Master d'Informatique - Spécialité Androide
Module MAOA

Recherche Opérationnelle et Optimisation Combinatoire

Partie C - Algorithmes de coupes (Branch-and-Cut)

Pierre Fouilhoux

Sorbonne Université

2019-2020

Lorsqu'un PLNE (P) n'a pas une relaxation linéaire $(\tilde{P})$ entière, le PL $(\tilde{P})$ possède don des points extrêmes fractionnaires.

Il est utile de déterminer des inégalités que l'on appelera valides (et non redondantes voirs section "Approches Polyédrales") pour un PLNE. Ces inégalités sont choisies de manière à couper ces points extrêmes fractionnaires.

On les détermine soit par intuition, soit par extrapolation de cas simples, soit en les dérivant par des procédés tels que celui de Chvàtal-Gomory,...

L'ensemble de ces idées algorithmiques permettent aujourd'hui de résoudre efficacement des PLNE de bonnes tailles (quelques milliers de lignes/contraintes). Surtout dans les cas où le PLNE a une structure particulière. On nomme souvent cet aspect du domaine de recherche MIP pour Mixed Integer Programming.

Inégalités valides

Pour un PLNE donné, $(P) \quad Max\{c^T x \mid Ax \leq b, x \in \boldsymbol{N}^n\}$ avec ou non un nombre compact de contraintes et de variables, on peut ajouter des contraintes supplémentaires à sa relaxation linéaire $(\tilde{P})$.

On considére le polyèdre $\mathcal{P} = \{x \in \boldsymbol{R}^n \mid Ax \leq b\}$ associé à (P), appelé **polyèdre de la formulation**.

Definition

Une inégalité $ax \leq \alpha$ est dite **valide** pour (P)
si elle est vérifiée par tous les points entiers de $\mathcal{P}$.

Ajouter une inégalité valide au PLNE (P) conserve l'espace des solutions de (P) : en revanche cela peut changer sa relaxation $(\tilde{P})$.

Coupes

Une solution fractionnaire potentielle de la relaxation linéaire $(\tilde{P}) = max\ \{c^T x \mid Ax \leq b\}$ est donc un point extrême fractionnaire du polyèdre $\mathcal{P} = \{x \in \boldsymbol{R}^n \mid Ax \leq b\}$.

Considérons un tel point extrême fractionnaire $\tilde{x}$.

Une inégalité $ax \leq \alpha$ est dite **violée** par $\tilde{x}$ si $a\tilde{x} > \alpha$.
Une inégalité $ax \leq \alpha$ est dite **serrée** par $\tilde{x}$ si elle satisfait $ax \leq \alpha$ à l'égalité, c'est-à-dire $a\tilde{x} = \alpha$.

Ajouter au PLNE (P) une inégalité valide ET violée par $\tilde{x}$ permet de couper un point extrême indésirable de l'espace des solutions optimales.
On dit alors qu'une telle inégalité est une **coupe** pour le PLNE.

De plus, ces contraintes permettent d'obtenir une relaxation plus intéressante (même si parfois c'est numériquement très faible).
On parle alors de **renforcement** de la formulation.

Exemple pour la formulation du stable

Soit un graphe $G = (V, E)$ muni d'un poids c associé aux sommets. Le problème du stable maximum se modélise selon la formulation suivante

$$\begin{aligned} \text{Max} \sum_{u \in V} c(u)x(u) & & \\ x(u) + x(v) \leq 1 & \quad \text{pour tout } uv \in E, & (1) \\ 0 \leq x(u) \leq 1 & \quad \text{pour tout } u \in E, & (2) \\ x(u) \text{ entier}, & \quad \text{pour tout } u \in V. & \end{aligned}$$

En effet, les contraintes (2) sont trivialement valides : on les appelles d'ailleurs les contraintes triviales car elles sont vérifiées par tout vecteurs d'incidence d'un ensemble.

Les contraintes d'arêtes (1) sont valides.
En effet, prenons un stable $S \subset V$ et considérons son **vecteurs d'incidence** χ^S tel que $\chi^S[u] = 1$ si $u \in S$ et $\chi^S[u] = 0$ sinon.
On peut voir que pour un sommet u donné, un stable contient au plus une arête incidente à u, donc $\chi^S[u] + \chi^S[v] \leq 1$ pour toute arête $uv \in E$. Ce qui est équivalent à dire que χ^S satisfait toutes les inégalités d'arêtes.

Exemple pour la formulation du stable

On vient de prouver que toute solution du problème du stable vérifie la formulation.

ATTENTION Pour prouver l'équivalence entre la formulation et le problème du stable, il faut aussi prouver le "retour", c'est-à-dire que toute solution entière de la formulation correspond à un stable.
C'est bien le cas : prenons une solution entière x^* de la formation. Par les contrantes triviales et d'intégrité, on peut poser $M = \{u \in V \mid x^*(u) = 1\}$ qui est un sous-ensemble de sommets. Si deux sommets sont adjacents dans V, on voit par les contraintes d'arêtes que ces deux sommets ne peuvent appartenir tous les deux à M : donc M est bien un stable.

Et pour terminer, on remarque que les fonctions objectives du PLNE et du problème du stable maximum sont exactement la même fonction.

La formulation est bien équivalente au problème.

Exemple pour la formulation du stable

Cette formulation du stable a une relaxation linéaire non entière (sinon le problème du stable serait polynomial...).

Par exemple, si G contient une clique (un ensemble de sommets induisant un sous-graphe complet) de k sommets, le point y suivant :

$y(u) = \frac{1}{2}$ pour tout $u \in K$

$y(u) = 0$ pour tout $u \in V \setminus K$

vérifie k inégalités d'arêtes et $n - k$ inégalités triviales à l'égalité.
De plus ces n inégalités sont linéairement indépendantes.
C'est un point extrême fractionnaire (qui est solution optimale pour la fonction poids cardinalité).

Comment le "couper" ?

Exemple pour la formulation du stable

Les **inégalités de clique**

$$\sum_{u \in K} x(u) \leq 1 \qquad \text{pour toute clique } K, \tag{3}$$

sont valides pour la formulation du stable.

En effet, tout vecteur d'incidence d'un stable vérifie ces inégalités car au plus un sommet peut être pris dans une clique d'un graphe.

De plus la contrainte associée à une clique K (de taille au moins 3) coupe le point extrême y.
En effet, $\sum_{u \in K} y(u) = \frac{k}{2} > 1$ dès que $k \geq 3$.

On peut remarquer que les contraintes de clique de taille 2 sont en fait les contraintes d'arêtes.

Pour simplifier l'écriture, on écrit souvent $x(K) = \sum_{u \in K} x(u)$ et les inégalités de cliques deviennent alors $x(K) \leq 1$ pour toute clique K.

Exemple pour la formulation du stable

Supposons que le graphe G contienne à présent un cycle impair C : un cycle comportant un nombre impair de sommets (qui est le même que le nombre d'arêtes). Notons $V(C)$ les sommets correspondant au cycle C (un cycle est une suite d'arêtes).

Regardons le point y tel que :
$y(u) = \frac{1}{2}$ pour $u \in V(C)$
$y(u) = 0$ pour $u \in V \setminus V(C)$
Ce point satisfait $|C|$ inégalités d'arêtes et $n - |C|$ inégalités triviales à l'égalités.
Ces inégalités sont linéairement indépendantes car C est impair (ce n'est pas le cas si C est pair !).
Comme précédemment ce point est un point extrême fractionnaire de la formulation

Comment le couper ?

Exemple pour la formulation du stable

Les **inégalités de cycle impair**

$$\sum_{u \in V(C)} x(u) \leq \frac{|C|-1}{2} \qquad \text{pour tout cycle impair } C, \tag{4}$$

sont valides pour la formulation du stable.

En effet, prenons S un stable de V.
On peut prendre au plus un "sommet sur deux" du cycle dans S. Et on voit qu'aucun stable ne peut contenir $\frac{|C|+1}{2}$ sommets du cycle. Donc un stable contient au plus $\frac{|C|-1}{2}$ d'un cycle impair : l'inégalité est bien valide.

Et elle coupe le point y.
En effet, $\sum_{u \in V(C)} y(u) = \frac{|C|}{2} > \frac{|C|-1}{2}$.

Remarque : On note souvent $x(V(C)) = \sum_{u \in V(C)} x(u)$

Comparaison de formulations

Ajouter une inégalité valide qui coupe une solution fractionnaire optimale de la relaxation améliore la valeur de la relaxation linéaire : on parle de **renforcement**.

Dans un cadre plus général, il n'est pas évident de prouver "à l'avance" si une inégalité valide va être systématiquement un renforcement.

On le démontre au cas par cas.

Le problème du voyageur de commerce asymétrique

Soient n villes avec c_{ij} coût de transport de i à j (asymétrique).
Déterminer un circuit passant par toutes les villes de plus petit coût.

Avec les variables $x_{ij} = 1$ si l'arc (i, j) est dans le circuit et 0 sinon et des variables additionnelles u réelles, la formulation MTZ que nous avons vue est une formulation du problème.

$$
\begin{aligned}
&\text{Min } \textstyle\sum_{i,j} c_{ij} x_{ij} && \\
&\textstyle\sum_{j \in V} x_{ij} = 1 && \forall i \in V, \\
&\textstyle\sum_{i \in V} x_{ij} = 1 && \forall j \in V, \\
&u_1 = 1, && \\
&2 \leq u_i \leq n && \text{pour tout } i \neq 1, \\
&u_i - u_j + 1 \leq n(1 - x_{ij}) && \text{pour tout } i \neq 1, j \neq 1, \\
&u_i \in \boldsymbol{R} && \forall i \in V \\
&x_{ij} \in \boldsymbol{N} && \forall (i,j) \in V \times V.
\end{aligned}
$$

Cette formulation est compacte !
Mais elle n'est pas très performante en terme de valeur de relaxation linéaire. Un point x avec de très petites valeurs x (non entière) va pouvoir vérifier les inégalités MTZ et être optimales : cette valeur est très basse.

Le problème du voyageur de commerce asymétrique

Une autre formulation mène à une relaxation bien plus intéressante dites formulation par les coupes

$$
\begin{aligned}
&\text{Min } \textstyle\sum_{i,j} c_{ij} x_{ij} \\
&\textstyle\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V, \\
&\textstyle\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V, \\
&\sum_{e \in \delta^+(W)} x(e) \geq 1 \qquad \text{pour tout } W \subsetneq V \text{ et } W \neq \emptyset, \\
&x_{ij} \in \mathbb{N} \qquad \forall (i,j) \in V \times V.
\end{aligned}
$$

Cette formulation apporte en pratique de bien meilleure valeur (expérimentale) de relaxation.

Et on peut prouver cela de manière théorique !

Le problème du voyageur de commerce asymétrique

La difficulté ici est que les formulations ne sont pas tout à fait sur le même eespace de variables : on doit faire "disparaître" les x.
Un façon de le faire est de sommer les inégalités MTZ sur un circuit C : on obtient alors

$$\sum_{(i,j)\in C} x_{ij} \leq \left(1 - \frac{1}{n-1}\right)|C|$$

En notant $V(C)$ les sommets de C et en l'appliquant à la formulation par les coupes, on obtient :

$$\sum_{i,j\in V(C)} x_{ij} \leq |C| - 1$$

On voit ici que cette deuxième inégalité est bien plus forte en terme de coupes et donc de relaxation.

Cette idée peut se généraliser et on peut prouver que toutes les inégalités provenant de la formulation MTZ sont battue par une inégalité plus forte dans la formulation par les coupes : On dit qu'elle **domine** la formulation MTZ.

Méthodes génériques

Une question importante est de se demander comment produire des inégalités automatiquement pour couper des points fractionnaires et renforcer les formulations.

C'est un domaine d'étude riche qui a permis les meilleurs améliorations des solveurs entiers, citons-en quelques unes :

- La méthode d'arrondi de Chvátal-Gomory (voir plus loin) : elle est malheureusement peu efficace si elle est utilisée seule mais très utile pour renforcer un PLNE quelconque.
- L'utilisation des structures sous-jacentes de "stable" dans la matrice des contraintes : on repère tout d'abord des paires de variables ne pouvant être toutes deux à 1 : cela crée un graphe dit d'incompatibilité. On sait que toute soltion sera entre autre un stable de ce graphe : donc toutes les inégalités de cliques et de cycles impaires sont valides (et très très utiles).
- Par déduction à partir de certaines contraintes : par exemple les inégalité cover provenant des inégalité de knapsack (voir plus loin).

Méthodes génériques

- La construction de Lovasz-Shrivjer : qui permet de déduire des contraintes en les "multipliant" entre elles, en linéarisant par ajouts de variables de linéarisation (principe de Sheralli-Adams), puis en projetant sur l'espace d'origine. Cela mène à la méthode de Lift-and-Project initiée par Balas.
- Les inégalités disjonctives qui génèrent aujourd'hui les coupes les plus efficaces.
- et quelques autres

Somme de Chvàtal

Une technique célèbre pour obtenir de telles inégalités valides de manière automatique est la somme de Chvátal (avec arrondi dit de Gomory) basé sur la recette suivante :

Additionner plusieurs contraintes
Diviser les coefficients de la contrainte obtenue de manière à ramener le plus petit coefficient à 1.
Utilisant le fait que certaines sommes de termes sont entières
En déduire une nouvelle inégalité valide

Somme de Chvàtal

Exemple du polyèdre du stable.
Partons des inégalités aux arêtes.
Supposons que G contienne un cycle de G.
On somme les k contraintes associées aux k arêtes du cycle :

$$\begin{array}{lllllll} x(u_1) & + & x(u_2) & & & \leq & 1 \\ & & x(u_2) & + & x(u_3) & \leq & 1 \\ & & & & .... & & \\ x(u_1) & & & & + \quad x(u_k) & \leq & 1 \end{array}$$

$$2 \sum_{u \in V(C)} x(u) \leq k$$

$$\sum_{u \in V(C)} x(u) \leq \frac{k}{2}$$

Somme de Chvàtal

Comme le côté gauche de l'inégalité est une somme d'entier, le côté droit peut lui aussi être ramené à l'entier immédiatement inférieur. D'où, comme k est impair, on obtient finalement une inégalité valide par construction !

$$\sum_{u \in V(C)} x(u) \leq \frac{k-1}{2}.$$

Lorsque k est impair, on obtient ainsi les contraintes dites de cycles impairs pour le problème du stable :

$$\sum_{v \in V(C)} x(v) \leq \frac{|C|-1}{2} \quad \forall C \text{ cycle impair}$$

Méthodes duale fractionnaire

On peut utiliser la somme de Chvátal-Gomory pour créer un algorithme de coupes génériques et adaptables à tout PLNE : la métode duale fractionnaire.

Elle utilise le tableau de la méthode duale de l'algorithme du simplexe afin de produire itérativement une suite d'inégalités valides qui coupent peu à peu les points extrêmes non entiers. Elle mène donc à elle-seule à une solution entière optimale !
Cette méthode converge vers une solution optimale en temps fini... mais elle est excessivement lente ! ! !

En revanche, les inégalités produites par cette méthode permettent d'accélérer fortement les algorithmes de coupes et branchements génériques.

Exemple d'inégalités génériques, les cover inequalities

Après transformations basiques, toute inégalité d'un PLNE peut être vue comme une inégalité de "sac-à-dos" :

$$\sum_{i=1}^{n} a_i x_i \leq b \text{ avec } a_i \geq 0, b \geq 0.$$

Un ensemble de variables $R \subset \{1, ..., n\}$ est un **cover** si $\sum_{i \in R} a_i > b$

et alors l'inégalité

$$\sum_{i \in R} x_i \leq |R| - 1.$$

est valide pour la formulation (voir cahier d'exercice).

Nous avons rencontré deux cas où on considère un grand nombre de contraintes :

- les formulations PLNE contenant au départ un grand nombre de contraintes (potentiellement exponentiel) que l'on ne peut donc pas énumérer : formulation du problème du voyageur de commerce,...
- des exemples où l'ajout de contraintes supplémentaires permet d'obtenir une meilleure valeur de relaxation : on appelle cela parfois le "renforcement" de formulation : ajout des contraintes de cliques pour la formulation du problème du stable.

Comment gérer algorithmiquement l'ajout d'un grand nombre de contraintes dans un PLNE ?

Comment résoudre une formulation non-compacte ?

Considérons ici un programme linéaire

$$(P) \quad Max\{c^T x \mid Ax \leq b\}$$

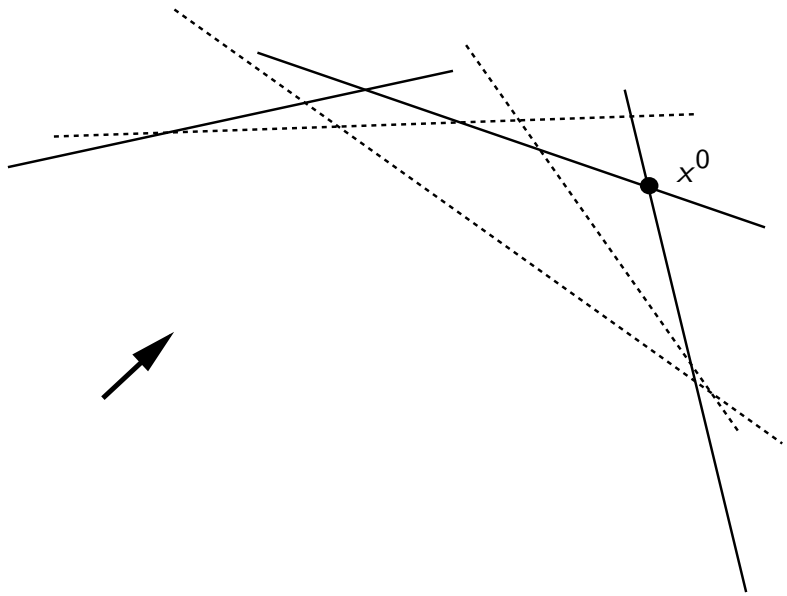
comportant n variables avec n une quantité limitée en taille (quelques milliers) mais où A contient un nombre important, par exemple exponentiel par rapport à n de contraintes : on dit que le programme est non compact sur les contraintes.

Un tel programme ne peut donc pas être introduit comme instance d'un algorithme de B&B et donc ne peut pas être utilisé directement dans les solveurs entiers. Nous allons pourtant voir qu'il existe un cadre algorithme efficace, les algorithmes de coupes pour ces programmes.

Initialisation

Choisissons $A_0 x \leq b_0$ sous-ensemble de contraintes de $Ax \leq b$ tel qu'il existe une solution finie x^0 au problème restreint à A_0.

$$(P_0) \begin{cases} \text{Max } c^T x \\ A_0 x \leq b_0 \end{cases}$$



Une solution optimale x_0 est donc un point extrême du polyèdre $A_0 x \leq b_0$.

La figure illustre en 2 dimensions, les contraintes de $A_0 x \leq b_0$ en traits pleins et les autres contraintes de $Ax \leq b$ en pointillés.
La flèche indique la "direction d'optimisation" : la direction orthogonale à toute droite d'équation $z = c^T x$.

Problème de séparation

Notons que si x_0 satisfaisait toutes les inégalités de $Ax \leq b$ restante, alors x_0 serait optimal pour tout P (et il serait point extrême de $Ax \leq b$).

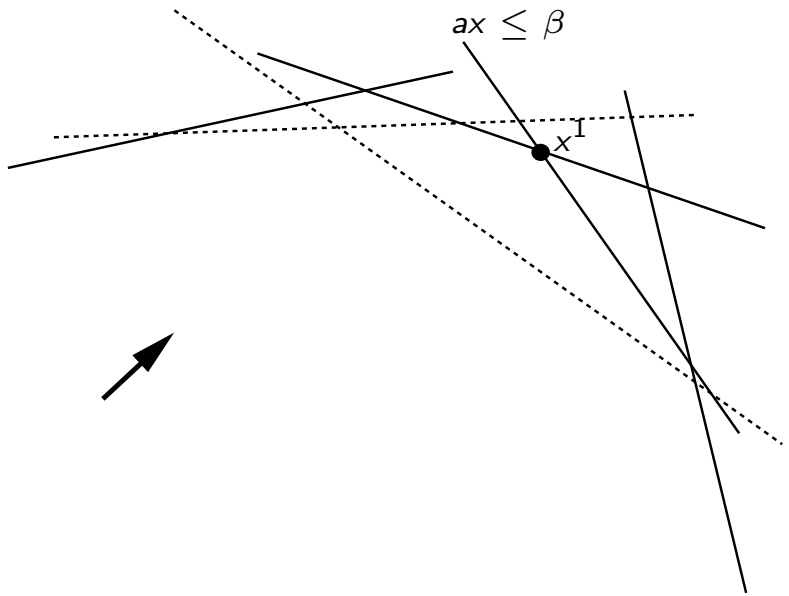
Si x^0 un point extrême de $Ax \leq b$, il existe une inégalité de $Ax \leq b$ qui n'est pas satisfaite par x^0 : on dit que x^0 **viole** la contrainte.

Problème de séparation
Etant donné un point $\tilde{x} \in \boldsymbol{R}^n$, le **problème de séparation** associé à $Ax \leq b$ et $\tilde{x}$ consiste à déterminer si $\tilde{x}$ satisfait toutes les inégalités de $Ax \leq b$ et sinon à trouver une inégalité de $Ax \leq b$ violée par $\tilde{x}$.

Itération

Si le problème de séparation pour $\tilde{x} = x_0$ fournit une inégalité $\alpha x \leq \beta$ violée par $\tilde{x}$, on l'ajoute au programme (P_0) pour obtenir le programme (P_1) :

$$(P_1) \begin{cases} \text{Max } c^T x \\ A_0 x \leq b_0 \\ \alpha x \leq \beta \end{cases}$$



Et on réitère !

Algorithme de séparation

Un algorithme qui répond au problème de séparation s'appelle **algorithme de séparation**.

Si on connaît un tel algorithme de séparation, on sait donc déterminer une inégalité $ax \leq \beta$ qui est violée par le point extrême x^0.
Une telle inégalité est appelée une **coupe** car elle "coupe" le point indésirable de l'espace des solutions possibles.
Dans l'illustration précédente, la contrainte $\alpha x \leq \beta$ qui était violée par x^0 a été ajoutée au système courant : on voit qu'elle a **séparé** une partie de l'espace indésirable du programme P_1

Algorithme de coupes

En réitérant l'algorithme de séparation autant tant que l'on peut ajouter des contraintes violées, on obtient alors une algorithme appelé **méthode ou algorithme de coupes** (cutting plane algorithm).

Algorithme de coupes (cutting-plane based algorithm) :
Tant qu'il existe une inégalité de $Ax \leq B$ violée par x_i
Ajouter l'inégalité au programme (P_i) : on obtient P_{i+1}
Résoudre la relaxation linéaire de P_{i+1} : on obtient une solution x_{i+1}
$i = i + 1$

Terminaison et validité :
L'algorithme de coupes se terminent nécessairement : au pire il aura énuméré toutes inégalités de $Ax \leq \alpha$.
A la fin d'une méthode de coupes, on obtient donc nécessairement un point extrême x^* qui est point extrême du système courant et qui n'est pas violé par une contrainte de $Ax \leq b$: on obtient donc une solution optimale de (P).

Complexité de la méthode de coupes

La question importante est donc alors de connaîtrela complexité d'une méthode de coupes. Ceci nous est donné par le résultat extrêmement puissant suivant :

Theorem (Grötschel, Lovász, Schrijver, 1981)

Une méthode de coupes sur un système $Ax \leq b$ de contraintes est polynomial **si et seulement** *si l'algorithme de séparation associé à $Ax \leq b$ est polynomial.*

Ce résultat fondamental permet ainsi de manipuler des formulations exponentielles pour la PLNE ! Il indique ainsi :

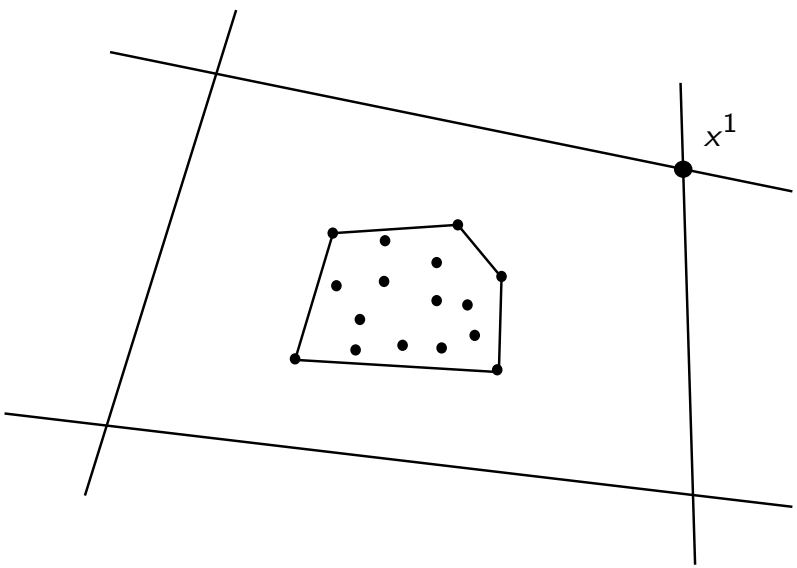
Optimiser est équivalent à Séparer.

Ce résultat est issu de l'algorithme des ellipsoïdes de Karmarkar et n'indique pas le degré du polynôme de la complexité de la méthode de coupes... En théorie il semble être élevé mais en pratique... c'est très efficace.

En pratique :

On connaît uniquement $Ax \leq B$ un premier ensemble d'inégalités tel que tout point entier les vérifiant est solution.

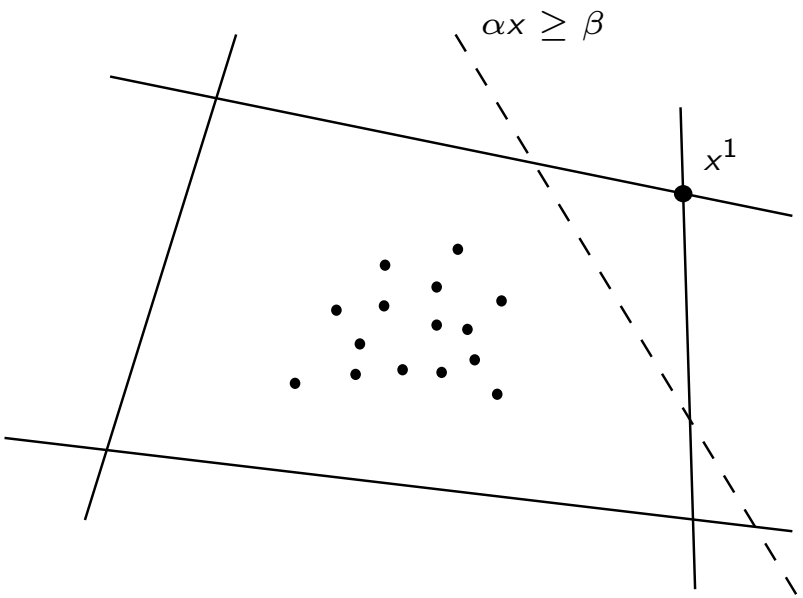
$$(P) \begin{cases} \text{Max } cx \\ Ax \leq B \\ x \text{ entier} \end{cases}$$



En pratique :

On connaît uniquement $Ax \leq B$ un premier ensemble d'inégalités tel que tout point entier les vérifiant est solution.

$$(P_2) \begin{cases} \text{Max } cx \\ Ax \leq B \\ \alpha x \leq \beta \\ x \text{ entier} \end{cases}$$



Le polyèdre du problème est l'inconnue... (voir section E)

Taille du PL manipulé

Une autre conséquence concerne la taille du système linéaire utilisé : en fait, comme on ajoute une inégalité un nombre polynomial de fois : le système courant a au plus une taille polynomiale ! Ainsi, au lieu de manipuler un système exponentiel, un système polynomial est suffisant.

Cette remarque n'est pas surprenante, en effet, pour définir un point extrême (solution) il suffit de n contraintes (linéairement indépendantes) satisfaites à l'égalité : le reste peut être donc être "écarté" du PL correspondant : la question est alors : comment trouver ces n contraintes ? En effet, si on les a, une seule utilisation de l'algorithme de séparation permettrait de tester si le point est une solution de $Ax \leq b$.

Preuve de polynomialité et de NP-difficulté

Cette équivalence entre Séparation et Optimisation permet de prouver que certains problèmes sont polynomiaux : il faut pour cela avoir prouvé qu'un problème se ramène à résoudre un PL correspondant à un polyèdre entier à nombre exponentiel d'inégalités mais pour lequel vous connaissez une méthode de coupes polynomiale.

Inversement, si vous prouvez que, **pour toute fonction poids**, la méthode de coupes associées à votre PL est NP-difficile, alors le problème équivalent au PL sera à son tour NP-difficile.

Séparation pour le problème du stable maximum

Inégalités de cycles impairs

Les inégalités de cycles impairs

$$\sum_{u \text{ dans } C} x(u) \leq \left\lfloor \frac{|C|}{2} \right\rfloor \forall \text{ cycle impair } C$$

sont séparables en temps polynomial.

Inégalités de cycles impairs

Considérons un point $\tilde{x}$ solution de la relaxation linéaire de la formulation aux arêtes du problème du stable.

Donc le point $\tilde{x}$ vérifient toutes les inégalités d'arêtes (mais si l'on cherche à le couper, c'est qu'il est fractionnaire).

On pose alors

$$w_{uv} = 1 - \tilde{x}_u - \tilde{x}_v$$

qui est bien positif ou nul car $\tilde{x}$ vérifient toutes les inégalités d'arêtes.

L'inégalité est alors équivalente à

$$\sum_{e \in C} w_{uv} \geq 1$$

Le problème de séparation revient donc à déterminer un plus petit cycle impair par rapport au poids w dans le graphe.
En effet :
- Si ce poids est strictement plus petit que 1, cela signifie qu'on a déterminé une inégalité violée.
- Si ce poids est plus grand que 1, cela signifie que l'inégalité avec le plus petit écart avec la violation, n'est justement pas violée : donc il n'y a pas d'inégalités violées.

Inégalités de cycles impairs

Pour déterminer un cycle impair de plus petits poids par rapport à w, on crée un graphe biparti G'

- où chaque sommet i de G est dupliquée en i' et i'' : V est ainsi dupliqué en l'ensemble V' et l'ensemble V''.
- et où chaque arête ij de G devient deux arêtes $i'j''$ et $i''j'$ portant chacune le poids w_{ij}.

On voit bien que G est biparti car il n'y pas d'arête au sein de V' ou au sein de V''. Dans ce cycle, tout chemin (élementaire) de i' à i'' va correspondre à un cycle impair car il faut effectuer un nombre impair d'aller retour entre V' et V''.

En effectuant une recherche d'un plus court chemin de i' à i'' dans G' pour tout $i \in V$, on obtient un cycle impair de plus petit poids dans G.

Comme la recherche d'un plus court chemin avec des poids w positifs est polynimial, cet algorithme de séparation revient à n fois la complexité de la recherche d'un plus court chemin et est donc polynomiale.

Séparation pour le problème du stable maximum

Inégalités de cycles cliques

Les inégalités de cliques

$$\sum_{u \in K} x(u) \leq 1 \text{ pour toute clique } K$$

ont un problème de séparation associé NP-complet...
En effet, prouver qu'il n'y a plus d'inégalités violées est équivalent à rechercher une clique de poids maximal dans G : ce qui est un problème NP-difficile.

Donc on ne peut pas construire un algorithme de coupes polynomial pour cette formulation.

Inégalités de cycles cliques

En revanche, on peut quand même considérer la formulation suivante

$$\begin{aligned} \text{Max} \sum_{u \in V} & c(u)x(u) \\ & x(u) + x(v) \leq 1, \text{ pour tout } uv \in E, \\ & \sum_{u \in K} x(u) \leq 1, \text{ pour “quelques” cliques } K \text{ de } G, |K| \geq 3. \qquad (5) \\ & x(u) \in \{0,1\}, \text{ pour tout } u \in V. \end{aligned}$$

En effet, seules les contraintes d'arêtes sont nécessaires à la formulation : elles peuvent donc être énumérées et utilisées tout au long de l'algorithme de coupes.

Les contraintes associées à des cliques de tailles au moins 3 seront elles utilisées dans un algorithme de coupes heuristiques : c'est-à-dire qu'au lieu de rechercher s'il existe ou non une contrainte de cliques violées, on recherche heuristiquement une contrainte de cliques violée !

Elle viendront renforcer la formulation très efficacement.

Inégalités de cycles cliques

Un algorithme très rapide et performant pour la séparation heuristique des inégalités de clique est l'algorithme glouton suivant :

Pour une solution optimale x^* de la relaxation linéaire :
- on classe les sommets par ordre de valeurs $x^*(u)$ décroissante.
- on initialise un ensemble K avec le premier sommet (de plus grande valeur)
- puis, itérativement, on essaye d'ajouter un sommet dans l'ordre des valeurs s'il forme une clique avec les sommets de K.

Cet algorithme glouton n'est évidemment pas exact mais permet d'obtenir des contraintes de cliques en temps polynomial : on peut donc le réitérer dans un processus de génération de contraintes.

On peut noter également qu'il fournit une inégalité de cliques K pour une clique K maximale au sens de l'inclusion : on verra en section F que c'est très utile car toutes les inégalités de clique non maximale sont "dominées" fortes par les inégalités de clique maximale.

Séparation pour le problème du voyageur de commerce

Inégalités de coupes

Considérons la formulation suivante pour le voyageur de commerce symétrique que nous avons vu dans le début de cette partie.

$$Min \sum_{e \in E} c(e)x(e)$$

$$\sum_{e \in \delta(v)} x(e) = 2, \text{ pour tout } u \in V, \tag{6}$$

$$\sum_{e \in \delta(W)} x(e) \geq 2, \text{ pour tout } W \subsetneq V \text{ et } W \neq \emptyset, \tag{7}$$

$$x(e) \geq 0, \text{ pour tout } e \in E$$

$$x(e) \in \{0, 1\}, \text{ pour tout } e \in E.$$

Les contraintes (6) de degré sont au nombre de $|V|$ et sont toujours présentes dans la formulation.
En revanche les contraintes (7) sont en nombre exponentiel mais elles sont séparables en temps polynomial !

Inégalités de coupes

On peut construire un algorithme de coupes en prenant comme ensemble initial les contraintes de degré et les contraintes triviales.

Le but ici est de résoudre la relaxation linéaire de cette formulation PLNE. Notons x^* une solution optimale du PL limité aux contraintes initiales.

Le problème de séparation peut alors s'écrire de la façon suivante : déterminer une contrainte (7) violée par x^* si elle existe et dire sinon qu'il n'en n'existe pas.

Ce problème se ramène en fait à déterminer une coupe min dans le graphe G en utilisant pour capacité le vecteur x^* associée aux arêtes de G.
En effet, si l'on dispose d'une coupe $\delta(W^*)$ avec $W^* \subsetneq V$ et $W^* \neq \emptyset$, on a deux cas :
- soit $\sum_{e\in\delta(W)} x^*(e) < 2$ dans ce cas, on a déterminé la contrainte suivante qui est violée par x^*

$$\sum_{e\in\delta(W^*)} x(e) \geq 2$$

- soit $\sum_{e\in\delta(W)} x^*(e) \geq 2$ dans ce cas, toutes les contraintes (7) sont non violées.

Comme on sait déterminer une coupe-min avec des capacités positives (et x^* est un vecteur positif) en temps polynomial, on sait donc résoudre en temps polynomial cette formulation exponentielle.

Algorithme de Branch-and-Cut

A moins que $P = NP$, il semble impossible de posséder un algorithme de coupes pour résoudre un PLNE sans branchement.

Le couplage d'un algorithme de coupes et d'un branchement est appelé algorithme de coupes et branchement ou plutôt **Branch-and-Cut algorithm** :
- chaque nœud de l'arbre de branchement correspond à une relaxation résolue par une méthode de coupes : donc un nombre polynomial mais important de réitération de l'algorithme de coupes. La relaxation d'un nœud s'arrête quand il n'y a plus d'inégalités violées trouvées par l'algorithme.
- l'arbre de branchement suit le principe générique du branchement.

On peut noter que cet algorithme produit un grand nombre de points fractionnaires (qui seront coupés) : ils sont à un moment donné des points extrêmes de la formulation PL en cours de traitement. En utilisant une heuristique primale (rapide) pour chacun de ces points, on énumère un grand nombre de solutions entières... parmi elles il y a bien souvent la solution optimale !

Algorithme de Branch-and-Cut

En pratique, un algorithme de Branch-and-Cut nécessite un "framework logiciel"

- pour l'arbre de branchement
- pour la résolution des programmes linéaires (utilisant les solveurs linéraires puissants Cplex, LP,...) qui seront utilisés à chaque itération de l'algorithme de coupes, dans chaque nœud.
- pour l'assemblage du ou des algorithmes de séparation
- et d'autres astuces...

Plusieurs frameworks existent :
- au sein de Cplex et de Gurobi
- en logiciel Open Source SCIP
- et d'autres projets universitaires : Abacus, BCP de Coin-OR, Symphony,..

Qu'y-a-t-il dans les solveurs ?

Un solveur entier comme CPLEX, GUROBI, XPRESS, GLPK, SCIP,... est en fait la mise en œuvre d'un algorithme de Branch-and-Cut automatique.

- Les algorithmes de séparations sont issus d'inégalités génériques (comme celles de covers, de cycles impairs ou de cliques) à partir de la connaissance du PLNE compact fourni en entrée.
- L'arbre de branchement est l'arbre classique du Branch-and-Bound gérés très optimalement.
- Les plus puissants possèdent en plus une phase de prétraitement par inférence logique qui permettent de réduire la taille du problème
- et il y a de nombreuses techniques connues ou secrètes : le Strong Branching, les Local Cuts, les Lifting automatiques,... qui sont pour la plupart issues de recherche publiée par des chercheurs de tout horizon.