E	$\sqrt{17}$	$2\sqrt{2}$	1
F	2	1	$\sqrt{2}$

On ordonne ensuite les distances sans répétition, et on met en place un dichotomie. A chaque étape, on considère une distance d et on cherche une solution au problème de set-cover avec k=2, en ne considérant que les distances inférieures à d.

On cherche à améliorer la qualité de la relaxation continue de ce problème en lui rajoutant des coupes

1.2 Etude des coupes $\{0,\frac{1}{2}\}$

Avant de mettre en place la génération de coupe, j'ai étudié l'article de ?koster2009algorithms}.

1.2.1 Définition des coupes $\{0,\frac{1}{2}\}$

On considère le problème d'optimisation linéaire suivant :

$$\begin{cases} min & c^T x \\ tq & Ax \leq b \\ & x \geq 0 \\ & x \in Z^n \end{cases}$$

Définition : Une coupe de Chvatal-Gomory(CG) est définie par :

$$\lfloor u^T A \rfloor x \leq \lfloor u^T b \rfloor, \text{ avec } u \in [0,1[^m$$

Définition : Une coupe $\{0,\frac{1}{2}\}$ est une CG coupe telle que $u \in \{0,\frac{1}{2}\}^m$

1.2.2 Existence d'une coupe

Tout d'abord le document présente le raisonnement qui permet de déterminer si à partir d'une solution non entière le système admet un coupe, c'est à dire un vecteur u tel que l'inégalité n'est pas respectée. Pour cela, on définit une fonction de violation de la manière suivante :

$$z(u,x^*) = \lfloor u^T A \rfloor x^* - \lfloor u^T b \rfloor$$

On cherche donc un vecteur u tel que $z(u,x^*) > 0$. Pour cela on définit les éléments suivants :

$$\bar{A} = A \ mod \ 2, \ \bar{b} = b \ mod \ 2, \ s = b - Ax^* \geq 0$$

Dans un premier temps, l'auteur démontre le lemme suivant, qui donne une condition nécessaire suffisante quand à l'existence d'une coupe :

Soit x^* une solution entière, $\exists$ u $\in \{0,\frac{1}{2}\}^m$ telle que $z(u,x^*) < 0$, si et seulement si $\exists$ v $\in \{0,1\}^m$ qui vérifie $v^T\bar{b}$ impair et $v^T s + (v^T \bar{A} \ mod \ 2)x^* < 1$

Pour démontrer ce lemme, il faut développer l'expression de $z(u,x^*)$:

$$\begin{aligned} z(u,x^*) &= \lfloor u^T A \rfloor x^* - \lfloor u^T b \rfloor \\ &= \frac{1}{2}((2u^T)\bar{b} \ mod \ 2) - v^T s - \frac{1}{2}((2u^T \bar{A} \ mod \ 2)x^*) \end{aligned} \tag{1.1}$$

En posant $v = 2u$, et en remarquant que tous les termes de la différences sont positifs, on en déduit les conditions précédentes.

Ensuite, afin de faciliter d'un vecteur v adéquats sans ignorer aucune non-dominée coupe, l'auteur met en place les règles de simplification suivantes.

Proposition 3 :

- (i) On peut supprimer les colonnes dans $\bar{A}$ qui correspondent à une valeur nulle de x^*. En effet cela ne modifie pas l'inégalité $v^T s + (v^T \bar{A} \; mod \; 2)x^* < 1$.
- (ii) On peut supprimer les lignes nulles dans ($\bar{A}$,$\bar{b}$), car cela n'a d'effet que su la slack s.
- (iii) On peut supprimer les colonnes de zéro de $\bar{A}$.
- (iv) On peut remplacer les colonnes identiques dans $\bar{A}$ par une seule colonne représentative, dont la variable associée est la somme des variables correspondantes. Soit aucune soit toutes les variables correspondantes devront sinon être arrondies.
- (v) On peut supprimer toutes colonnes unitaires de $\bar{A}$ de la forme $\bar{a}^i = e^j$ en prenant soin d'ajouter x_i^* à s_j.
- (vi) Toutes les colonnes dont la slack associée vérifie $s_j \geq 1$ peuvent être supprimées, car sinon cela contredit directement l'inégalité si le v_j correspondant vaut 1.
- (vii) On peut supprimer toutes les colonnes identiques dans ($\bar{A}$,$\bar{b}$) sauf pour celle dont la slack est la plus petite.

Lors de la première itération chaque ligne du système ($\bar{A}$,$\bar{b}$,s) correspond à une unique inégalité. Cependant pour les itérations suivantes, on doit mettre en place une opération afin de pouvoir combiner les lignes. Si on combine les lignes i et j sur la ligne j, cette opération est la suivante :

$$\forall k, \bar{a_{ik}} + \bar{a_{jk}} = \bar{a_{jk}} \text{ mod } 2, \; \bar{b_j} = \bar{b_i} + \bar{b_j} \text{ mod } 2, \; s_j = s_i + s_j \text{ et } R_j = R_i \Delta R_j$$

avec

$$X \Delta Y = X \cap Y \backslash (X \cup Y)$$

On initialise R_j de la manière suivante :

$$R_j = \{j\}$$

R_j correspond alors à l'ensemble des indices des lignes dont la ligne j est la combinaison linéaire.

Dans le cas particulier suivant, il est possible de repérer facilement un vecteur u entraînant la violation de la coupe associée. Si on se place dans le cas où la i^{eme} rangé de $\bar{A}$ est nulle et que $\bar{b_j} = 1$, alors si $s_j < 1$, en posant u tel que

$\forall j \in R_j$ $u_j = \frac{1}{2}$ et 0 sinon, on définit une coupe violée du système initial (A,b,s). Cela se voit rapidement, en posant v tel que $\forall j \in R_j$ 1 et 0 sinon, et en appliquant le lemme précèdent.

L'auteur met en place deux nouvelles propositions, qui mettent en place de nouvelles règles de simplification.
Proposition 5 :
Soient i et k des indices de lignes et de colonnes tels que $\bar{a}_{ik} = 1$ et $s_i = 0$, alors on peut supprimer la colonne k de $\bar{A}$ à condition d'ajouter la ligne i à toutes les autres lignes telles que $\bar{a}_{ik} = 1$ et en imposant $s_i = x_k^*$. La démonstration repose sur le fait que les fonctions de violation sont égales avant et après cette combinaison linéaire.
Proposition 6 :
Soient i et k des indices de lignes et de colonnes tels que $\bar{a}_{ik} = 1$ et $s_i = 0$ et $x^* \geq 1$, alors on peut supprimer la ligne i et la colonne k de $\bar{A}$ à condition d'ajouter la ligne i à toutes les autres lignes telles que $\bar{a}_{ij} = 1$ et en imposant $s_j = x_j^*$. Pour cela on applique la proposition précédente et on se retrouve dans la situation de la troisième proposition de suppression.

Ces règles de simplification, ainsi que de détection de coupes violées seront très utiles pour la mise en place de la résolution heuristique.

1.2.3 Pré-traitement des données

Avant de mettre en place la recherche de coupe, un pré-traitement des données est réalisée :

- (a) supprimer les colonnes nulles en appliquant (3i) ;
- (b) supprimer les colonnes qui correspondent à x^* petit en appliquant (3ii) ;
- (c) supprimer les lignes de slack supérieure ou égale à 1 en appliquant (3vi) ;
- (d) supprimer les colonnes en appliquant la proposition 5 ;
- (e) supprimer les colonnes unitaires en appliquant (3v) ;
- (f) ajouter à l'ensemble des coupes violées toute coupe correspondant à une

ligne où $\bar{b} = 1$ et qui a une violation d'au moins ϵ, puis la supprimer de la matrice car toute coupe la contenant aura une violation inférieure ;

— (g) supprimer toutes les colonnes identiques sauf pour celle de plus petite slack (3vii) ;

Enfin à chaque étape on prend le soin de supprimer les colonnes et lignes nulles, ainsi que les lignes de slack supérieure à 1 dans ($\bar{A}$,$\bar{b}$).

1.2.4 Méthode avec un programme linéaire

Une première approche peut-être de modéliser le problème de coupe par un système linéaire de la forme suivante :

$$\begin{cases} z = & min\ s^T v + (x^*)^T y \\ tq & \bar{b}^T v - 2q = 1 \\ & \bar{A}^T v - 2r - y = 0 \\ & v \in \{0,1\}^n \\ & y \in \{0,1\}^n \\ & r \in Z_+^n \\ & q \in Z^n \end{cases}$$

La première égalité impose que $\bar{b}^T v$ soit impair, la seconde permet de calculer le reste de la division euclidienne de $\bar{A}^T v$ par 2. La fonction coût permet de déterminer $v^T s + (v^T \bar{A}\ mod\ 2)x^*$. Si cette valeur est inférieure ou égale à 1 alors une coupe violée n'a pas été trouvée, et si cette valeur est supérieure à 1 alors il en existe une coupe violée, qui correspond à la combinaison des inégalités pour laquelle v_j vaut 1. Cependant, on peut s'attendre à ce que le temps de résolution de ce programme auxiliaire soit trop long, il est donc intéressant d'étudier des méthodes de résolution heuristique.

1.2.5 Méthode heuristique

La méthode de résolution heuristique suivante est développée. Tant que l'on ne trouve pas de coupe violée, on teste toutes les combinaisons de plus en plus

grande d'inégalités. De plus les données sont pré-traitées afin d'accélérer la vitesse de résolution.

1.2.6 Résultat et performance des différentes méthodes

Résultat du pré-traitement des données

Les tests ont été réalisés sur tous les noeuds du *branch-and-cut*, sans imposer de violation minimale. Les étapes (a) à (c) réduisent la taille de $\bar{A}$ d'en moyenne 83,2%, allant de 46,96% jusqu'à 99,9%. Après applications des étapes (a) à (c), on étudie les résultats des étapes suivantes en fonction de la taille actuelle de la matrice $\bar{A}$. Les étapes (d) et (e) simplifient le système de 1,26% et 1,5% en supprimant respectivement les colonnes unitaires et les lignes nulles qui contiennent une coupe violée. Enfin on peut supprimer 40,7% en supprimant les colonnes identiques. On obtient alors en considérant chacune des étapes une simplification de 95,5% en moyenne. Cette simplification est très significative et permet d'accélérer grandement le temps de résolution, et permet de résoudre en moins de 10s des instances qui ne pouvaient sinon pas être résolues en dessous de ce temps.

Comparaison du nombre de noeuds du *branch-and-cut*

Un premier élément pour comparer les performances de la coupe $\{0,\frac{1}{2}\}$ est la mesure du nombre de noeuds dans le *branch-and-cut*. Dans cette approche, on compare 3 éléments :

- Fischetti et Lotti font appel à la coupe de Chvatal-Gomory pour la séparation
- SCIP par défaut
- SCIP en remplaçant le séparateur standard avec la coupe $\{0,\frac{1}{2}\}$ en ajoutant à chaque fois que la coupe la plus violée.

Par rapport à Fischetti et Lotti, 40% de coupe en moins sont réalisées. Cependant SCIP réalise en général plus d'un branch&cut, on est donc amené en général à réaliser plus de coupes. En considérant la méthode faisant appel au programme linéaire auxiliaire, et en ajoutant les coupes violées ne jouant pas un rôle dans la coupe optimale, on arrive à diminuer de 26% en général le nombre de noeuds

dans le *branch-and-cut*. Cependant on atteint ce résultat en sacrifiant le temps de calcul, et en outre on observe une augmentation du nombre noeuds dans certain cas. En effet on voit une augmentation de plus 158% au temps de calcul. Une telle évaluation du problème n'est donc pas adaptée. On va choisir de comparer plutôt le temps CPU.

Comparaison du temps CPU

Afin de mettre en évidence que la résolution faisant à appel un programme linéaire auxiliaire n'est pas adaptée, on va comparer le temps de résolution des 3 éléments suivants :

— SCIP par défaut ;
— SCIP faisant appel au programme linéaire auxiliaire ;
— SCIP faisant appel à la méthode heuristique ;

De plus, on ne considère pas toutes les coupes violées, seulement celle dont la fonction de violation est supérieure à 0.35 afin d'éviter de générer des coupes inefficaces. Ensuite, jusqu'à 100 coupes violées sont transmises à SCIP, qui va choisir de les prendre en compte ou non. La méthode heuristique améliore de 7% en moyenne sur toutes les instances le temps de résolution de l'algorithme, et de 21% pour les instances dont le temps de résolution absolue est plus grand que 10s avec SCIP par défaut.
On remarque bien que la résolution heuristique semble la plus adaptée.

Bibliographie