Master d'Informatique - Spécialité Androide
Module MAOA

Recherche Opérationnelle et Optimisation Combinatoire

Partie F - Génération de colonnes (Branch-and-Price)

Pierre Fouilhoux

Sorbonne Université

2019-2020

S

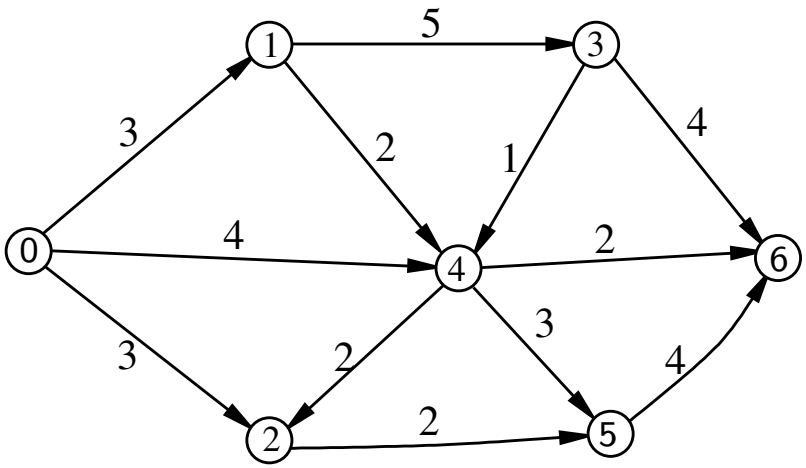
Un exemple : Le problème de multiflot-max

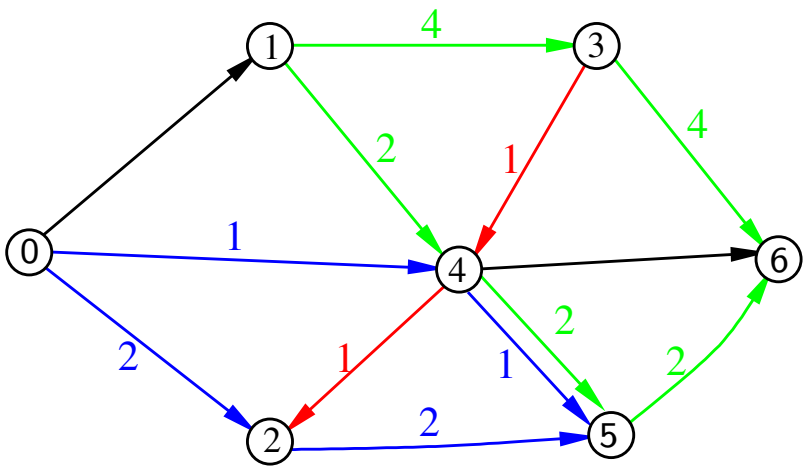
Soit $G = (V, A)$ un graphe orienté
et k **commodités**, c'est-à-dire des paires de sommets $(s_1, t_1), ..., (s_k, t_k)$ entre lesquelles doit circuler un flot.

Le réseau G est muni d'une capacité $b(a) \in \boldsymbol{N}$ associé à chaque arc $a \in A$.
Pour chaque paire $i \in \{1, ..., k\}$, on désire trouver des (s_i, t_i)-flots entiers tels que la somme des capacités utilisées par les flots sur un même arc respecte la capacité de cet arc.
L'objectif du **problème de multiflot-max** est de maximiser la quantité total de flot circulant sur le graphe.

Si flots continus, le problème de **multiflot-max fractionnaire** est polynomial.
Si flots entiers, le problème de **multiflot-max entier** est NP-difficile dès que $k \geq 2$.

Un exemple : Le problème de multiflot-max





Commodités : (0,5), (1,6), (3,2)

On peut noter que les trois flots reliant chacune des trois commodités sont en concurrence sur le graphe.
L'arc $(4, 5)$ porte à la fois du flot des commodités $(0, 5)$ et $(1, 6)$.
Notons que pour chaque nœud, il y a une conservation des flots individuelle (il n'y a pas de mélange des flots...).

Une première formulation naturelle

Considérons des variables $f^i(a)$ indiquant la quantité de flot passant par l'arc a pour desservir la commodité (s_i, t_i).

$$\begin{aligned}
\text{Max} \sum_{i=1}^{k} \sum_{u \in \delta^+(s_i)} & f^i(s_i u) \\
\sum_{v \in \delta^-(u)} f^i(vu) = \sum_{v \in \delta^+(u)} & f^i(uv) \quad \forall i \in \{1, \ldots, k\}, \forall u \in V, \\
\sum_{i=1}^{k} f^i(a) \leq b(a) & \quad \forall a \in A, \\
f^i(a) \geq 0 & \quad \forall i \in \{1, \ldots, k\}, \forall a \in A.
\end{aligned}$$

Ce problème est donc polynomial en utilisant un PL avec km variables et $kn + m$ inégalités.

Pour un graphe de 1000 sommets, 10000 arêtes et des commodités entre toute paire de sommets du graphe, cela représente un taille de PL trop importante pour être traitée par un solveur linéaire... mais il existe une reformulation bien plus efficace !

Reformulation

Une reformulation peut être obtenue en considérant d'autres variables.

Pour chaque commodité (s_i, t_i), $i \in \{1, \ldots, k\}$, considérons $\mathcal{P}_{s_i t_i}$ l'ensemble des chemins (élémentaires) allant du sommet s_i au sommet t_i.
Posons $\mathcal{P} = \bigcup_{i=1}^{k} \mathcal{P}_{s_i t_i}$.

Il est bien connu qu'un (s, t)-flot est un ensemble de chemins allant de s à t et portant une part du flot allant de s à t. L'idée est de chercher, pour chaque commodité, un tel ensemble de chemins.

On associe une variable f_μ associée à chaque chemin $\mu \in \mathcal{P}$.

Reformulation

La formulation $(\tilde{F})$ est un programme linéaire, dite **formulation arc-chemin**, qui modélise le problème de multiflot-max fractionnaire.

$$
(\tilde{F}) \quad
\begin{array}{lll}
\text{Max} \sum_{\mu \in \mathcal{P}} f_\mu & & \\
\sum_{\mu \in \mathcal{P} \mid a \in \mu} f_\mu \leq b(a) & & \forall a \in A \\
f_\mu \geq 0 & & \forall \mu \in \mathcal{P}.
\end{array}
$$

Si on ajoute en plus la contrainte d'intégrité, on obtient alors une formulation (F) qui est un PLNE modélisant le problème de multiflot-max entier.

$$
(F) \quad
\begin{array}{lll}
\text{Max} \sum_{\mu \in \mathcal{P}} f_\mu & & \\
\sum_{\mu \in \mathcal{P} \mid a \in \mu} f_\mu \leq b(a) & & \forall a \in A \\
f_\mu \geq 0 & & \forall \mu \in \mathcal{P}. \\
f_\mu \in \boldsymbol{N} & & \forall \mu \in \mathcal{P}.
\end{array}
$$

Notez donc que $(\tilde{F})$ est la relaxation linéaire de (F).

Principe de la génération de colonnes sur un exemple

La formulation PL arc-chemin

$$(\tilde{F}) \quad \begin{array}{lll} \text{Max} \sum_{\mu \in \mathcal{P}} f_\mu & & \\ \sum_{\mu \in \mathcal{P} \mid a \in \mu} f_\mu \leq b(a) & & \forall a \in A \\ f_\mu \geq 0 & & \forall \mu \in \mathcal{P}. \end{array}$$

possède seulement m inégalités... mais un nombre exponentel de variables !

Ce qui veut dire que même sa relaxation linéaire ne peut pas être directement obtenue par un solveur linéaire (algo du simplexe,...). On résoud la relaxation linéaire d'une telle formulation par **l'algorithme de génération de colonnes** (column generation, en anglais)

Rappel : Dualité

Pour un programme linéaire $(\tilde{P})$, appelé alors *primal*, le *dual* est le programme linéaire $(\tilde{D})$ suivant

$$(\tilde{P}) \begin{cases} \text{Max} & z = c^T x \\ & Ax \leq b \\ & x \geq 0 \end{cases} \qquad (\tilde{D}) \begin{cases} \text{Min} & w = b^T y \\ & A^T y \geq c \\ & y \geq 0 \end{cases}$$

Sous forme "algébrique" :

$$(\tilde{P}) \begin{cases} \text{Max} & z = \sum_{j=1}^{n} c_j x_j & \\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \forall i = 1, \ldots, m \\ & x_i \geq 0 & \forall j = 1, \ldots, n \end{cases} \qquad (\tilde{D}) \begin{cases} \text{Min} & w = \sum_{i=1}^{m} b_i y_i & \\ & \sum_{i=1}^{m} a_{ij} y_i \geq c_j & \forall j = 1, \ldots, n \\ & y_j \geq 0 & \forall i = 1, \ldots, m \end{cases}$$

Les **variables duales** de $(\tilde{D})$ correspondent aux inégalités de $(\tilde{P})$.
La matrice de $(\tilde{D})$ est la transposée A^T.
Les coûts de la fonction objective et les termes de droite des inégalités échangent leurs rôles.

Rappel : Dualité

• *Théorème (faible) de la dualité :*
Si $(\tilde{P})$ et $(\tilde{D})$ admettent chacun une solution $\tilde{x}$ et $\tilde{y}$,
alors $c^T\tilde{x} \leq b^T\tilde{y}$.

• *Theorème de la dualité :*
Si $(\tilde{P})$ admet une solution optimale (finie),
alors $(\tilde{D})$ aussi et de plus elles "coïncident", *i.e.*

$$(\tilde{P}) \quad max\{c^T x \mid Ax \leq b\} = min\{b^T y \mid y^T A = c, y \geq 0\} \quad (\tilde{D}).$$

• **Corollaire de la dualité :**
Soit x une solution de $(\tilde{P})$ et y une solution de $(\tilde{D})$. Alors
x et y sont solutions optimales de $(\tilde{P})$ et $(\tilde{D})$ **si et seulement si** $c^T x = b^T y$.

Principe de la génération de colonnes sur un exemple

Comme $(\tilde{F})$ est un programme linéaire, considérons son dual $(\tilde{D})$.
On note λ_a la variable duale associée à l'inégalité associée à un arc $a \in A$ de $(\tilde{F})$.

$$\text{Max} \sum_{\mu \in \mathcal{P}} f_\mu$$
$$\sum_{\mu \in \mathcal{P} \mid a \in \mu} f_\mu \leq b(a) \quad \forall a \in A$$
$$f_\mu \geq 0 \quad \forall \mu \in \mathcal{P}.$$

$$\text{Min} \sum_{a \in A} b(a)\lambda_a$$
$$\sum_{a \in \mu} \lambda_a \geq 1 \quad \forall \mu \in \mathcal{P},$$
$$\lambda_a \geq 0 \quad \forall a \in A.$$

En tant que transposé, le dual $(\tilde{D})$ possède alors un nombre exponentiel de lignes mais par contre un nombre fini de variables !

La condition d'optimalité donné par le corollaire de la dualité est la suivante :
Si l'on sait produire à la fois :
- une solution f^* du primal $(\tilde{F})$
- et une solution λ^* du dual $(\tilde{D})$
- qui ont même valeur objective,
alors f^* est optimale pour $(\tilde{F})$.

Principe de la génération de colonnes sur un exemple

Initialisation :
Dans ce cas simple de multiflot-max fractionnaire, il est facile d'avoir une solution réalisable.
En effet, pour chaque commodité (s_i, t_i), prenons un chemin quelconque μ_i^0 reliant s_i à t_i.
Considérons alors l'ensemble $\mathcal{P}_0$ de ces chemins.
Et les k variables f_μ associée aux chemins de $\mathcal{P}_0$.
Alors la formulation arc-chemin restreinte à ces variables admet une solution **et** cette solution est bien une solution du problème de multiflot-max fractionnaire (au pire les valeurs des flots véhiculés sont toutes nulles).

Considérons :
- le PL initial $(\tilde{F}_0)$ qui est la restriction de $(\tilde{F})$ aux k variables f_μ, $\mu \in \mathcal{P}_0$.
- son dual $(\tilde{D}_0)$.

Notons que $(\tilde{D}_0)$ est aussi la restriction de $(\tilde{D})$ aux k inégalités associées aux chemins $\mu \in \mathcal{P}_0$.

Principe de la génération de colonnes sur un exemple

Programme linéaire initial $(\tilde{F}_0)$ et son dual $(\tilde{D}_0)$.

$$\text{Max} \sum_{\mu \in \mathcal{P}_0} f_\mu$$
$$\sum_{\mu \in \mathcal{P}_0 \mid a \in \mu} f_\mu \leq b(a) \quad \forall a \in A$$
$$f_\mu \geq 0 \quad \forall \mu \in \mathcal{P}_0.$$

$$\text{Min} \sum_{a \in A} b(a)\lambda_a$$
$$\sum_{a \in \mu} \lambda_a \geq 1 \quad \forall \mu \in \mathcal{P}_0,$$
$$\lambda_a \geq 0 \quad \forall a \in A.$$

Résoudre le PL compact $(\tilde{F}_0)$ (et en même temps son dual $(\tilde{D}_0)$) est polynomial.

Notons f^0 la solution optimale de $(\tilde{F}_0)$
et λ^0 la solution optimale de $(\tilde{D}_0)$.

Remarques essentielles :
- λ^0 est un vecteur candidat à être solution de $(\tilde{D})$ tout entier.
- f^0 possède seulement k composantes parmi le nombre exponentiel possible dans $(\tilde{F})$.

Posons alors f^* le vecteur de $(\tilde{F})$ obtenu en fixant à 0 toutes les colonnes n'apparaissant pas dans $\mathcal{P}_0$: **alors f^* est une solution de $(\tilde{F})$!**

Principe de la génération de colonnes sur un exemple

On peut déjà tester l'optimalité de f^* :
En effet, par dualité forte, on sait que les fonctions objective de f^0 et λ^0 sont identiques pour $(\tilde{F}_0)$ et pour $(\tilde{D}_0)$.
De plus, par construction (ajout des valeurs nulles), la fonction objective pour f^0 dans $(\tilde{F}_0)$ est la même que celle de * dans $(\tilde{F})$.
Donc le corollaire de la dualité nous dit que :
Si λ^0 est solution de $(\tilde{D})$, alors f^* est optimale pour $(\tilde{F})$.

Or tester si λ^0 est solution de $(\tilde{D})$ revient à tester si le vecteur λ^0 vérifie les contraintes du dual, c'est-à-dire si

$$\sum_{a \in \mu} \lambda_a \geq 1 \quad \forall \mu \in \mathcal{P}$$

Cette condition est donc vérifiée si le vecteur λ^0 vérifie ce lot d'inégalités qui est en nombre exponentiel : autant qu'il y a de chemins !
On appelle ce test : le problème de "pricing" (ou en français le problème de génération d'une colonne).

Principe de la génération de colonnes sur un exemple

Algorithme de génération de colonnes :
- Si le problème de pricing renvoie "vraie" : alors vecteur dual λ^0 vérifie les inégalités et la solution primale f^* est optimale.
- Sinon (le problème de pricing renvoie "faux") : alors il existe une inégalité violée par λ^0 : cette inégalité correspond à un chemin $\mu \in \mathcal{P}$. On ajoute alors la variable f_μ au PL maître

$$(\tilde{F}_{i+1}) \leftarrow (\tilde{F}_i) \text{ plus la variable } f_\mu$$

Et on recommence pour $(\tilde{F}_{i+1})$!
C'est le principe de génération de colonnes (ou de variables).

Principe de la génération de colonnes sur un exemple

Validité de l'algorithme :
L'algorithme, dans le pire des cas, va ajouter toutes les variables possibles : il est donc trivialement valide.

Complexité de l'algorithme :
Il s'agit de la même question que la complexité de la méthode de coupes :
en effet, le problème de pricing est le problème de séparation pour le dual de la formulation !
Donc par le résultat de Groëtschel-Lovasz-Schrivjer :
si le problème de pricing se résoud en temps polynomial (*i.e.* si λ^0 vérifie les inégalités), alors la réitération du problème de pricing dans une méthode de génération de colonnes est polynomiale !

Principe de la génération de colonnes sur un exemple

Le problème de pricing de cette génération de colonnes pour le problème de multiflot est de tester les inégalités du dual pour chaque chemin de $\mathcal{P}$, c'est-à-dire si

$$\sum_{a \in \mu} \lambda'_a \geq 1 \quad \forall \mu \in \mathcal{P}.$$

C'est-à-dire
Tester s'il existe un chemin μ correspondant à une contrainte violée et répondre VRAI en produisant ce chemin
Et sinon répondre FAUX.

On recherche donc un algorithme capable de résoudre le problème de pricing :
un tel algorithme est appelé en anglais pricer.

Principe de la génération de colonnes sur un exemple

Ici ce problème de pricing se ramène à déterminer, pour chaque paire de commodités (s_i, t_i), un plus court chemin de s_i à t_i selon les poids λ^0 associés aux arcs de A :
Si le plus court de ces chemins est de valeur ≥ 1, alors toutes les inégalités du dual sont vérifiées : le problème de pricing renvoie FAUX.
Sinon, le plus court de ces chemins viole l'inégalité et ce chemin μ^* est retourné par le problème de pricing.
Dans ce dernier cas, on crée la variable f_{μ^*} et on l'ajoute au problème maître.

Comme les coûts duaux λ^0 sont positifs, la recherche d'un plus court chemin est polynomial (par exemple avec l'algorithme de Dijkstra $O(mlog(n))$, le problème de pricing peut être résolu en $O(kmlog(n))$: un tel algorithme résoud bien le problème de pricing (c'est un pricer).
Comme ce pricer est polynomial, sa réitération dans une génération de colonnes est alors polynomiale !

La formulation arc-chemin pour le problème de multiflot-max fractionnaire peut donc être résolue en temps polynomial !

Principe de la génération de colonnes sur un exemple

On peut noter que cette méthode de génération de colonnes suit le principe de l'algorithme du simplexe : entrée d'une variable en base en fonction des coûts réduits.

La méthode de génération de colonnes suit ainsi les bonnes propriétés observées (et par encore tout à fait bien comprises) de l'algorithme du simplexe comme le fait de rarement nécessiter beaucoup d'itérations (et donc de variables).

Malheureusement, la méthode de génération de colonnes suit aussi les mauvaises propriétés de l'algorithme du simplexe : bien souvent il s'agit d'itérations fortement dégénérées où plusieurs bases donnent la même solution ! Et il est plus difficile que dans le cas du simplexe de faire converger la méthode : des outils d'optimisation continue sont souvent nécessaires.

Dans le cas du problème de multiflot fractionnaire, la convergence est rapide et cette formulation est très efficace, bien plus efficace que la formulation naturelle avec un nombre polynomial mais très élévée de variables.

Le problème de multiflot de coût minimum

Le problème de multiflot-max considéré jusqu'ici est une version simplifiée du problème classique de multiflot.

Soit $G = (V, A)$ un graphe orienté
et k **commodités**, c'est-à-dire des triplets $(s_1, t_1, d_1), ..., (s_k, t_k, d_k)$ où (s_i, t_i) est une paire de sommets entre lesquelles doit circuler un flot de valeur au moins d_i.

Le réseau G est muni :
- d'une capacité $b(a) \in \mathbb{N}$ associé à chaque arc $a \in A$
- et d'un coût unitaire (positif) d'utilisation des arcs $c(a)$.

L'objectif du **problème de multiflot** (de coût minimum) est de déterminer k flots reliant les commodités avec un coût total minimal.

Ce problème de multiflot est est un modèle assez réaliste pour le routage des données entre les paires de clients voulant communiquer dans un réseau de télécommunications.

Si flots continus, le problème de **multiflot fractionnaire** est polynomial.
Si flots entiers, le problème de **multiflot entier** est NP-difficile dès que $k \geq 2$.

Une première formulation naturelle

Commme pour le multiflot-max, on peut considérer des variables $f^i(a)$ indiquant la quantité de flot passant par l'arc a pour desservir la commodité (s_i, t_i).

$$\text{Min} \sum_{i=1}^{k} \sum_{a \in A} c(a) f^i(a)$$

$$\sum_{v \in \delta^-(u)} f^i(vu) = \sum_{v \in \delta^+(u)} f^i(uv) \qquad \forall i \in \{1, \ldots, k\}, \forall u \in V \setminus \{s_i, t_i\},$$

$$\sum_{v \in \delta^-(s_i)} f^i(vs_i) + d_i = \sum_{v \in \delta^+(s_i)} f^i(s_i v) \qquad \forall i \in \{1, \ldots, k\},$$

$$\sum_{v \in \delta^-(t_i)} f^i(vt_i) = \sum_{v \in \delta^+(t_i)} f^i(t_i v) + d_i \qquad \forall i \in \{1, \ldots, k\},$$

$$\sum_{i=1}^{k} f^i(a) \leq b(a) \qquad \forall a \in A,$$

$$f^i(a) \geq 0 \qquad \forall i \in \{1, \ldots, k\}, \forall a \in A.$$

Ce problème est donc polynomial par ce PL mais sa taille est bien trop importante pour être traitée par un solveur entier.

Reformulation

On utilise la même reformulation $(\tilde{F})$ qui est un programme linéaire, dit **formulation arc-chemin**, qui modélise le problème de multiflot fractionnaire

$$
(\tilde{F}) \qquad
\begin{array}{lll}
\text{Min} \sum_{\mu \in \mathcal{P}} c(\mu) f_\mu & & \\
\sum_{\mu \in \mathcal{P} \mid a \in \mu} f_\mu \leq b(a) & & \forall a \in A \\
\sum_{\mu \in \mathcal{P}_{s_i t_i}} f_\mu = d_i & & \forall i \in \{1, \ldots, k\} \\
f_\mu \geq 0 & & \forall \mu \in \mathcal{P}.
\end{array}
$$

où la notation $c(\mu)$ désigne le coût total des arcs du chemin μ : $c(\mu) = \sum_{a \in \mu} c(a)$.

Si on ajoute en plus la contrainte d'intégrité, on obtient alors une formulation (F) qui est un PLNE modélisant le problème de multiflot entier.

Pricing

Comme $(\tilde{F})$ est un programme linéaire, considérons son dual $(\tilde{D})$.
Posons λ_a la variable duale associée à l'inégalité associée à un arc $a \in A$ de $(\tilde{F})$,
et ω_i la variable duale associée à l'inégalité associée à une commodité $i \in \{1, \ldots, k\}$.

$$
(\tilde{F}) \quad
\begin{array}{ll}
\text{Min} \displaystyle\sum_{\mu \in \mathcal{P}} c(\mu) f_\mu & \\
\displaystyle\sum_{\mu \in \mathcal{P} \mid a \in \mu} f_\mu \leq b(a) & \forall a \in A \\
\displaystyle\sum_{\mu \in \mathcal{P}_{s_i t_i}} f_\mu = d_i & \forall i \in \{1, \ldots, k\} \\
f_\mu \geq 0 & \forall \mu \in \mathcal{P}
\end{array}
$$

$$
(\tilde{D}) \quad
\begin{array}{ll}
\text{Max } \textit{(fonction objective inutile)} & \\
\displaystyle\sum_{a \in \mu} \lambda_a + \omega_i \leq c(\mu) & \forall i \in \{1, \ldots, k\}, \forall \mu \in \mathcal{P}_i \\
\lambda_a \leq 0 & \forall a \in A \\
\omega_i \in \boldsymbol{R} & \forall i \in \{1, \ldots, k\}
\end{array}
$$

Initialisation

Pour appliquer le principe précédent, il faut obtenir un ensemble de chemins $\mathcal{P}'$ tel que la **formulation restreinte** $\tilde{F}_{\mathcal{P}'}$ possède une solution...
C'est **l'étape d'initialisation de la génération de colonnes**.

Or déterminer une solution réalisable pour ce problème de multiflot n'est pas évident (bien que polynomial). Il y a plusieurs façons de procéder :

- soit déterminer heuristiquement une solution réalisable (mais cela n'est pas garanti de fonctionner)
- soit déterminer heuristiquement beaucoup de chemins (jusqu'à ce que cela soit réalisable)
- soit démarrer "à vide" c'est-à-dire partir d'une formulation avec seulement un chemin par commodité (donc pas forcément réalisable) : dans ce cas, un primal non réalisable correspondn à des coûts réduits infinis... On peut par contre utiliser le pricer classique avec des coûts particuliers, dit coût de Farkas, (ils proviennent du Lemme de Farkas (voir cours sur l'approche polyédrale) : ces coûts guident vers une solution réalisable.

Pricing et Pricer

Supposons à présent que l'on a trouvé un ensemble $\mathcal{P}'$ de chemins tels que la formulation $\tilde{F}_{\mathcal{P}'}$ ait une solution.
Le problème de pricing est de tester s'il existe un chemin $\mu \in \mathcal{P}$ tel que

$$\sum_{a \in \mu} \lambda_a + \omega_i \leq c(\mu) \quad \forall i \in \{1, \ldots, k\}, \forall \mu \in \mathcal{P}_i$$

Une façon efficace de répondre à ce problème est :
- d'effectuer l'étape suivant pour chaque $i \in \{1, \ldots k\}$
- de se rappeler que $c(\mu) = \sum_{a \in \mu} c(a)$

Alors l'inégalité s'écrit

$$\sum_{a \in \mu} (c(a) - \lambda_a) \geq \omega_i$$

où ω_i est une donnée fixe pour chaque itération i.

Pricing et Pricer

Posons alors une longueur auxiliaire associé aux arcs du graphe

$$l_a = c(a) - \lambda_a \quad \forall a \in A$$

Comme $\lambda_a \leq 0$, alors cette longueur est positive ! !

Le pricer revient à nouveau ici, pour chaque commodité i, à recherche un plus court chemin entre s_i et t_i selon le poids l_a.
Si cette plus courte longueur est supérieur à d_i, alors il n'existe aucune variable améliorant le problème maître dans $\mathcal{P}_i$. Sinon on a déterminer une variable à ajouter (on peut en ajouter plusieurs à la fois).

Ce pricer est donc polynomial et la résolution de la formulation arc-chemin pour le problème du multiflot fractionnaire est donc aussi polynomiale (et très efficace).

Problème de multiflot entier

Le problème de multiflot entier est en revanche NP-difficile.
Sa formulation arc-chemin est un PLNE :

$$
(F) \qquad
\begin{array}{lll}
\text{Min} \sum_{\mu \in \mathcal{P}} c(\mu) f_\mu & & \\
\sum_{\mu \in \mathcal{P} \mid a \in \mu} f_\mu \leq b(a) & & \forall a \in A \\
\sum_{\mu \in \mathcal{P}_{s_i t_i}} f_\mu = d_i & & \forall i \in \{1, \ldots, k\} \\
f_\mu \geq 0 & & \forall \mu \in \mathcal{P} \\
f_\mu \in \boldsymbol{N} & & \forall \mu \in \mathcal{P}
\end{array}
$$

dont la relaxation linéaire est $(\tilde{F})$ est celle du multiflot fractionnaire.
Donc la relaxation linéaire de ce PLNE peut être résolue en temps polynomial par un algorithme de génération de colonnes.

Il est donc possible d''utiliser cet algorithme de génération de colonnes pour résoudre chacun des nœuds d'un arbre de branchement sur les variables f_μ : un tel algorithme est appelé **algorithme de génération de colonnes et branchement** ou **Branch-and-Price algorithm**, en anglais.

Problème de multiflot entier

La règle classique de branchement sur les variables entières :
- sélectionner une variable fractionnaire f_μ parmi celles non-entière de la relaxation $\tilde{F}$.
- prendre la partie entière inférieure v de f_μ
- créer deux sous-problèmes fils, l'un où $f_\mu \leq v$ et l'autre avec $f_\mu \geq v + 1$

Cette règle n'est pas très efficace pour cette formulation (F) : en effet, il existe de nombreux chemins possibles, certains très proches les uns des autres à une arête près.

De nombreuses règles de branchement plus efficaces ont été tentées... passant souvent par réécrire tout ou partie la formulation pour avoir un branchement efficace.

Une des difficultés est que l'ajout d'inégalités dans le problème maître conduit à créer de nouvelles variables duales à prendre en compte dans le coût dual de la colonne à calculer...

Nous verrons plus loin le cas d'un branchement complet pour le problème de coloration.

Notations

Considérons une formulation PLNE décrite de la façon suivante :

$$(P) \quad \begin{array}{ll} \text{Min} \sum_{s \in \mathcal{S}} c(s) t_s & \\ \sum_{s \in \mathcal{S}} \chi_k^s t_s \geq b_k & \forall k \in \mathcal{K} \\ t_s \in \boldsymbol{N} & \forall s \in \mathcal{S} \end{array}$$

où

- $\mathcal{S}$ est l'ensemble des variables entières de la formulation (potentiellement en nombre exponentiel).
- $c(s)$ est le coût d'une variables.
- $\mathcal{K}$ est un ensemble d'inégalités (a priori en nombre polynomial).
- pour une contrainte k donnée, χ_s^k est le coefficient de la varuble t_s pour cette inégalité : notez que le vecteur colonne χ_s correspond ainsi à un vecteur décrivant la variable t_s.

Algorithme de génération de colonnes

L'algorithme de génération de colonnes permet de résoudre la relaxation linéaire $(\tilde{P})$ de (P) peut-être décrit comme suit

0- Partir d'un ensemble $\mathcal{S}_0$ de variables tel que la formulation PL $(\tilde{P}_0)$ restreinte à $\mathcal{S}_0$ ait une solution.

$\mathcal{S}^* \leftarrow \mathcal{S}_0$

1- Résoudre le PL compact $(\tilde{P}^*)$ restreint aux variables de $\mathcal{S}^*$: soit t^* la solution optimale de $(\tilde{P}^*)$ et λ_k^* le coût dual associé à l'inégalité $k \in \mathcal{K}$

2- Tester s'il existe une variable t_s dans $\mathcal{S} \setminus \mathcal{S}^*$ telle que son **coût réduit** soit strictement négatif (c'est le problème de pricing) :

$$\bar{c}(s) = c(s) - \sum_{k \in \mathcal{K}} \chi_k^s \lambda_k^*$$

3- S'il n'en existe pas alors STOP : t^* est solution fractionnaire optimale pour $(\tilde{P})$

4- Sinon $\mathcal{S}^* \leftarrow \mathcal{S}^* \cup \{t_s\}$ et GOTO 1

Problème de pricing

Formellement le problème de pricing consiste donc à :
- produire une colonne de coût réduit $\bar{c}(s) = c(s) - \sum_{k \in \mathcal{K}} \chi_k^s \lambda_k^*$ strictement négatif
- ou de prouver que toutes les colonnes ont un coût positif ou nul.

Très souvent pour répondre au problème de pricing, on recherche
la colonne de coût réduit minimal

$$\text{i.e. } \text{Min}_{s \in \mathcal{S}} \quad \bar{c}(s) = c(s) - \sum_{k \in \mathcal{K}} \chi_k^s \lambda_k^*$$

en effet :
- soit cette colonne est de coût réduit positif ou nul (et on a répondu négativement au problème de pricing)
- soit cette colonne est de coût réduit négatif, et on a produit la colonne demandée.

Rq : une colonne déjà dans le problème restreint a toujours un coût réduit positif ou nul.
En effet, le PL restreint a été résolu à l'optimum et toutes les variables du PL sont ainsi de coût positif ou nul à l'optimum par le critère d'optimalité de l'algorithme du simplexe.

Ainsi en ajoutant des variables de coûts réduits strictement négatif, on ajoute toujours de nouvelles colonnes.

Algorithme de Branch-and-Price

Pour résoudre la formulation PLNE (P) il faut utiliser un arbre de branchement où la relaxation linéaire de chacun des PL associés aux nœuds est résolu par l'algorithme de génération de colonnes : c'est **l'algorithme Branch-and-Price**.

En général le branchement est difficile à mettre en œuvre et peu efficace :
En effet :

- le fait de brancher implique souvent d'adapter le problème de pricing au sous-problème : en effet, si l'on conserve le même algorithme de pricing, il peut produire des colonnes interdites dans la branche de l'arbre.
  Une solution (peu performante) est de conserver dans un nœud la liste des colonnes interdites et de rechercher une colonne hors de cette liste
- le branchement classique sur les variables $t_S = 0$ / $t_S = 1$) est très peu efficace car la branche $t_S = 0$ peut representer 99.999% du nœud père...
- un branchement sur des inégalités implique de prendre en compte ces inégalités de branchement dans le problème maître, induisant alors de nouvelles variables duales...

Heureusement, il existe de nombreux cas où l'on peut brancher en se basant sur des propriétés du problème et de la décomposition ayant mené à la génération de colonnes (voir l'exemple de la règle de Ryan&Foster en fin de chapitre).

"Pricer" sans "Brancher"

Les méthodes de génération de colonnes limitées à la résolution du PL, que l'on appelle souvent "le" pricing peuvent aussi s'utiliser de manière utile sans brancher.
Attention : ce sont alors rarement des méthodes exactes, sauf si elles réussissent à produire une borne inférieure = à une borne supérieure (ce qui est rare.)

En tant que méthodes approchées, le pricing est utile car :

- il produit des bornes inférieures intéressantes pour un problème (problème d'ordonnancement, de coloration,...) par sa relaxation (voir section Décompostion de Dantzig-Wolfe)
- les colonnes produites lors du pricing peuvent suffirer à produire une très bonne solution heuristique

On obtient alors un encadrement obtenu expérimentalement entre une borne inférieure et une borne supérieure associée à une solution heuristique.

Heuristique

Plusieurs techniques heuristiques existent pour produire des solutions entières à partir d'un lot de colonnes générées lors du pricing

- par arrondi des valeurs fractionnaires des colonnes
- par analyse "statistique" des colonnes générées :
  on regarde les composantes des colonnes les plus utilisées dans une solution optimales fractionnaires : dans l'exemple du multiflot c'est repérer les arêtes les plus utilisées par un flot.
  Cela "pilote" la construction de solutions par une heuristique gloutonne
- on peut aussi à la fin du pricing, ne pas brancher et résoudre le PLNE formé par les colonnes générées jusqu'ici. On obtient bien une solution entière, souvent de très bonne valeur : cette méthode s'appelle **Stop&Solve** ou **Price&Branch**.
- ...

Il est à noter que ces techniques, plus ou moins consommatrices de temps peuvent s'utiliser après chaque pricing, à la fin du pricing ou alors en stoppant le pricing au bout d'un temps fixe etc.

Outils et mise en œuvre

La mise en œuvre des méthodes de Branch-and-Price est assez complexe :
- gestion des variables qui, même en nombre polynomial, deviennent trop importantes pour la résolution des relaxation (suppression en foncton d'une évaluation de leur activité)
- convergence de la génération de colonnes
- utilisation d'un ou plusieurs pricer pour ne pas systématiquement utiliser un pricer trop lent (utilisation de pricer heurisitique)
- réaliser des compromis entre résoudre la relaxation linéaire et accélérer le branchement
- règle et stratégie de branchement efficaces...

Si l'on désire résoudre un PLNE avec un nombre exponentiel de variables, il existe des frameworks
- commerciaux : Concert technology de Cplex-IBM ou Gurobi.
- universitaires : SCIP de la ZIB, BCP de COIN-OR, Symphony de COIN-OR
- un produit universitaire BapCode de l'Université de Bordeaux tente même de mettre en œuvre automatiquement une méthode de décomposition (dite de Dantzig-Wolfe).
- on peut avoir besoin de gérer des formulations avec un nombre exponentiel de variables et de contraintes ! Il faut donc un algorithme de **Branch-and-cut-and-Price (BCP)** : il n'existe là pratiquemment que les outils universitaires SCIP de la ZIB et BCP de COIN-OR.

Convergence d'une génération de colonnes

On observe expérimentalement

- une convergence des méthodes de génération de colonnes souvent très lente.

- Dans un contexte où il y a théoriquement un nombre exponentiel de colonnes possibles,

  il peut y avoir un nombre important de colonnes ajoutées par la procédure,
- Parfois l'amélioration de la valeur courante est nulle ou minuscule : c'est particulièrement le cas après un bon nombre d'itérations.

- les valeurs des coûts réduits des colonnes générées fluctuent énormément.

Retour sur l'algorithme du simplexe

Une itération de la génération de colonnes peut être vue comme une itération de l'algorithme du simplexe
En effet :

- résoudre le PL restreint fournit des valeurs aux variables duales optimales λ^* du dual restreint qui ne sont pas réalisables pour le dual complet (sauf à la fin)
- la formule $\bar{c}(s) = c(s) - \sum_{k \in \mathcal{K}} \chi_k^s \lambda_k^*$ donne les coûts réduits des variables de cette étape d'une itération du simplexe
  où les colonnes non-encore générées sont nécessairement hors-base
  et les colonnes déjà générées ont un coût réduit positif ou nul

Retour sur l'algorithme du simplexe

Vu comme une itération de l'algorithme du simplexe, la méthode de génération de colonne a pour but est de générer les colonnes nécessaires pour avoir une base (réalisable) parmi les colonnes générées et avec toutes les colonnes non-générées de coût réduit strictement négatif : ce qui prouve alors l'optimalité de la solution.

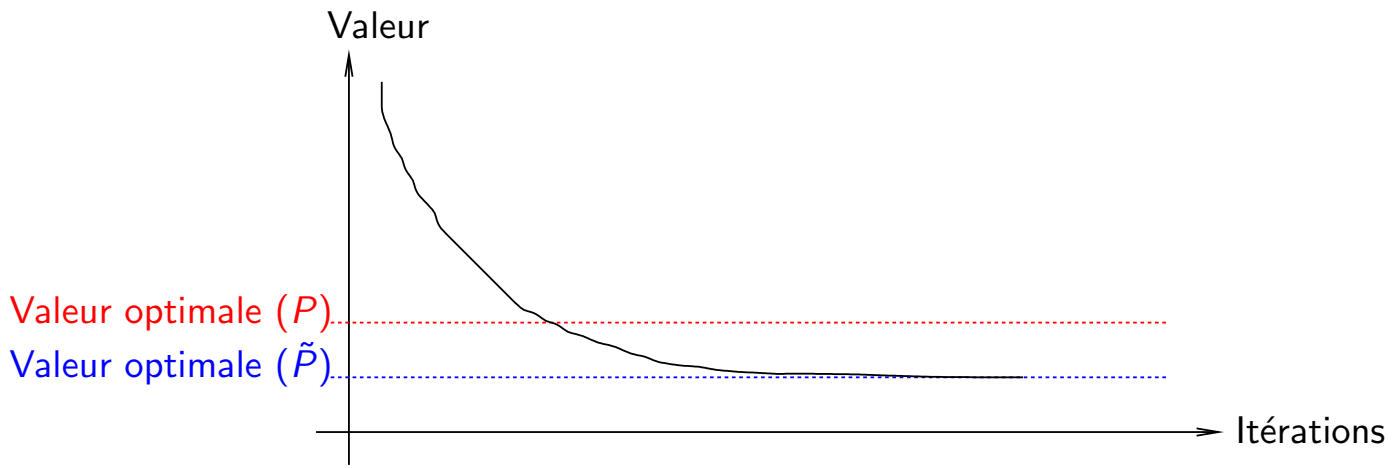
Or dans les cas étudiés de génération de colonnes à nombre exponentiel de colonnes... le dual est bien souvent fortement dégénéré.
C'est-à-dire que les inégalités (correspondantes aux variables du dual) sont très proches linéairement les unes des autres, provoquant de nombreuses bases possibles ayant même valeur.

Plusieurs techniques ont été proposée pour sortir ces itérations dégénérées :
- les méthodes de faisceaux (bundle méthod)
- les méthodes In/Out,...
On les nomme en général **méthodes de stabilisation**.

Produire des bornes inférieures

Cette lenteur de convergence produit une difficulté qui n'apparaît pas en génération d'inégalités (Branch&Cut) est la difficulté d'obtenir des bornes inférieurs

En effet, regardons ce schéma où la courbe suit l'amélioration de la valeur de la meilleure valeur connue de la fonction objective pendant une génération de colonnes : on voit que la courbe tend peu à peu vers la solution optimale fractionnaire de la relaxation linéaire $(\tilde{P})$.



Comme la valeur optimal de (P) est au-dessus de la valeur optimale de $\tilde{P}$, seule la valeur optimale de $(\tilde{P})$ fournit à coup sûr une borne inférieure à (P).

Un cas particulier de borne inférieure

Dans le cas particulier où on possède B une borne supérieure telle que $B \geq \sum_{s \in \mathcal{S}} t_S$
(Si $t_s \in \{0, 1\}$, la borne B est le nombre maximum de variables dans une solution).
Notons :
- z_P la valeur optimale d'une solution entière de (P)
- $z_{\tilde{(P)}}$ la valeur optimale de la relaxation $(\tilde{P})$
- z_{iter} la valeur obtenue pour une itération quelconque
- c^*_{min} la plus petit coût réduit parmi toutes les variables :
  i.e. $c^*_{min} = \min_{s \in \mathcal{S}} c(s) - \sum_{k \in \mathcal{K}} \chi^s_k \lambda^*_k$

Alors

$$z_{iter} + B c^*_{min} \leq z_{\tilde{P}} \leq z_P$$

Produire des bornes inférieures

Preuve : Notons $(e_k)_{k\in\mathcal{K}}$ les variables d'écarts de $(\tilde{P})$. Par définition des coûts réduits, on a

$$\sum_{s\in\mathcal{S}} c(S)t_S = z_{iter} + \sum_{s\in\mathcal{S}} \bar{c}(S)t_S + \sum_{k\in\mathcal{K}} \bar{c}(e_k)e_k$$

Rappelons que les coûts réduits des variables d'écarts sont (en minimisation) égales aux valeurs des variables duales : $\bar{c}(e_k)e_k = \lambda_k^* e_k$ et comme $\lambda_k^* \geq$ et $e_k \geq 0$ donc

$$\sum_{s\in\mathcal{S}} c(S)t_S \geq z_{iter} + \sum_{s\in\mathcal{S}} \bar{c}(S)t_S$$

or $c^*_{min} \leq \bar{c}_S$ pour tout $S \in \mathcal{S}$ et $c^*_{min} < 0$ à une itération non terminale (donc $c^*_{min} \sum_{s\in\mathcal{S}} t_S \geq Bc^*_{min}$)

$$\sum_{s\in\mathcal{S}} c(S)t_S \geq z_{iter} + \sum_{s\in\mathcal{S}} c^*_{min}t_S = z_{iter} + c^*_{min}\sum_{s\in\mathcal{S}} t_S \geq z_{iter} + Bc^*_{min}$$

Et ceci est vrai pour tout solution t y compris la solution optimale. □

Produire des bornes inférieures

Il existe d'autres façons d'obtenir des bornes inférieures de bonne qualité mais elles sont souvent spécifiques à des PL ($\tilde{P}$) particulier comme de problèmes de partitionnement (set partitionning problem) ou de couverture (set covering problem) :

- la technique du dual ascent (qui est aussi plus rapide que résoudre la relaxation linéaire par l'algorithme du simplexe)
- des techniques avancées basées sur des propriétés de fonctions comme les dual-feasible function (DDF) ou des fonctions superadditives

Principe général

Soit $c \in \mathbf{R}^n$ un vecteur-colonne, $A \in \mathbf{R}^{n \times m}$, $a \in \mathbf{R}^m$, $B \in \mathbf{R}^{n \times m'}$ et $b \in \mathbf{R}^{m'}$. On considère le PLNE (P) suivant

$$(P) \begin{cases} \text{Min } c^T x \\ \quad Ax \geq a \\ \quad Bx \geq b \\ \quad x \in \mathbf{N}^n \end{cases}$$

où :
- les inégalité $Ax \geq a$ sont "couplantes", c'est-à-dire qu'elles concernent une bonne partie des variables
- les $Bx \geq b$ sont "décomposables".

On dit que l'ensemble d'inégalités $Bx \geq b$ est **décomposable** si le problème

$$(SP) \begin{cases} \text{Min } d^T x \\ \quad Bx \geq b \\ \quad x \in \mathbf{N}^n \end{cases}$$

(avec d ici un poids quelconque) est la juxtaposition de plusieurs problèmes sur des espaces de variables indépendantes.

Principe général

Le problème (SP) peut être reformulé différemment en utilisant une description dite "par les solutions".

On considère pour cela l'ensemble Q des solutions de (SP) : c'est-à-dire tous les points entiers du polyèdre $Bx \geq b$ (cet ensemble est en général exponentiel sur la taille du problème !)

$$Q = \{x \in \mathbf{N}^n \mid Bx \geq b\}$$

Supposons ici que Q soit fini, c'est-à-dire que $conv(Q)$ soit un polytope : on peut alors noter $Q = \{\chi^1, ..., \chi^{|Q|}\}$ ces solutions qui sont donc des vecteurs entiers.

Remarquons qu'alors les solutions de Q sont données par

$$\sum_{q=1}^{|Q|} \chi^q t_q \quad \text{avec} \quad \sum_{q=1}^{|Q|} t_q = 1$$

où t_q sont des variables binaires.

Principe général

Déterminer une solution de (SP) revient donc à choisir un vecteur χ dans Q, en d'autre terme cela revient à réécrire (SP) de la façon suivante :

$$(SP)\begin{cases} \text{Min } d^T x & \\ x = \sum_{q=1}^{|Q|} \chi^q t_q & \\ \sum_{q=1}^{|Q|} t_q = 1 & \\ t_q \in \{0,1\} & \forall 1 \in \{1,\ldots,|Q|\}. \end{cases}$$

Cette écriture "naïve" revient à énumérer toutes les solutions et donner à x le vecteur la meilleure d'entre elles : on peut noter que les variables t sont ici des valeurs binaires permettant de définir quels vecteurs choisir dans Q.

Principe général

Remarquons que l'on pourrait ici prendre des variables t réelles pour définir (SP) : on appelle cela la convexification.

$$(SP)\begin{cases} \text{Min } d^T x & \\ x = \sum_{q=1}^{|Q|} \chi^q t_q & \\ \sum_{q=1}^{|Q|} t_q = 1 & \\ t_q \geq 0 & \forall 1 \in \{1, \ldots, |Q|\}. \end{cases}$$

En effet, le polyèdre associé est naturellement entier car chacun de ses points extrêmes correspond à exactement à un élément de Q (c'est-à-dire une solution de (SP)).

Principe général

Utilisons cette reformulation de (SP) dans (P) :

$$(P)\begin{cases} \text{Min } c^T x \\ \quad Ax \geq a \\ \quad x = \sum_{q=1}^{|Q|} \chi^q t_q \\ \quad \sum_{q=1}^{|Q|} t_q = 1 \\ \quad t_q \in \{0,1\} \qquad \forall 1 \in \{1,\ldots,|Q|\}. \end{cases}$$

Et faisons alors disparaître x !

$$(P)\begin{cases} \text{Min } \sum_{q=1}^{|Q|} (c^T \chi^q) t_q \\ \quad \sum_{q=1}^{|Q|} (A\chi^q) t_q \geq a \\ \quad \sum_{q=1}^{|Q|} t_q = 1 \\ \quad t_q \in \{0,1\} \qquad \forall 1 \in \{1,\ldots,|Q|\}. \end{cases}$$

Principe général

On définit la **Relaxation de Dantzig-Wolfe** comme la relaxation fractionnaire (MP) de (P) ainsi définie :

$$(MP)\begin{cases} \text{Min} \sum_{q=1}^{|Q|}(c^T\chi^q)t_q \\ \sum_{q=1}^{|Q|}(A\chi^q)t_q \geq a \\ \sum_{q=1}^{|Q|} t_q = 1 \\ t_q \geq 0 \qquad\qquad \forall 1 \in \{1,\ldots,|Q|\}. \end{cases}$$

Cette relaxation n'est pas la relaxation linéaire classique de (P).

Il est important de remarquer que (MP), appelé *problème maître*, est un PL mais où le fait que les solutions de (SP) soit entière a été en partie pris en compte.

Principe général

La formulation (MP) contient par construction un nombre exponentiel de variables autant que de solutions dans Q !

Afin de résoudre (MP), on peut alors mettre en place un algorithme de génération de colonnes.

En posant π le vecteur dual des inégalités provenant de $Ax \geq a$ et ρ la valeur duale provenant de la contrainte d'égalité, le problème de pricing revient à rechercher s'il existe une solution de $\chi^q \in Q$ telle que son coût réduit est négatif, *i.e.*,

$$(c - \pi^T A)\chi^q - \rho < 0$$

Ainsi le pricer revient à résoudre le problème (SP) avec le coût $d = (c - \pi^T A)$: si ce poids est négatif strictement, on ajoute une colonne χ^q à Q. Et on réitère cela jusqu'à ce que (SP) renvoie une plus petite solution de coût réduit positif : ce qui signifie que la solution du master restreint est une solution optimale de (MP).

Principe général

Le pricer peut donc se résoudre par PLNE en utilisant :

$$(SP)\begin{cases} \text{Min } (c - \pi^T A)^T x \\ Bx \geq b \\ x \in \boldsymbol{N}^n \end{cases}$$

On peut alors voir (SP) comme le sous-problème de la méthode de Dantzig-Wolfe.

Bien entendu, il est tout à fait possible de résoudre le problème de pricing par d'autres moyens algorithmiques, surtout s'ils sont plus efficaces.

Relaxation de Dantzig-Wolfe

Comme l'on cherche à résoudre un PLNE (P), cette génération de colonnes ne mène qu'à une relaxation fractionnaire (MP) de ce PLNE, dite **relaxation de Dantzig-Wolfe**.

En général, cette relaxation (MP) est meilleure que la relaxation linéaire du PLNE (P) !
De nombreuses bornes ont été fournies par cette méthode pour des problèmes d'optimisation combinatoire.

De plus un algorithme de Branch-and-Price basée sur cette décomposition de Dantzig-Wolfe est souvent efficace.

Amélioration de la relaxation linéaire ?

En fait, si la relaxation (MP) de Dantzig-Wofle est plus intéressante en général que la relaxation linéaire de (P), il existe de nombreux cas où les deux relaxations sont identiques !

C'est le cas dans ce cadre très fréquent :

Theorem (Geoffrion)

Si (SP) est entièrement décrit par $conv(x \in \mathbf{R}^+ \mid Bx \geq b)$, c'est-à-dire si cette enveloppe convexe est entière, alors les deux relaxations linéaires et de Dantzig-Wolfe sont identiques.

Ce résultat se déduit de l'écriture de (P) où l'on peut remplacer (SP) par un polyèdre dont chacun des sommets est une de ses solutons.
On peut interpréter ce résultat en disant que, dans ce cas, la décomposition de Dantzig-Wolfe sur (SP) n'apporte pas rien sur l'intégrité de (SP).

Amélioration de la relaxation linéaire ?

A partir du résultat précédent, on peu noter que, dans tous les cas où le problème de pricing d'une génération de colonnes est polynomial, alors la relaxation linéaire de la génération de colonnes est la même qu'une version non décomposée.

Dans le cas de la relaxation du multiflot entier, ce sera bien la même valeurs !

Par contre, lorsque le pricer est NP-difficile, alors on constate le plus souvent une forte amélioration !

Dantzig-Wolfe versus décompo lagrangienne

On peut se demander quelle relaxation est la plus intéressante entre la relaxation de Dantzig-Wolfe et la relaxation lagrangienne.

Theorem

Les relaxations de Dantzig-Wolfe et la relaxation lagrangienne ont même valeurs.

Preuve : En effet, la relaxation lagrangienne de (P) est

$$(LRP) \begin{cases} \text{Min } c^T x - \gamma^T (Ax - a) \\ x = \sum_{q=1}^{|Q|} \chi^q t_q \\ \sum_{q=1}^{|Q|} t_q = 1 \\ t_q \in \{0,1\} \qquad \forall q \in \{1, \ldots, |Q|\} \end{cases}$$

avec γ le vecteur colonne optimal des multiplicateurs de Lagrange.

Dantzig-Wolfe versus décompo lagrangienne

On définit le dual lagrangien de la manière suivante

$$(LDP) \begin{cases} \max\limits_{\gamma \in R_+^{|a|}} \ \text{Min}\ c^T x - \gamma^T (Ax - a) \\ x \in Q \end{cases}$$

On peut linéariser le min en utilisant Q l'ensemble des solutions de (SP).

$$(LDP) \begin{cases} \text{Max}\ \eta \\ \eta \leq c^T \chi^q - \gamma^T (A\chi^q - a) \quad \forall q \in \{1, \ldots, |Q|\} \\ \gamma \in R_+^{|a|} \\ \eta \in R \end{cases}$$

En notant λ_q les variables duales associées aux contraintes du dual lagrangien, on peut écrire son dual de la façon suivante (en ajoutant sans perte de généralité la normalisation des valeurs λ.)

$$(DLDP) \begin{cases} \text{Min} \sum\limits_{q=1}^{|Q|} (c^T \chi^q) \lambda_q \\ \sum\limits_{q=1}^{|Q|} (A\chi^q) \lambda_q \geq a \\ \sum\limits_{q=1}^{|Q|} \lambda_q = 1 \\ \lambda_q \geq 0 \qquad \forall q \in \{1, \ldots, |Q|\} \end{cases}$$

On obtient la relaxation (MP) dans l'espace de variable correspondant à la décomposition de Dantzig-Wolfe. Donc le dual de la formulation par décomposition de Dantzig-Wolfe correspond au dual lagrangien. □

Problème de coloration

Soit $S \subseteq \mathbb{N}$. Une *coloration* des sommets d'un graphe $G = (V, E)$ est une fonction $r : V \to S$ telle que $r(u) \neq r(v)$ pour tout couple de sommets adjacents u, v. Les éléments de l'ensemble S sont appelés les *couleurs* disponibles. Une *k-coloration* est une coloration $c : V \to \{1, .., k\}$. Un graphe est dit *k-coloriable* s'il possède une k-coloration.

Tester si un graphe est k-coloriable est un problème NP-complet si $k \geq 3$, et il est polynomial si $k = 2$. En effet, si $k = 2$, il suffit de tester si le graphe est biparti, c'est-à-dire s'il ne contient pas des cycle impair. Ce qui peut se faire par un simple parcours de graphe.

Soit $G = (V, E)$ un graphe non-orienté. Le *problème de coloration* consiste à déterminer le plus petit k tel que G soit k-coloriable.

Première formulation

On associe à chaque sommet u de V un vecteur binaire à K dimensions $x_u = (x_u^1, ..., x_u^K)$, où K est une borne supérieure sur la coloration de G (au maximum $K = |V|$).
Comme l'on désire minimiser le nombre de couleur, on ajoute une variable binaire w^l par couleur $l = 1, ..., K$ indiquant si cette couleur a été utilisée ou non.

Le problème est donc équivalent au programme

$$(P_C) \begin{cases} \text{Min} \sum_{l=1}^{K} w^l & \\ \sum_{l=1}^{K} x_u^l = 1 & \forall u \in V, \\ x_u^l + x_v^l \leq w^l & \forall e = uv \in E \text{ et } 1 \leq l \leq K, \\ x_u^l \in \{0,1\} & \forall u \in V \text{ et } 1 \leq l \leq K, \\ w^l \in \{0,1\} & \forall 1 \leq l \leq K. \end{cases}$$

Les premières contraintes sont celles couplante ($Ax \leq a$) et les suivantes celles décomposables par "couleur l" ($Bx \geq b$).

Décomposition

Ainsi le sous-problème est

$$(SP_C)\begin{cases} \text{Min } \sum_{l=1}^{K}(d_l^0 w^l + d_l x^l) & \\ x_u^l + x_v^l \leq w^l & \forall e = uv \in E \text{ et } 1 \leq l \leq K \\ x_u^l \in \{0,1\} & \forall u \in V \text{ et } 1 \leq l \leq K \\ w^l \in \{0,1\} & \forall 1 \leq l \leq K \end{cases}$$

qui est en fait la somme des PLNE pour chaque couleur $l \in \{1, ..., K\}$

$$(SP_C^l)\begin{cases} \text{Min } d_l^0 w^l + d_l x^l & \\ x_u^l + x_v^l \leq w^l & \forall e = uv \in E \\ x_u^l \in \{0,1\} & \forall u \in V \\ w^l \in \{0,1\}. & \end{cases}$$

Le problème est donc décomposé en K sous-problèmes (SP_C^l) : un par couleur l.

Décomposition

On peut voir que le problème (SP_C) consiste donc à déterminer K stables !

Un vecteur solution appartant à l'ensemble Q des solutions est donc un vecteur représentant K stables et pour chaque stable I un entier w^I indiquant si ce stable I est ou non l'ensemble vide.

Posons alors $\mathscr{S}$ l'ensemble des stables G et $\chi^{\mathscr{S}}$ l'ensemble des vecteurs d'incidence de ces stables. Alors Q peut être défini comme un vecteur composé de toutes les combinaisons de K vecteurs $\chi^S \in \chi^{\mathscr{S}}$ chapeauté chacun de 0 si $S = \emptyset$ et 1 sinon.

Suivant le principe de Dantzig-Wofle, (SP_C^I) peut aussi s'écrire

$$(SP_C^I) \begin{cases} \text{Min } d_I^0 w^I + d_I x^I & \\ \quad x_u^I + x_v^I \leq w^I & \forall e = uv \in E \\ \quad x^I = \sum_{s \in \mathscr{S}} \chi^S t_s^I & \\ \quad \sum_{s \in \mathscr{S}} t_s^I = 1 & \\ \quad t_s^I \in \{0,1\} & \forall s \in \mathscr{S}. \end{cases}$$

Décomposition

On peut remarquer que si $d_l^0 > 0$ et pour une solution (entière) de (SP_C^l), on a en fait $w^l = \sum_{s \in \mathscr{S}} t_s^l$.
En effet, en considérant uniquement des vecteurs d'incidence de stables, la contrainte sur les arêtes n'a qu'au plus un des terme de gauche non nul. De plus, dans une solution optimale, w_l sera donc 0 ou 1. En fait, w_l vaudra 0 si la couleur l n'est attribué à aucun stable et 1 sinon : d'où le resultat.
On peut alors écrire

$$(SP_C^l) \begin{cases} \text{Min } d_l^0 \sum_{s \in \mathscr{S}} t_s^l + d_l x^l \\ x^l = \sum_{s \in \mathscr{S}} \chi^S t_s^l \\ \sum_{s \in \mathscr{S}} t_s^l = 1 \\ t_s^l \in \{0,1\} \qquad \forall s \in \mathscr{S}. \end{cases}$$

Décomposition

En utilisant cette écriture de (SP), on peut réécrire (P_C) de la manière suivante

$$(P_C)\begin{cases} \text{Min } \sum_{l=1}^{K} \sum_{s\in\mathscr{S}} t_s^l & \\ \sum_{l=1}^{K} x_u^l = 1 & \forall u \in V, \\ x^l = \sum_{s\in\mathscr{S}} \chi^s t_s^l & \\ \sum_{s\in\mathscr{S}} t_s^l = 1 & \\ t_s^l \in \{0,1\} & \forall s \in \mathscr{S} \text{ et } 1 \leq l \leq K \end{cases}$$

Décomposition

On peut noter ici que toutes les variables et vecteurs ne dépendent pas de I ! ! !
On peut donc réécrire de la façon suivante

$$(P_C)\begin{cases} \text{Min} \sum_{s\in\mathscr{S}} t_s & \\ \sum_{s\in\mathscr{S}} \chi_u^s t_s = 1 & \forall u \in V, \\ t_s \in \{0,1\} & \forall s \in \mathscr{S} \end{cases}$$

Cette écriture est une formulation, bien classique et bien compréhensible pour le problème de coloration !

$$(P_C)\begin{cases} \text{Min} \sum_{s\in\mathscr{S}} t_s & \\ \sum_{s\in\mathscr{S} \mid u\in s} \chi_u^s t_s = 1 & \forall u \in V, \\ t_s \in \{0,1\} & \forall s \in \mathscr{S} \end{cases}$$

Génération de colonnes pour la coloration

On peut alors considérer la relaxation linéaire (MP_C) de ce programme

$$(MP_C)\begin{cases} \text{Min} \sum_{s\in\mathscr{S}} t_s & \\ \sum_{s\in\mathscr{S}} \chi_u^s t_s = 1 & \forall u \in V, \\ t_s \geq 0 & \forall s \in \mathscr{S} \end{cases}$$

Considérons un sous-ensemble $\mathscr{S}^*$ de stables tels qu'il existe une coloration avec ces stables (on peut aisemenent obtenir une telle initialisation).

Notons (MP_C^*) la relaxation linéaire de (P_C) restreinte à $\mathscr{S}^*$.Notons t^* la solution optimale t^* de ce programme restreint (MP_C^*) et π_v les coûts duaux associés à ses inégalités.

Alors t^* est la solution optimale de (MP_C) si et seulement si

$$(1 - \sum_{v\in V} \pi_v \chi_v^s) \geq 0 \quad \forall s \in \mathscr{S}$$

c'est-à-dire s'il n'existe pas de stable s tel que $\sum_{v\in s} \pi_v < 1$.

Génération de colonnes pour la coloration

Le problème de pricing de génération de colonnes revient ainsi à rechercher un stable de poids maximum selon les coûts duaux λ_v, $v \in V$. Si ce stable maximal est plus grand stable est de poids < 1 alors on a trouvé un colonne à ajouter à (MP) sinon on a atteind la relaxation linéaire de (MP).

La borne obtenue est une des meilleures bornes inférieures connues pour le problème de coloration.

De plus, en utilisant cette génération de colonnes dans un algorithme de Branch-and-Price, on a pu obtenir des solutions optimales (entières) pour de nombreuses instances difficile.

En effet, le principe de branchement pour la formulation (MP) est assez simple à mettre en œuvre : on peut utiliser la règle de Ryan-And-Foster qui propose de créer deux fils à partir d'un couple de sommet (u, v) non adjacent du graphe : un fils où u et v ont même couleur et un fils où u et v sont de couleurs différentes. L'astuce de mise en œuvre repose sur le fait que pour un fils, on identifie les deux sommets dans le graphe et pour l'autre, on ajoute une arête : ainsi à chaque profondeur de l'arbre, le graphe étudié est simplifié.