Étude théorique et numérique du problème de la gestion de la diversité

Olivier Briant

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

THÈSE

présentée par

Olivier BRIANT

pour obtenir le grade de DOCTEUR

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 30 Mars 1992)

Spécialité : **Recherche Opérationnelle, Combinatoire et Optimisation**

Discipline : **Informatique**

Étude théorique et numérique du problème de la gestion de la diversité

Date de soutenance : 7 janvier 2000

Composition du Jury

Président :	Gerd	FINKE
Rapporteurs :	Gérard	CORNUÉJOLS
	Laurence	WOLSEY
Directeur de thèse :	Denis	NADDEF
Examinateurs :	Claude	LEMARÉCHAL
	François	VANDERBECK

Thèse soutenue au Laboratoire Informatique et Distribution
et préparée au Laboratoire de Modélisation et Calcul
(Institut d'Informatique et de Mathématiques Appliquées de Grenoble)

REMERCIEMENTS

Je remercie d'abord tous les membres du jury pour l'intérêt qu'ils ont porté à ma thèse. Je tiens à exprimer ma reconnaissance à Gérard Cornuéjols et Laurence Wolsey pour leurs conseils quant à l'amélioration de ce document, ainsi qu'à François Vanderbeck et Gerd Finke. Un grand merci aussi à Claude Lemaréchal pour sa disponibilité et son aide dans l'utilisation de la méthode des faisceaux. Et, bien sûr, je n'oublie pas celui qui a préparé mon retour à Grenoble, au moment où j'apprenais, sous la pluie de Picardie et le soleil des Antilles, que le terme *treillis* pouvait désigner autre chose qu'un objet mathématique, merci d'avoir été disponible même pour répondre à mes questions les plus stupides, je parle bien sûr de Denis Naddef.

Merci aussi à Martine Labbé et Yves Pochet avec qui j'ai eu la chance de discuter du problème de la diversité, discussions qui m'ont permis d'avancer à grands pas dans mes recherches.

Merci à tous les membres de l'ancienne équipe d'algorithmique parallèle, devenue le laboratoire ID. Ces remerciements s'adressent en premier lieu au plus italien des brésiliens, Alexandre, pour son amitié et ses conseils, et aussi Greg et Roberta, Alfredo, Gustavo, Mathias et Chris pour m'avoir accueilli respectivement dans leur bureau, et dans leur équipe de joueurs invétérés de 18h. Et pour les avoir ennuyés à tour de rôle pour des problèmes administratifs, informatiques ou de Latex : Andréa, Martha, Benhur, Cyril, François, Gerson, Jean-Guillaume, Nicolas, Renaud, et tous les permanents.

Merci encore à Frédérique qui a eu la gentillesse et la patience de relire ce document afin de chasser les "fotes de phransai".

Comment finir ces quelques lignes sans parler de celles et ceux que j'ai la chance de connaître depuis le DEA et qui m'ont fait découvrir les endroits les plus folkloriques et sympathiques de Grenoble, et du Sud-Est de la France. Je commencerai par celui pour qui désormais ça plane tout les jours, Ekbel, ancien

collègue de bureau, ancien coéquipier du Challenge ROADEF'99, et interprète officiel franco-tunisien, et, bien sûr, je n'oublie pas Pascal, Florence, Zouhir, Sophie, Julien, Christine et Imed.

Merci enfin aux trois personnes qui, depuis plus de 28 ans m'ont supporté dans tous les sens du terme, et à qui je dédie ce travail : mes parents, Jacques et Jeanine, et mon frère Laurent.

Table des matières

INTRODUCTION

Le problème de la gestion de la diversité est un problème d'origine industrielle. Il s'apparente à de nombreux problèmes d'optimisation combinatoire tels que les problèmes de localisation sans capacité, le problème dit du k-médian, ainsi que les problèmes de flot dans un réseau possédant des coût fixes.

Le problème de la gestion optimale de la diversité est défini sur un ensemble partiellement ordonné. L'objectif est de produire un sous-ensemble optimal de k éléments références. Chaque élément non produit doit être remplacé par une référence qui lui est supérieure d'après l'ordre partiel, ce qui provoque un surcoût. Les demandes et les coûts unitaires de production sont connus.

Bien que très souvent rencontré dans l'industrie, ce problème n'a été le sujet que de très peu de publications jusqu'à ce jour.

Notre travail consiste à l'étudier, d'abord en le modélisant afin de décrire le plus fidèlement possible ses solutions, puis à le résoudre jusqu'à l'optimalité.

Dans un premier chapitre, nous définissons exactement ce que nous entendons par "Problème de la Gestion de la Diversité". Nous donnons un exemple d'application industrielle, ainsi que la preuve de sa complexité. Nous concluons en présentant la modélisation que nous avons choisie, ainsi que les raisons qui nous ont poussés à délaisser d'autres types de modélisation, a priori plus simples.

La modélisation choisie est proche de celles des problèmes de localisation k-médians. Cette modélisation très fidèle à la réalité est un programme linéaire en nombres entiers que nous résolvons à l'aide d'un algorithme d'énumération implicite des solutions du problème.

Pour les instances de grande taille, il s'avère très difficile de résoudre ne serait-ce que la relaxation linéaire de ce programme. Nous initialisons donc cet algorithme d'énumération à l'aide d'une procédure basée sur les relaxations lagrangiennes, afin de réduire le nombre de variables de décision, et de fournir une solution heuristique de bonne qualité.

Au second chapitre, nous rappelons quelques notions sur les relaxations lagrangiennes, et nous détaillons les principales phases de cette procédure d'initialisation.

Le troisième chapitre est consacré aux différentes méthodes permettant de réduire le nombre de variables. Nous donnons une liste très complète des différents critères de fixations possibles.

Parallèlement aux fixations de variables, la procédure lagrangienne permet d'obtenir des solutions réalisables du problème de la gestion de la diversité. Nous détaillons toutes les heuristiques que nous avons mises au point au quatrième chapitre. Certaines d'entre elles, sont facilement adaptables à d'autres problèmes.

Nous consacrons ensuite un chapitre à l'algorithme de résolution exacte, basé sur une énumération implicite des solutions. Cet algorithme, de type *Branch and Cut*, nécessite une étude polyédrale de l'enveloppe convexe des solutions. Nous présentons notamment un type de contraintes permettant d'éliminer des solutions fractionnaires, ainsi qu'une heuristique de séparation.

Enfin, nous concluons ce document par des résultats numériques effectués sur des instances réelles.

1

PRÉSENTATION DU PROBLÈME ET MODÉLISATION

Sommaire

Ce premier chapitre est consacré à la présentation du problème de la gestion optimale de la diversité et à sa modélisation.

Dans une première section, nous définissons ce que nous entendons par "problème de la gestion optimale de la diversité", ainsi que la terminologie que nous utiliserons tout au long de ce document. Dans la seconde section, nous donnons un exemple d'application industrielle. Dans la section suivante, nous complétons cette terminologie, et nous présentons la représentation sous forme de graphe que nous utiliserons dans les chapitres suivants. Nous nous intéressons ensuite à la complexité de ce problème, ainsi qu'à celle de cas particuliers intéressants. Puis, nous concluons ce chapitre en présentant la modélisation que nous avons adoptée.

1.1 ÉNONCÉ ET NOTATIONS

Soit V un ensemble, dont les éléments seront appelés *combinaisons*, en référence à l'exemple d'application industrielle de la section suivante.

Chaque combinaison $i \in V$ doit être produite à d_i exemplaires.

Cet ensemble V est partiellement ordonné (la relation d'ordre partielle sera notée $\succ$), dans le sens où une combinaison i peut être remplacée par une combinaison $j \succ i$.

Le coût unitaire de production d'une combinaison i est de $c_i > 0$, et vérifie la proposition : $\quad i \prec j \quad \Rightarrow \quad c_i < c_j$

Nous supposerons que le sur-coût engendré par la substitution de i par $j \succ i$ est égal à $d_i(c_j - c_i)$.

Enfin, soit k un nombre de combinaisons différentes que l'on souhaite produire.

Question : comment choisir les k types de combinaisons produites, et quelles sont les substitutions que nous devons opérer si nous voulons minimiser le sur-coût total dû à ces substitutions ?

Noter que minimiser le sur-coût total revient à minimiser le coût total de la production. En effet, à l'optimum, la différence de ces deux valeurs est égale au coût engendré par la production de chaque combinaison séparément, c'est-à-dire celui de la diversité maximale où $k = |V|$. Pour simplifier l'écriture, nous minimiserons dans tout le document le coût total de la production, excepté au chapitre 6, où le calcul de garantie nécessitera de revenir au calcul du sur-coût total.

La section suivante est consacrée à l'exposé d'une application industrielle possible de ce problème.

1.2 EXEMPLE D'APPLICATION INDUSTRIELLE

La gestion optimale de la diversité est un problème d'origine industrielle. Cette section en montre un exemple.

Les constructeurs automobiles européens proposent à leurs clients un très grand nombre d'options (coussins gonflables conducteur et/ou passager, latéraux, trois ou cinq portes, conduite à gauche, à droite, etc...), et autorisent ceux-ci à les combiner comme bon leur semble, dans la limite du réalisable (on ne peut pas commander une voiture ayant simultanément la conduite à gauche et à droite, ou ayant le coussin gonflable passager sans celui du conducteur, par exemple).

Malheureusement, cette liberté laissée aux clients a souvent pour conséquence la demande d'un très grand nombre de combinaisons différentes, et aussi par conséquent la production d'un très grand nombre de types de faisceau électrique. Ce nombre pouvant atteindre plusieurs milliers, il est hors de question de gérer autant de faisceaux différents sur la chaîne de montage. Les constructeurs demandent donc systématiquement à leur équipementier de leur en fournir uniquement un échantillon très réduit : certains ne souhaitent avoir que 5 types de faisceaux, d'autres que 30 au maximum, quitte à poser sur certaines automobiles un faisceau plus complet que celui qui est nécessaire. Cette opération consistant à substituer un premier faisceau par un second ne peut se faire que si le faisceau posé permet le fonctionnement de toutes les options pour lesquelles le premier faisceau étaient conçu. Ceci implique que certains fils du faisceau posé ne seront pas utilisés, et par conséquent un sur-coût de production.

Le nombre de faisceaux ayant été déterminé en accord avec le constructeur, le problème de l'équipementier est de savoir quels sont ceux qu'il produira, et quelles substitutions il proposera, si son but est de minimiser le sur-coût total.

Ce problème, bien que fréquemment rencontré dans l'industrie, n'a été jusqu'à maintenant, et à notre connaissance, que très peu étudié.

1.3 GRAPHE ASSOCIÉ

Dans tout ce document, nous utiliserons fréquemment le graphe orienté $G = (V, A)$ associé à la relation d'ordre partiel $\succ$.

Ce graphe est défini par :

– V ensemble des combinaisons

– $A = \{ij \in V \times V \quad | \quad j \succ i\}$

La figure 1.1 illustre un exemple. Le tableau de gauche donne la liste des combinaisons sous forme de vecteurs où un 1 signifie que l'option associée est demandée.

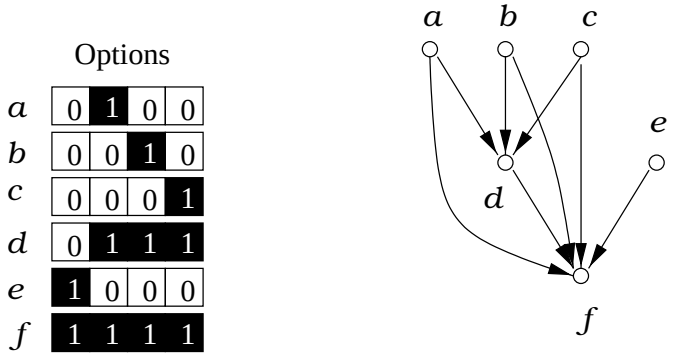


FIG. 1.1 – *Exemple de représentation sous forme de graphe*

Nous utiliserons indifféremment les termes de sommets ou combinaisons, ainsi que d'arcs ou de substitutions.

Les sommets, ou combinaisons, maximaux par rapport à l'ordre partiel seront qualifiés de *terminaux*, et les éléments minimaux de *sources*. Dans l'exemple de la figure 1.1, les sommets a, b, c et e sont des sources, et f est le seul terminal.

L'ensemble des terminaux sera noté T.

La terminologie étant définie, nous prouvons dans la section suivante que ce problème de la gestion de la diversité, qui a priori semble très simple, fait parti de la classe des problèmes très difficiles à résoudre.

1.4 COMPLEXITÉ

Cette section est consacrée à l'étude de la complexité du Problème de la Gestion de la Diversité, noté *PGD*. Pour plus de détails concernant la théorie de la complexité, nous conseillons au lecteur de se reporter au livre de Garey et Johnson [GJ79]. Nous verrons que, de cette étude, peuvent être déduits d'autres résultats concernant certains cas particuliers.

Mais avant d'énoncer le théorème principal, nous définissons ce que nous entendrons par *absorbant d'un ensemble partiellement ordonné,* et rappelons la définition du Problème de la Gestion Optimal dans le cas général, ainsi que celle du

Problème du Transversal que l'on sait être NP-complet (voir [GJ79]) et que nous utiliserons dans la suite :

Définition 1.4.1

Soit V un ensemble et $\succ$ une relation d'ordre partiel sur V
On dit qu'un ensemble $R \subseteq V$ est un absorbant de V si pour tout $i \in V \setminus R$ il existe $j \in R$ tel que $j \succ i$.

Si on considère le graphe $G = (V, A)$ associé à $(V, \succ)$, que nous avons décrit à la section précédente, on retrouve la notion traditionnelle d'absorbants de la théorie des graphes.

Problème de la Gestion de la Diversité

`INSTANCE :` Un ensemble V, une relation d'ordre partiel $\succ$ sur V, une fonction coût $c : V \to I\!N$ telle que $\forall i, j \in V, j \succ i \Rightarrow c(j) > c(i)$, une fonction demande $d : V \to I\!N$, un entier positif k et une borne B.

`QUESTION :` Existe-t-il un absorbant $R \subseteq V$ de V de cardinalité k et de poids $z(R)$ inférieur ou égal à B ?
avec $z(R) = \sum_{j \in R} c(j)d(j) + \sum_{i \in V \setminus R} c_R(i)d(i)$
et pour tout $i \in V \setminus R$, $c_R(i) = \min\{c(j) \ / \ j \in R \text{ et } j \succ i\}$

Problème du Transversal

`INSTANCE :` Un graphe (W, E) et un entier positif $q \leq |W|$

`QUESTION :` Existe-t-il un ensemble $W' \subseteq W$, nommé *transversal*, de cardinalité q tel que toute arête de E a au moins une extrémité dans W' ?

Théorème 1.4.2

PGD est NP-complet

Preuve de 1.4.2 : Il est évident que PGD $\in NP$, car si nous connaissons un ensemble R, nous pouvons vérifier en temps polynômial que sa cardinalité est de k, qu'il existe au moins un arc reliant tout élément $V \setminus R$ à un élément de R, et que $z(R) \leq B$.

Nous allons transformer le Problème du Transversal en PGD.

Soit une instance arbitraire du Problème du Transversal donnée par un graphe (W, E) et un entier $q \leq |W|$. Sans perte de généralité nous pouvons supposer que G ne possède pas de sommet isolé. Nous allons construire une instance $(V, \succ$

$, c, d, k, B)$ du PGD telle qu'il existe un absorbant R de V de cardinalité k ayant un poids inférieur à B si et seulement si le graphe (W, E) admet un transversal W' de cardinalité q.

Nous définissons cette instance $I_{PGD}(W, E, q) = (V, \succ, c, d, k, B)$ par

1. $V = W \cup E \cup \{p\}$, où p est un sommet *puits*
2. pour tout $i, j \in V$, $j \succ i$ si et seulement si
   - ou bien $i = uv \in E$ et $j \in \{u, v\}$
   - ou bien $i \in E \cup W$ et $j = p$
3. pour tout $uv \in E,\ c(uv) = 1$, pour tout $u \in W,\ c(u) = 2$, et $c(p) = 3$
4. pour tout $uv \in E,\ d(uv) = 1$, pour tout $u \in W,\ d(u) = 1$, et $d(p) = 0$
5. $k = q + 1$

La valeur de B sera définie ultérieurement. Nous la supposons donnée en attendant.

La figure 1.2 illustre cette transformation sur un exemple, le graphe de droite représentant $(V, \succ)$.

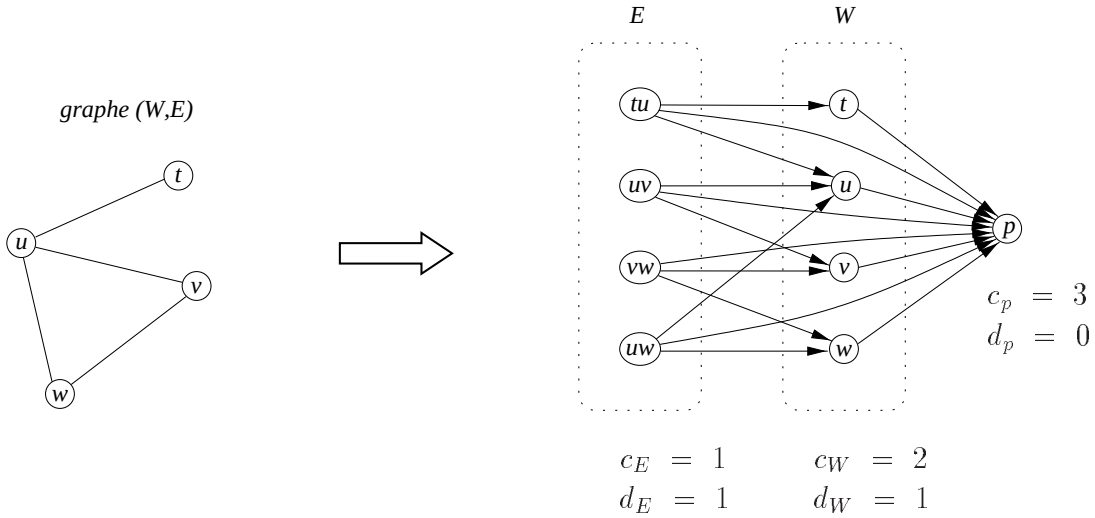


FIG. 1.2 – *Transformation du Problème du Transversal en PGD*

Le lemme suivant va nous permettre de caractériser certains absorbants de cardinalité $k = q + 1$ ayant un poids minimum.

Lemme 1.4.3

$I_{PGD}(W, E, q)$ possède au moins un absorbant R de cardinalité $q + 1$ de poids minimum tel que $R \cap E = \emptyset$, c'est-à-dire s'écrivant sous la forme $R = W' \cup \{p\}$, avec $W' \subseteq W$ et $|W'| = q$.

Preuve de 1.4.3 : Il est clair que $I_{PGD}(W, E, q)$ admet au moins un absorbant de cardinalité $q + 1$, il suffit de prendre n'importe quel sous-ensemble de $E \cup W$ de cardinalité q et de le compléter par p.

Le nombre d'absorbants de cardinalité $q + 1$ est fini, donc la notion de *poids minimum* de ces ensembles est bien définie.

De plus, par définition, p est élément de tout absorbant de V. Donc, pour démontrer ce lemme il suffit de construire à partir d'un absorbant de poids minimum arbitrairement choisi un autre absorbant de même poids ayant une intersection vide avec E.

Soit S un absorbant de V de cardinalité $q+1$ de poids minimum. Si S contient un élément $uv \in E$ alors deux cas peuvent se présenter : ou bien ni u ni v n'appartient à S, ou bien u ou v appartient à S.

Soit $S' = S \cup \{w\} \setminus \{uv\}$ avec $w = u$ dans le premier cas, et $w \in W \setminus S$ dans le second (un tel w existe car au plus $q - 1$ éléments de W appartiennent à S). Il est clair que S' est un absorbant de V et que sa cardinalité est $q + 1$.

De plus $z(S') \leq z(S) - (c(uv)d(uv) + c_S(w)d(w)) + (c_{S'}(uv)d(uv) + c(w)d(w))$

Or $c_S(w) = c(p) = 3$ car $w \notin S$, et $c_{S'}(uv) = c(u)$ (ou $c(v)$) = 2 car $uv \notin S'$ et u ou $v \in S'$.

D'où $z(S') \leq z(S) - (1 \times 1 + 3 \times 1) + (2 \times 1 + 2 \times 1) = z(S)$

Or S est de poids minimum, donc S' l'est aussi.

Ainsi nous avons construit un absorbant S' de cardinalité $q + 1$ de poids minimum et possédant exactement un élément de E de moins que S.

Par réitération de ce processus, nous obtenons l'absorbant R cherché. ♢

Soit $R = W' \cup \{p\}$ un absorbant de V ayant les propriétés décrites au lemme 1.4.3. Si E_R désigne l'ensemble des $uv \in E$ tels que ni u ni v appartient à R, i.e. $c_R(uv) = c(p)$, alors on a

$$\begin{aligned} z(R) &= \sum_{v \in W'} c(v)d(v) + c(p)d(p) + \sum_{v \in W \setminus W'} c_R(v)d(v) + \sum_{uv \in E} c_R(uv)d(uv) \\ &= \sum_{v \in W'} 2 \times 1 + 3 \times 0 + \sum_{v \in W \setminus W'} 3 \times 1 + \sum_{uv \in E \setminus E_R} 2 \times 1 + \sum_{uv \in E_R} 3 \times 1 \\ &= (2|E| + 3|W| - q) + |E_R| \end{aligned}$$

Ce qui implique que tout absorbant de V de cardinalité $q + 1$ a un poids supérieur à $(2|E| + 3|W| - q) + |E_R|$. Donc, en posant $B = 2|E| + 3|W| - q$, on

en déduit que $I_{PGD}(W, E, q)$ admet un absorbant de cardinalité k et de poids inférieur à B si et seulement s'il admet un absorbant $R = W' \cup \{p\}$, avec $W' \subseteq W$ et $|W'| = q$, tel que $E_R = \emptyset$, c'est-à-dire si et seulement s'il existe un ensemble W' de q éléments de W tel que pour tout $uv \in E$, u ou v appartient à W', ce qui conclut la preuve du théorème 1.4.2. ♢

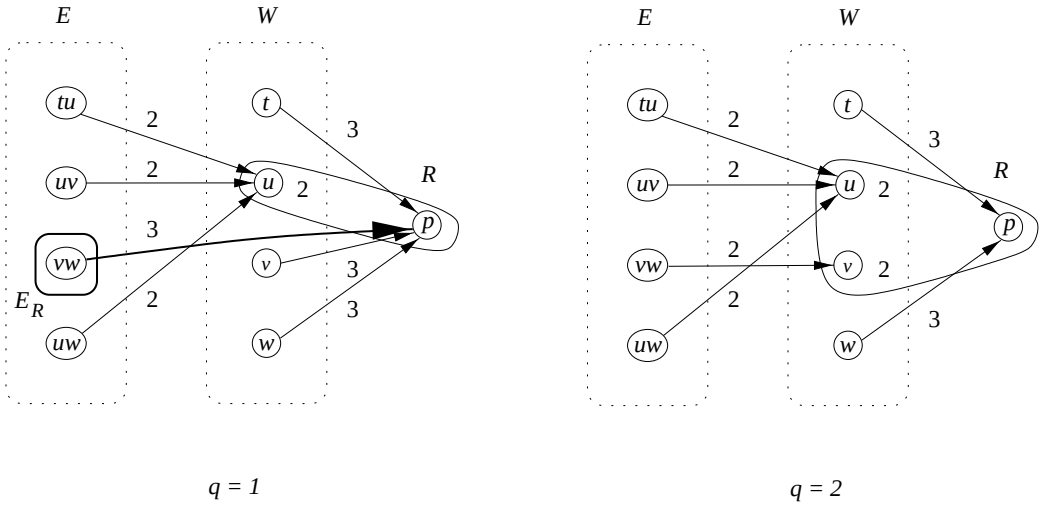


FIG. 1.3 – *Définition de E_R et calcul de $z(R)$*

Si on reprend l'exemple précédent, on voit qu'il n'existe aucun transersal de (W, E) de cardinalité $q = 1$ car $R = \{u, p\}$ est le seul absorbant du graphe associé à $(V, \succ)$ de cardinalité 2 et de poids minimum, et ce poids est égal à $20 > 2|E| + 3|W| - q = 20 - 1 = 19$, en revanche $\{u, v, p\}$ a un poids de 18, donc $\{u, v\}$ est un transversal de (W, E) possédant $q = 2$ sommets. (c.f. figure 1.3, les indications sur les arcs sont les valeurs des $c_R(i)d(i)$ pour les $i \notin R$, et celles à côté des sommets i de R sont les valeurs de $c(i)d(i)$).

Dans le problème de la gestion de la diversité telle que nous l'avons défini à la section 1.1, les coûts sont strictement croissants, dans le sens que si nous avons $i \prec j$ alors $c_i < c_j$. Cette définition est une représentation réaliste du problème industriel, mais nous aurions pu aussi admettre une croissance non stricte des coûts, c'est-à-dire demander simplement que $c_i \leq c_j$ dès que $i \prec j$. Le problème resterait NP-difficile, vu que le théorème 1.4.2 a été prouvé pour un cas particulier de cette extension.

D'un point de vue théorique, il est intéressant d'identifier une classe d'instances qui provoque une complexité aussi élevée. Ainsi, pour prouver qu'un cas

particulier du PGD est NP-difficile, il suffira que l'ensemble des instances sur lesquelles il est défini contienne cette classe. Par exemple, nous pourrons en déduire que même si nous exigeons que toutes les demandes sont égales à une certaine constante, ce problème reste NP-difficile.

Comme pour toutes les instances du PGD, nous définissons celles appartenant à cette classe, que nous noterons $\mathcal{C}$, en choisissant un ensemble de combinaisons V partiellement ordonné par une relation $\succ$, un nombre k de combinaisons à produire et les fonctions coût c et demande d. La classe $\mathcal{C}$ est l'ensemble de toutes les instances du PGD qui peuvent être construites à partir de graphes (W, E) quelconques de la façon suivante :

1. $V = W \cup E \cup \{p\}$, où p est un sommet *puits*
2. pour tout $i, j \in V$, $j \succ i$ si et seulement si
   - ou bien $i = uv \in E$ et $j \in \{u, v\}$
   - ou bien $i \in E \cup W$ et $j = p$
3. $\forall i \in V,\ c(i) = \begin{cases} c_E & \text{si } i \in E \\ c_W & \text{si } i \in W \\ c_p & \text{si } i = p \end{cases}$ et $d(i) = \begin{cases} d_E & \text{si } i \in E \\ d_W & \text{si } i \in W \\ d_p & \text{si } i = p \end{cases}$
4. $1 < k \leq |W| + 1$

où c_E, c_W, c_p, d_E, d_W et d_p sont des constantes de $\mathbb{N}$ vérifiant

a) $d_E > 0$

b) $c_p > c_W$

c) $(c_W - c_E)d_E \leq (c_p - c_W)d_W$

Corollaire 1.4.4

Soit PGD' un cas particulier du PGD.
Si toutes les instances de la classe $\mathcal{C}$ sont des instances de PGD', alors PGD' est NP-complet.

Preuve de 1.4.4 : Il suffit de reprendre la démonstration de 1.4.2

Le lemme 1.4.3 reste vrai grâce à 3), et pour tout absorbant R vérifiant les conditions énoncées dans ce lemme, on a $z(R) = \alpha + \beta|E_R|$ avec $\alpha = c_W d_E |E| + c_p d_W |W| + q(c_W - c_p)d_W + c_p d_p$ et $\beta = (c_p - c_W)d_E$.

Vu que $\beta > 0$ d'après 1) et 2), il suffit de poser $B = \alpha$ pour conclure la démonstration. $\diamondsuit$

En pratique, il peut arriver que des combinaisons aient des demandes nulles. Il s'agit alors de combinaisons qui, techniquement, peuvent être produites pour en

remplacer d'autres, mais qui ne correspondent à aucun faisceau demandé par des clients.

Pour modéliser cette variante, il est clair que la notion d'absorbant est trop forte. En effet, par définition, tout absorbant contient l'ensemble des combinaisons terminales, celles-ci ne pouvant être remplacées par aucune autre combinaison. Par conséquent, cette notion d'absorbant nous oblige à produire toutes les combinaisons terminales, même celles qui ne sont pas demandées, ce qui n'est pas, a priori, une stratégie optimale. Nous allons donc définir une notion d'absorption plus générale.

Définition 1.4.5

Soit V un ensemble, $\succ$ une relation d'ordre partiel sur V, et une fonction demande $d : V \rightarrow \mathbb{N}$.
On dit qu'un ensemble $R \subseteq V$ est un pseudo-absorbant de (V, d) si pour tout $i \in V \setminus R$ tel que $d(i) > 0$ il existe $j \in R$ tel que $j \succ i$.

En d'autres termes, un ensemble R est pseudo-absorbant s'il absorbe tous les éléments de $W \setminus R$, comme le ferait un absorbant classique, sauf que l'on s'autorise à ne pas absorber les éléments qui ont une demande nulle.

Ce nouveau problème est évidemment NP-difficile, car le PGD n'en est qu'un cas particulier.

Dans tout ce document, nous travaillerons toujours avec la notion traditionnelle d'absorbants. Cependant, si nous voulons tenir compte qu'une combinaison non demandée peut ne pas être produite, nous pouvons ajouter une combinaison *super terminale*, de demande nulle et de coût infinie (ou exagérément grande), reliée à toutes les combinaisons appartenant à V. Ainsi, dans toutes les solutions réalisables que nous obtiendrons, une combinaison possédant une demande nulle, même si elle était terminale dans le graphe initial, pourra être remplacée par ce super terminal, sans altérer le coût de la solution (le coût de cette substitution étant nul).

Exemple de variante polynômiale : le cas particulier du PGD, consistant à ne considérer que les instances du PGD où $\succ$ est un ordre total, n'admet pas les instances $I_{PGD}(W, E, q)$ pour les graphes (W, E) possédant au moins deux sommets. En effet, les éléments de W sont incomparables entre eux dans ces instances, ce qui est contraire au fait que l'ordre $\succ$ doit être total. Ce problème est d'ailleurs polynômial (se reporter par exemple à l'article [JLM+95]).

1.5 MODÉLISATION

La modélisation d'un problème est certainement la phase la plus importante de son étude, car elle dicte les méthodes de résolution que nous devons utiliser, ainsi que les variables et les contraintes du problème.

Nous allons étudier dans cette section trois modélisations naturelles du problème de la gestion de la diversité : modélisation en problème de flot dans un réseau possédant des coûts fixes (section 1.5.1), en problème de localisation avec coût fixe (section 1.5.2), et en problème de localisation k-médian (section 1.5.3).

Nous concluons cette section en donnant les principales raisons qui nous ont amenées à choisir la troisième modélisation.

1.5.1 PROBLÈME DE FLOT DANS UN RÉSEAU À COÛT FIXE

Les problèmes de flot appartiennent aux problèmes d'optimisation combinatoire les plus classiques. Dans un graphe orienté donné $G' = (V', A')$, un flot consiste à "transporter" une marchandise de certains sommets i dits *sources*, ayant une demande $d_i > 0$, vers d'autres sommets j dits *puits*, ayant une demande $d_j < 0$. Nous supposons que la somme des demandes est nulle, c'est-à-dire que la quantité de marchandise offerte est égale à la quantité demandée. Ce transport s'effectue à travers les arcs du graphe G'. Pour un problème de flot dans un réseau à coût fixe, les arcs ij possèdent deux types de coûts : d'une part un coût a_{ij} proportionnel à la quantité traversant l'arc ij, et d'autre part un coût fixe b_{ij} qui est forfaitaire, c'est-à-dire payé, en une seule fois, dès que la moindre marchandise emprunte l'arc ij.

Pour modéliser ce problème à l'aide d'un programme mixte, nous définissons deux types de variables associées à chaque arc ij du graphe :

$f_{ij} =$ quantité de marchandise empruntant l'arc ij

$$g_{ij} = \begin{cases} 1 & \text{si l'arc est emprunté} \\ 0 & \text{sinon} \end{cases}$$

Le programme linéaire mixte que nous considérons est donc

$$\text{minimiser} \quad \sum_{ij \in A'} a_{ij} f_{ij} \quad + \quad \sum_{ij \in A'} b_{ij} g_{ij} \tag{1.1}$$

$$\text{sachant} \quad \sum_{j\ell \in A'} f_{j\ell} - \sum_{ij \in A'} f_{ij} = d_j \qquad \forall j \in V' \tag{1.2}$$

$$f_{ij} \leq M g_{ij} \qquad \forall ij \in A' \tag{1.3}$$

$$f_{ij} \geq 0 \qquad \forall ij \in A' \tag{1.4}$$

$$g_{ij} \in \{0, 1\} \qquad \forall ij \in A' \tag{1.5}$$

Les contraintes (1.2) traduisent la conservation du flot et la satisfaction de la demande de chaque sommet. Les contraintes (1.3) signalent si une quantité de marchandise non nulle traverse chaque arc (la constante M est choisie arbitrairement grande). Enfin les contraintes (1.4) et (1.5) signifient que seules les variables g_{ij} sont astreintes à être entières.

Afin de modéliser le problème de la gestion optimale de la diversité en un problème de flot dans un réseau à coût fixe, nous définissons le graphe $G' = (V', A')$ par :

- $V' = V \cup \{p\}$, ensemble de toutes les combinaisons, complété par un sommet puits p
- $A' = A \cup \{jp \mid j \in V\}$

Les demandes sont de d_j pour toutes les combinaisons j de V, et de $-\sum_{j \in V} d_j$ pour p.

Les coûts a_{ij} et b_{ij} des arcs $ij \in A'$ sont

$$a_{ij} = \begin{cases} 0 & \text{si } ij \in A \\ c_i & \text{si } j = p \end{cases}$$

$$b_{ij} = \begin{cases} 0 & \text{si } ij \in A \\ K & \text{si } j = p \end{cases}$$

avec K une constante donnée.

Remarque 1 : emprunter un arc jp revient à produire la combinaison j. Le flot passant sur cet arc peut être vu comme étant la quantité de combinaisons de type j que nous allons produire. Cette quantité provient non seulement de la demande de j mais aussi des demandes de toutes les combinaisons $i \prec j$ qui seront substituées par j. Ces combinaisons i sont celles dont le flot associé à leur demande passe par j.

Remarque 2 : nous pourrions, au lieu de considérer toutes les substitutions possibles ij de A, simplement choisir les substitutions "directes", c'est-à-dire celles pour lesquelles il n'existe pas de combinaison ℓ telle que $i \prec \ell \prec j$. Ceci permettrait de diminuer très fortement le nombre de variables du problème sans altérer la modélisation.

Remarque 3 : la constante K est fixée, mais il est évident que plus elle est grande, moins les arcs ip empruntés seront nombreux, et donc plus petit sera le nombre de combinaisons produites. Choisir la valeur de K qui nous permettra de produire le nombre k de combinaisons désiré est un problème en soit. Nous en discuterons à la section 1.5.4.

Nous pouvons réécrire le programme linéaire mixte de la façon suivante :

$$
\begin{aligned}
\text{minimiser} \quad & \sum_{i \in V} c_i f_{ip} \quad + \quad K(\sum_{i \in V} g_{ip}) & \\
\text{sachant} \quad & f_{jp} + \sum_{j\ell \in A} f_{j\ell} - \sum_{ij \in A} f_{ij} = d_j & \forall j \in V \\
& \sum_{j \in V} f_{jp} = \sum_{j \in V} d_j & \\
& f_{ip} \leq M g_{ip} & \forall i \in V \\
& f_{ij} \geq 0 & \forall ij \in A \cup (V \times \{p\}) \\
& g_{ip} \in \{0,1\} & \forall i \in V
\end{aligned}
$$

Le problème du flot de coût minimum dans un réseau à coût fixe a donné lieu à un grand nombre de publications. Citons notamment M.W. Padberg, T.J. Van Roy et L.A. Wolsey [PVW85], R.L Rardin et L.A. Wolsey [VW85], L.A. Wolsey [Wol89], T.J. Van Roy et L.A. Wolsey [VW85] qui étudièrent le polyèdre des solutions réalisables de ce problème afin de caractériser de nouvelles contraintes valides permettant d'éliminer certaines solutions fractionnaires de la relaxation linéaire. Le lecteur intéressé par ce type de problème peut aussi se reporter à l'ouvrage de G.L. Nemhauser et L.A. Wolsey [NW88a], et au chapitre de A. Balakrishnan et al. [BMM97] pour une bibliographie annotée sur le sujet.

La section suivante reprend l'idée que nous venons de développer, à savoir utiliser un coût fixe K pénalisant la production, plutôt que de fixer le nombre k de combinaisons produites. Pour cela nous allons modéliser notre problème grâce aux problèmes de localisation.

1.5.2 PROBLÈME DE LOCALISATION AVEC COÛTS FIXES

Il existe de très nombreux problèmes dans la littérature désignés sous le terme générique de “problème de localisation” (voir par exemple le chapitre de M. Labbé et F. V. Louveaux [LL97] donnant une bibliographie détaillée de ces différents problèmes). Pour chacun d’eux, il s’agit de desservir les demandes d_i d’un ensemble de clients I à partir d’un ensemble d’usines. L’exploitation de ces usines impliquant un coût fixe associé à leur localisation, le but est de désigner les usines à partir desquelles les clients pourront s’approvisionner à coût minimum. Nous noterons par J l’ensemble des localités d’implantation possibles.

Pour le problème auquel nous nous intéressons dans cette section, nous connaissons le coût de transport a_{ij} proportionnel à la quantité de marchandise transitant d’une usine située en $j \in J$ vers le client $i \in I$, ainsi que le coût fixe b_j associé à l’exploitation d’une usine en j. Ce problème est souvent référencé sous le nom de “problème de localisation simple” (de l’anglais “simple plant location problem”).

Nous définissons deux types de variables :

Pour tout $i \in I$ et $j \in J$,

$$x_{ij} = \begin{cases} 1 & \text{si le client } i \text{ est approvisionné par une usine située en } j \\ 0 & \text{sinon} \end{cases}$$

Pour tout $j \in J$,

$$y_j = \begin{cases} 1 & \text{si une usine est exploitée en } j \\ 0 & \text{sinon} \end{cases}$$

Ce problème peut être modélisé à l’aide du programme linéaire en nombres entiers suivant, que nous nommerons $(\mathcal{P}^{loc})$:

$$\text{minimiser} \quad \sum_{i \in I} \sum_{j \in J} d_i a_{ij} x_{ij} \quad + \quad \sum_{j \in J} b_j y_j \tag{1.6}$$

$$\text{sachant} \quad \sum_{j \in J} x_{ij} = 1 \qquad \forall i \in I \tag{1.7}$$

$$x_{ij} \leq y_j \qquad \forall i \in I, j \in J \tag{1.8}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I, j \in J \in J \tag{1.9}$$

$$y_j \in \{0,1\} \qquad \forall j \in J \tag{1.10}$$

Les contraintes (1.7) signifient qu’une des usines doit satisfaire la demande de chaque client. Les contraintes (1.8) rendent obligatoire l’exploitation d’une usine

en j si l'un des clients i est approvisionné par ce site. Enfin les contraintes (1.9) et (1.10) traduisent l'intégralité des variables.

Le problème de la gestion de la diversité peut être vu comme un cas particulier de ce problème, si, au lieu d'imposer le nombre k de combinaisons produites, nous pénalisons la production de chacune d'elle grâce à un coût fixe K.

Dans cette représentation, les ensembles I et J sont égaux à l'ensemble V des combinaisons. Par conséquent, il est clair que dans toute solution optimale, si une variable y_j vaut 1, alors la variable x_{jj} vaudra aussi 1. Nous remplaçons donc les variables y_j par les variables x_{jj}. De plus, nous ne définissons des variables x_{ij}, avec $i \neq j$, que si la demande de i peut être supportée par la combinaison j, c'est-à-dire si ij représente une substitution légale appartenant à A. La signification des variables est donc

Pour tout $ij \in A$,

$$x_{ij} = \begin{cases} 1 & \text{si la combinaison } i \text{ est substituée par la combinaison } j \\ 0 & \text{sinon} \end{cases}$$

Pour tout $j \in V$,

$$y_j = \begin{cases} 1 & \text{si la combinaison } j \text{ est produite} \\ 0 & \text{sinon} \end{cases}$$

Pour une combinaison j donnée, tous les coûts a_{ij} sont égaux au coût de production c_j de j. Nous obtenons ainsi le programme linéaire en nombres entiers suivant, que nous noterons $(\mathcal{P}_K)$:

$$\begin{array}{lll} \text{minimiser} & z_K(x) = \displaystyle\sum_{i \in V} \sum_{j \succeq i} d_i c_j x_{ij} \quad + \quad K\Big(\sum_{j \in V} x_{jj}\Big) & \\ \text{sachant} & \displaystyle\sum_{j \succeq i} x_{ij} = 1 & \forall i \in V \\ & x_{ij} \leq x_{jj} & \forall ij \in A \\ & x_{ij} \in \{0,1\} & \forall ij \in A \\ & x_{jj} \in \{0,1\} & \forall j \in V \end{array}$$

Remarque 4 : que ce soit pour le problème de localisation général, comme pour le problème de la gestion de la diversité, nous pourrions relâcher les contraintes d'intégralité pour les variables x_{ij} avec $i \neq j$. En effet, le principal problème consiste à choisir les combinaisons que nous devons produire (ou, pour le cas général, les sites d'implantation). Après avoir fait ce choix, affecter à chaque i l'élément j le moins cher est trivial.

Remarque 5 : comme à la section précédente, la constante K est une donnée. Il est évident que plus elle grande, moins de combinaisons seront produites, et réciproquement. Le choix de la valeur K permettant d'obtenir une production de k combinaisons sera abordé à la section 1.5.4.

J. Krarup et P.M. Pruzan [KP83] ont dressé un survol très complet sur les problèmes de localisation simples. Ils exposent notamment les relations qui existent entre ces problèmes et d'autres problèmes connus de l'optimisation combinatoire, tels que les problèmes de partitionnement, de stabilité ou de couverture. Ils abordent aussi les principaux algorithmes dont l'efficacité est reconnue pour résoudre ce type de problèmes. D. Erlenkotter [Erl78] propose un algorithme d'énumération implicite des solutions, basé sur la résolution du programme dual à l'aide d'une heuristique d'ajustement de variables. P.C. Jones et al. [JLM+95] se sont intéressés à des problèmes de localisation ayant une structure très particulière permettant de les résoudre en temps polynômial.

Pour plus de détails, le lecteur est invité à consulter le chapitre de M. Labbé et F. V. Louveaux [LL97] pour un état de l'art sur tous les problèmes de localisation, et celui de G. Cornuéjols et al. [CNW90] pour les problèmes de localisation simples.

En formulant le problème de la gestion de la diversité comme cas particulier d'un problème de flot dans un réseau possédant des coûts fixes, ou d'un problème de localisation simple, nous avons volontairement omis l'une des contraintes les plus astreignantes : la production d'exactement k combinaisons. La modélisation que nous décrivons à la section suivante va, pour la première fois, en tenir compte.

1.5.3 PROBLÈME DE LOCALISATION k-MÉDIAN

Les problèmes de localisation k-médians appartiennent à la famille des problèmes de localisation cités à la section précédente. Leur particularité se réduit à limiter le nombre d'usines exploitées, c'est-à-dire à ajouter au programme $(\mathcal{P}^{loc})$ la contrainte

$$\sum_{j \in J} y_j = k$$

Pour modéliser plus justement le problème de la gestion de la diversité, nous allons ajouter au programme $(\mathcal{P}_K)$ une telle contrainte. Ceci nous permet de ne plus pénaliser la production, c'est-à-dire d'annuler le coût fixe K.

Ainsi, le programme linéaire en nombres entiers que nous obtenons, et que nous noterons $(\mathcal{P}_k)$ s'énonce

$$\text{minimiser} \quad z_k(x) = \sum_{i \in V} \sum_{j \succeq i} d_i c_j x_{ij} \tag{1.11}$$

$$\text{sachant} \quad \sum_{j \in V} x_{jj} = k \tag{1.12}$$

$$\sum_{j \succeq i} x_{ij} = 1 \qquad \forall i \in V \tag{1.13}$$

$$x_{ij} \leq x_{jj} \qquad \forall ij \in A \tag{1.14}$$

$$x_{ij} \in \{0,1\} \qquad \forall ij \in A \tag{1.15}$$

$$x_{jj} \in \{0,1\} \qquad \forall j \in V \tag{1.16}$$

La signification des variables et des contraintes est identique à celle de la section précédente.

La remarque 5, quant à la possibilité de relâcher les contraintes d'intégralité (1.15), demeure valable.

Pour résoudre les problèmes de localisation k-médians, N. Christofides et J. Beasley [CB82] proposent un algorithme d'énumération basée sur les relaxations lagrangiennes (voir aussi [Bea93a]). P. Avella et A. Sassano ont caractérisé une classe de facettes du polytope associé aux problèmes k-médians définis sur une clique orientée. Pour cela, ils ont étudié les relations qui existent entre ce polytope et celui associé au problème consistant à rechercher un stable de cardinalité donnée et de poids minimum. M.E. Captivo [Cap91] propose une heuristique primal-dual basée sur l'heuristique que D. Erlenkotter [Erl78] avait décrite pour le problème de localisation simple. J. M. Mulvey et H.P. Crowder[MC79] se sont intéressés à un problème très proche de ceux de localisation k-médians : le problème de "grappes" (de l'anglais "clustering problem"). L'heuristique qu'ils ont développée est basée sur les relaxations lagrangiennes. R. Hassin et A. Tamir [HT91] décrivent un algorithme basé sur la programmation dynamique permettant de résoudre très efficacement le problème de localisation k-médian sur une ligne. Pour plus de détails sur les problèmes de localisation k-médians, nous invitons le lecteur à consulter notamment le chapitre de P.B. Mirchandani [Mir90].

Nous venons de voir trois modélisations possibles du problème de la gestion de la diversité. Les deux premières possèdent le très grand avantage d'être a priori bien plus facile à manipuler et plus naturelle que la troisième. Nous expliquons dans la section suivante les raisons qui nous ont amenés à choisir pourtant cette dernière modélisation.

1.5.4 CHOIX DE LA MEILLEURE MODÉLISATION

Ne pas fixer le nombre de combinaisons que nous allons produire, mais pénaliser la production, fournit a priori une très grande liberté. Mais comment choisir le coût de cette pénalité ? Et, sommes nous certains qu'il existera toujours une pénalité nous permettant de produire le nombre de combinaisons voulu ?

Avant de répondre à ces questions, nous remarquons d'abord que, pour la même pénalité K donnée, les deux premières modélisations nous fourniront des solutions optimales possédant le même coût. Pour s'en convaincre, il suffit de poser $f_{ij} = d_i x_{ij}$ pour tout $ij \in A$, $f_{jp} = \sum_{i \preceq j} d_i x_{ij}$, et $g_{jp} = x_{jj}$ pour tout $j \in V$. Nous voyons ainsi que les programmes linéaires associés à ces deux modélisations sont équivalents.

Pour répondre aux deux questions que nous avons posées, nous énonçons deux propriétés triviales sur les relations existant entre les solutions optimales de $(\mathcal{P}_K)$ et de $(\mathcal{P}_k)$.

Propriété 1.5.1

Soient $K \geq 0$, et $\bar{x}$ une solution optimale de $(\mathcal{P}_K)$.
$\bar{x}$ est solution optimale de $(\mathcal{P}_k)$ avec $k = \sum_{j \in V} \bar{x}_{jj}$.

Propriété 1.5.2

Soient $K > K' \geq 0$.
Soient $\bar{x}$ et $\bar{x}'$ des solutions optimales respectivement de $(\mathcal{P}_K)$ et de $(\mathcal{P}_{K'})$, et k et k' les valeurs $k = \sum_{j \in V} \bar{x}_{jj}$, et $k' = \sum_{j \in V} \bar{x}'_{jj}$
Alors l'une des propositions suivantes est vérifiée :

1. $k = k'$
2. $k < k'$ et $K' \leq \frac{z_k(\bar{x}) - z_{k'}(\bar{x}')}{k' - k} \leq K$

Pour prouver ces deux propriétés, il suffit de remarquer que, si $\bar{x}$ est une solution optimale de $(\mathcal{P}_k)$, alors nous avons $z_K(\bar{x}) = z_k(\bar{x}) + kK$.

La principale conséquence de cette dernière propriété nous permet de répondre négativement à la seconde question.

Conséquence 1.5.3

Soit $k \in \{|T|+1, ..., |V|-1\}$.
Soient z_{k-1}, z_k et z_{k+1} les valeurs de la fonction objectif de $(\mathcal{P}_{k-1})$, $(\mathcal{P}_k)$ et $(\mathcal{P}_{k+1})$ à l'optimum.
Si $z_k > \frac{z_{k-1}+z_{k+1}}{2}$ alors il n'existe aucune pénalité $K \geq 0$ tel que $(\mathcal{P}_K)$ admet une solution optimale dans laquelle k combinaisons sont produites.

Exemple : la figure 1.4 illustre un exemple où ce phénomène se produit.

Soit le graphe $G = (V, E)$ où $V = \{s_1, s_2, s_3, v, t\}$ et $E = \bigcup_{i=1}^{3} \{s_i v, s_i t\} \cup \{vt\}$ avec $c_{s_i} = 1$ pour $i = 1, 2, 3$, $c_v = 2$, $c_t = 3$, $d_{s_i} = 3$ pour $i = 1, 2, 3$, $d_v = d_t = 1$.

On a alors z_2 = 23, z_4 = 15 et $z_3 = 20 > \frac{z_2+z_4}{2} = 19$.

Il n'existe aucune pénalité $K \geq 0$ pour laquelle la production de trois combinaisons serait une solution optimale pour $(\mathcal{P}_K)$.

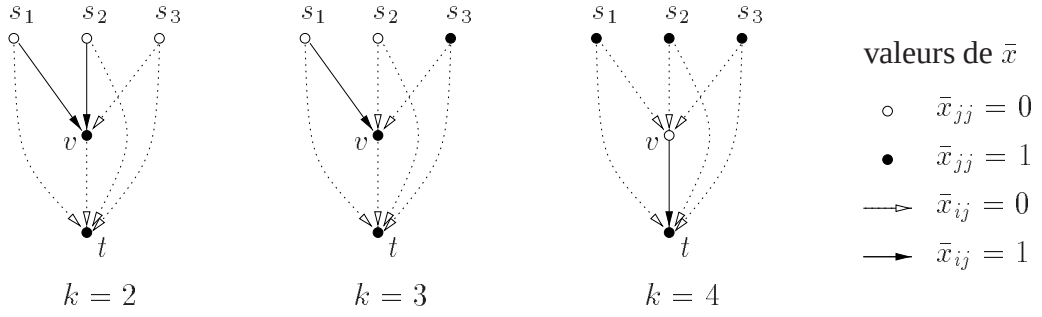


FIG. 1.4 – *Solutions optimales de $(\mathcal{P}_k)$*

Dans cet exemple, une seule valeur de k ne peut pas être atteinte à l'aide des modélisations avec pénalité, mais d'après la conséquence 1.5.3, il peut arriver que ce phénomène se produise pour plusieurs valeurs consécutives de k. Dans une telle situation, à moins d'énumérer toutes les solutions possibles, nous n'avons aucun recours pour trouver les solutions optimales que nous cherchons.

Ces observations nous ont donc poussés à privilégier la troisième modélisation. Cependant, nous utiliserons au chapitre 5 aussi la seconde afin de caractériser certaines contraintes valides du problème.

1.6 CONCLUSION

L'étude des différentes modélisations possibles nous a conduit à formuler le problème de la gestion de la diversité comme étant un cas particulier des problèmes de localisation k-médians.

Pour résoudre jusqu'à l'optimalité le programme linéaire en nombres entiers $(\mathcal{P}_k)$ nous allons utiliser un algorithme d'énumération implicite des solutions réalisables, ce sera le sujet du chapitre 5.

Cependant, notre but est de résoudre des instances de grande taille, possédant plusieurs milliers de combinaisons, et plusieurs centaines de milliers de substitutions possibles. Il est connu, et nous l'avons vérifié par des tests numériques, que la résolution de la simple relaxation linéaire de ce type de programme mathématique est une tâche des plus difficiles en pratique, même si la génération des variables et des contraintes est ici des plus triviales, car à chaque variables x_{ij} générée, nous devons aussi compléter le programme par la contrainte $x_{ij} \leq x_{jj}$.

Nous allons donc utiliser la théorie des relaxations lagrangiennes afin de réduire la taille de ce programme. Nous détaillerons cette méthode de fixation de variables au chapitre 3. Mais avant, nous consacrons un chapitre à quelques rappels sur les relaxations lagrangiennes, et à l'exposé de l'algorithme lagrangien que nous utilisons.

2

ALGORITHME GÉNÉRAL DES RELAXATIONS LAGRANGIENNES

Sommaire

Ce chapitre est consacré à un bref rappel sur la théorie des relaxations lagrangiennes. Dans une première section, nous exposons le principe de ces relaxations. Dans une seconde, nous décrivons quels types de relaxations lagrangiennes nous utilisons, quelles sont leurs particularités, et les motivations de notre choix. Nous présentons, à la troisième section, les trois méthodes permettant de mettre à jour les multiplicateurs lagrangiens. Puis nous terminons ce chapitre en décrivant l'algorithme que nous avons utilisé.

2.1 PRINCIPE

La notion de relaxation lagrangienne peut s'appliquer à un grand nombre de programmes d'optimisation, qu'ils soient quadratiques, linéaires ou simplement convexes. Nous l'abordons ici uniquement dans le cadre de la programmation linéaire en nombres binaires. Le lecteur souhaitant d'avantage de détails sur les relaxations lagrangiennes peut se reporter au chapitre de J. Beasley [Bea93b].

Nous considérons un programme (P) défini par

$$\begin{array}{rr} \text{minimiser} & cx \\ \text{sachant} & Ax \geq b \\ & Bx \geq d \\ & x \in \{0,1\}^n \end{array}$$

avec $c \in I\!R^n$, $A \in I\!R^{m \times n}$, $b \in I\!R^m$, $B \in I\!R^{p \times n}$, $d \in I\!R^p$

Une première relaxation bien connue de (P) consiste à remplacer la contrainte $x \in \{0,1\}^n$ par $0 \leq x \leq 1$. Il s'agit de la relaxation linéaire de (P). Cette relaxation peut être résolue très efficacement à l'aide d'algorithmes tels que le simplexe. Malheureusement, pour certains problèmes possédant un très grand nombre de variables et/ou de contraintes, cette résolution se révèle en pratique difficile. Comme nous l'avons vu au chapitre précédent, le problème de la gestion de la diversité est l'un de ces problèmes.

Une alternative est l'utilisation des relaxations lagrangiennes. Elles consistent à relâcher, i.e. supprimer, certaines contraintes, et à les introduire dans la fonction économique en leur affectant des pénalités (positives). Ces pénalités sont souvent appelées *multiplicateurs lagrangiens*, ou simplement *multiplicateurs*.

Dans cette section, nous noterons u, $u \geq 0$, le vecteur des multiplicateurs, et nous considérons des relaxations lagrangiennes $LR(u)$ où l'ensemble des

contraintes relâchées est représenté par $Bx \geq d$. Ces relaxations lagrangiennes $LR(u)$ sont définies par

$$\begin{array}{rc} \text{minimiser} & L(x,u) = cx + u(d - Bx) \\ \text{sachant} & Ax \geq b \\ & x \in \{0,1\}^n \end{array}$$

Quel que soit le vecteur de multiplicateurs u, si x^u désigne une solution optimale de $LR(u)$ alors la valeur $L(x^u, u)$ est une borne inférieure de la valeur de la fonction économique de (P) à l'optimum. Pour s'en convaincre, il suffit de remarquer que pour toute solution réalisable x de (P) nous avons $u(d - Bx) \leq 0$, et que x est une solution réalisable de $LR(u)$.

Deux question se posent :

1. Comment choisir le vecteur de multiplicateurs u pour obtenir la meilleure borne inférieure ?
2. Quelles contraintes doit-on relâcher ?

Si nous notons $\texttt{LB}(u)$ la valeur à l'optimum de la fonction économique de la relaxation lagrangienne $LR(u)$, alors répondre à la première question revient à résoudre le programme suivant, noté (PDL), et appelé *problème dual lagrangien*

$$\begin{array}{rc} \text{maximiser} & \texttt{LB}(u) \\ \text{sachant} & u \in I\!R^p_+ \end{array}$$

Malgré la simplicité de son écriture, résoudre ce programme est souvent une tâche des plus difficiles. La section 2.4 sera consacrée à l'exposé des trois principales méthodes heuristiques permettant de rechercher le meilleur vecteur de multiplicateurs u : les méthodes des ajustements, du sous-gradient, et des faisceaux.

L'idée principale de ces trois méthodes est de mettre à jour itérativement le vecteur des multiplicateurs u en fonction, notamment, de la valeur de $\texttt{LB}(u)$. Nous devons donc, à chaque itération, résoudre jusqu'à l'optimalité les relaxations lagrangiennes $LR(u)$. Ceci présuppose que ces relaxations lagrangiennes puissent être résolues très rapidement. Le choix des contraintes relâchées est donc guidé par cette première observation. De plus, en pratique, il est important de préserver une taille raisonnable au vecteur des multiplicateurs. Nous devons donc limiter le nombre des contraintes relâchées. Ce choix doit, bien sûr, aussi tenir compte de la qualité des bornes inférieures que nous allons obtenir. Il est connu que, quelles que soient les contraintes relâchées, la valeur maximale de $\texttt{LB}(.)$ est supérieure ou égale à la borne inférieure que nous aurions obtenue à l'aide de la relaxation linéaire de (P). Et nous avons l'égalité si les relaxations lagrangiennes $LR(u)$ possèdent la propriété d'intégralité, i.e., si, pour toute valeur $\bar{u}$ de u, il existe une

solution optimale $x^{\bar{u}}$ de $LR(\bar{u})$ qui est entière (voir [Geo74]). Il paraît donc préférable de choisir des relaxations lagrangiennes ne possédant pas cette propriété.

La réponse à la seconde question n'est donc pas évidente. Nous décrirons à la section 2.2 les relaxations lagrangiennes que nous avons choisies pour le problème de la gestion optimale de la diversité.

Le principe des relaxations lagrangiennes est l'une des techniques d'approximation les plus employées depuis les vingts dernières années. Leurs domaines d'application sont très nombreux. Par exemple, B. Dorhout [Dor95] les utilisent pour résoudre un problème de commande de fer étamé. Les clients spécifient quelle quantité et quels types de fer étamé ils souhaitent (dimension, épaisseur de l'étain, etc...). Et l'industriel doit commander à la fonderie les tôles brutes qu'il découpera. Sachant que certains types de fer étamé peuvent être remplacés par d'autres, et qu'il existe des remises sur la quantité, il cherche à maximiser son bénéfice. Cet exemple illustre l'une des nombreuses exploitations possibles des relaxations lagrangiennes dans l'industrie. Les problèmes rencontrés par les opérateurs de télécommunications, tels que les problèmes de dimensionnement de réseaux, de localisations d'antennes, etc..., sont l'une des sources de problèmes où l'utilisation des relaxations lagrangiennes s'est révélée des plus profitables. Comme nous le verrons à la section suivante, elles permettent aussi d'obtenir très facilement des bornes inférieures pour des problèmes de très grandes tailles, tels que les problèmes de localisation. J.M. Mulvey et H.P. Crowder [MC79] présentent une heuristique basée sur les relaxations lagrangiennes permettant de résoudre le problème des grappes. Ce problème consiste à partitionner un ensemble de points en m sous-ensembles, appelés *grappes*, avec m donné. L'objectif est d'obtenir les grappes les plus compactes possibles. J.E. Beasley [Bea93a] donne un algorithme générique exploitant les relaxations lagrangiennes pour résoudre tous les types de problèmes de localisation.

À la section suivante, nous détaillons les relaxations lagrangiennes que nous utilisons pour le problème de la gestion de la diversité.

2.2 CHOIX DES RELAXATIONS LAGRANGIENNES UTILISÉES

Nous avons choisi de modéliser le problème de la gestion de la diversité à l'aide du programme linéaire en nombres binaires $(\mathcal{P}_k)$ suivant :

$$\text{minimiser} \quad z_k(x) = \sum_{i \in V} \sum_{j \succeq i} c_j d_i x_{ij} \tag{2.1}$$

$$\text{sachant} \quad \sum_{j \in V} x_{jj} = k \tag{2.2}$$

$$\sum_{j \succeq i} x_{ij} = 1 \qquad \forall i \in V \tag{2.3}$$

$$x_{ij} \leq x_{jj} \qquad \forall j \in V, i \prec j \tag{2.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall j \in V, i \preceq j \tag{2.5}$$

Les relaxations lagrangiennes que nous considérons consistent à relâcher les contraintes (2.3), en pénalisant leurs éventuelles violations par des multiplicateurs u_i, avec $i \in V$.

Pour un vecteur de multiplicateur $\bar{u}$ donné, nous obtenons alors le programme $LR(\bar{u})$ suivant :

$$\text{minimiser} \quad L(x, \bar{u}) = \sum_{i \in V} \sum_{j \succeq i} c_j d_i x_{ij} + \sum_{i \in V} \bar{u}_i (1 - \sum_{j \succeq i} x_{ij}) \tag{2.6}$$

$$\text{sachant} \quad \sum_{j \in V} x_{jj} = k \tag{2.7}$$

$$x_{ij} \leq x_{jj} \qquad \forall j \in V, i \prec j \tag{2.8}$$

$$x_{ij} \in \{0,1\} \qquad \forall j \in V, i \preceq j \tag{2.9}$$

Est-il difficile de résoudre cette relaxation lagrangienne ?
Pour répondre à cette question, nous définissons d'abord deux notations :

1. Soit, pour tout $j \in V$ et tout $i \prec j$, $\bar{\gamma}_{ij} = c_j d_i - \bar{u}_i$ le *coût réduit lagrangiens* de l'arc ij associé à $\bar{u}$
2. Soit, pour tout $j \in V$, $S[j] = c_j d_j - \bar{u}_j + \sum_{i \prec j} (\bar{\gamma}_{ij})^-$

avec, pour tout réel α, la notation $(\alpha)^-$ pour désigner α si $\alpha \leq 0$, et 0 sinon.

Propriété 2.2.1

Soit R l'ensemble des k combinaisons possédant les plus petites valeurs $S[.]$. Si $x^{\bar{u}}$ désigne la solution de la relaxation lagrangienne $LR(\bar{u})$ définie par

Pour tout $j \in V$, $x^{\bar{u}}_{jj} = \begin{cases} 1 & \text{si } j \in R \\ 0 & \text{sinon} \end{cases}$

Pour tout $j \in V$ et tout $i \prec j$, $x^{\bar{u}}_{ij} = \begin{cases} 1 & \text{si } j \in R \text{ et } \bar{\gamma}_{ij} < 0 \\ 0 & \text{sinon} \end{cases}$

Alors $x^{\bar{u}}$ est une solution optimale de $LR(\bar{u})$

Preuve de 2.2.1 :

Réordonnons d'abord les termes de la fonction économique (2.6), afin d'obtenir une nouvelle écriture de $L(x, \bar{u})$:

$$L(x, \bar{u}) = \sum_{j \in V} \left((c_j d_j - \bar{u}_j) x_{jj} + \sum_{i \prec j} \bar{\gamma}_{ij} x_{ij} \right) + \sum_{i \in V} \bar{u}_i \qquad (2.10)$$

Noter que le terme $\sum_{i \in V} \bar{u}_i$ est une constante quand $\bar{u}$ est donné.

Pour prouver cette proposition il suffit de remarquer, d'une part, que, pour tout $j \in V$, $S[j]$ correspond à la valeur de la fonction économique, à l'optimum, du programme

$$\begin{aligned} \text{minimiser} \quad & (c_j d_j - \bar{u}_j) + \sum_{i \prec j} \bar{\gamma}_{ij} x_{ij} \\ \text{sachant} \quad & x_{ij} \in \{0, 1\} \qquad \forall j \in V, i \preceq j \end{aligned}$$

et d'autre part que la relaxation lagrangienne $LR(\bar{u})$ est équivalente au programme

$$\begin{aligned} \text{minimiser} \quad & \sum_{j \in V} S[j] x_{jj} + \sum_{i \in V} \bar{u}_i \\ \text{sachant} \quad & \sum_{j \in V} x_{jj} = k \\ & x_{jj} \in \{0, 1\} \qquad \forall j \in V \end{aligned}$$

♢

La résolution des relaxations lagrangiennes que nous avons choisies est donc des plus simples. Il suffit de procéder en quatre étapes :

1. calculer les coûts réduits lagrangiens

2. calculer les valeurs $S[j]$ pour tout $j \in V$
3. définir R comme étant l'ensemble des k combinaisons possédant les plus petites valeurs $S[.]$.
4. calculer $\texttt{LB}(\bar{u}) = \sum_{j \in R} S[j] + \sum_{i \in V} \bar{u}_i$

Remarque 1 : Une conséquence directe de la proposition 2.2.1 est le fait que les relaxations lagrangiennes que nous avons choisies possèdent la propriété d'intégralité. Nous ne pouvons donc pas espérer obtenir des bornes inférieures de meilleure qualité que celle fournie par la relaxation linéaire de $(\mathcal{P}_k)$. Cependant, ces relaxations lagrangiennes demeurent très intéressantes grâce à la simplicité de leur résolution.

Remarque 2 : Si nous avions relâché la contrainte (2.2), nous aurions obtenu la modélisation en problème de localisation avec coût fixe, i.e. le programme $(\mathcal{P}_K)$, avec K le multiplicateur lagrangien. Comme nous l'avons vu au chapitre précédent, parfois cette modélisation ne permet pas de trouver des solutions réalisables de $(\mathcal{P}_k)$, le nombre de combinaisons produites étant strictement inférieur, ou strictement supérieur à k, suivant la valeur de K.

Remarque 3 : Dans le cas général, si la solution optimale d'une relaxation lagrangienne est réalisable pour le problème initial, alors nous ne pouvons pas conclure qu'elle est optimale pour ce problème initial. En effet, si certaines contraintes relâchées sont des inéquations et si cette solution ne vérifie pas l'une d'elle à l'égalité, alors la valeur de la fonction économique de la relaxation lagrangienne diffère de celle du programme initial. Nos relaxations sont particulières dans le sens que nous n'avons relâché que des équations. Donc, contrairement au cas général, s'il existe un vecteur lagrangien $\bar{u}$ tel que la solution $x^{\bar{u}}$ que nous avons définie est réalisable pour $(\mathcal{P}_k)$ alors elle est optimale pour $(\mathcal{P}_k)$.

Remarque 4 : Pour une combinaison terminale $t \in T$, la contrainte (2.3) qui lui est associée se résume à $x_{tt} = 1$. Il est donc inutile de la relâcher, et de définir un multiplicateur u_t. De plus, nous pouvons exiger que t appartienne systématiquement aux ensembles R qui définissent les solutions optimales x^u de $LR(u)$, et ce, quelle que soit la valeur de $S[t]$. Ceci n'altère pas la simplicité de la résolution des relaxations lagrangiennes. Plus généralement, nous verrons au chapitre suivant qu'il est possible de fixer définitivement la valeur de certaines variables, c'est ce que nous appelons *la fixation de variables*. Si une variable x_{jj} est fixée à 1 (nous dirons alors que la combinaison j est fixée à 1), alors nous avons la certitude que j est produite, et nous pouvons, quelle que soit la valeur $S[j]$, exiger que j soit l'une des k combinaisons constituant les ensembles R. Nous pouvons tenir

compte aussi des autres fixations de variables dans le calcul des $S[.]$ et dans le choix des éléments de R. Pour ne pas alourdir les raisonnements développés dans ce chapitre, nous n'exploiterons que les combinaisons fixées à 1, dont l'ensemble sera noté V^1.

De cette dernière remarque, découle le théorème suivant :

Théorème 2.2.2

Il existe un vecteur de multiplicateurs lagrangiens $\bar{u}$ solution optimale du problème dual lagrangien (PDL) qui vérifie, pour tout $i \in V \setminus V^1$

$$c_{j(i)}d_i \leq \bar{u}_i \leq c_{t(i)}d_i$$

avec $j(i)$ la combinaison la moins chère pouvant remplacer i, et $t(i) \in V^1$ la combinaison fixée à 1 la moins chère pouvant remplacer i

Preuve de 2.2.2 : Soit $\bar{u}$ un vecteur de multiplicateurs optimal

1. S'il existe un multiplicateur $\bar{u}_i$ tel que $\bar{u}_i > c_{t(i)}d_i$, alors définissons $\bar{u}'$ par $\bar{u}'_j = \bar{u}_j$, pour tout $j \neq i$, et $\bar{u}'_i = c_{t(i)}d_i$. Il est facile de voir que diminuer la valeur de $\bar{u}_i$ entraîne l'augmentation de la valeur des $S[j]$, pour toutes les combinaisons $j \succeq i$ (ou la laisse égale à son ancienne valeur), et cet accroissement est de $\bar{u}_i - c_{t(i)}d_i$ pour $t(i)$. Sachant que la combinaison $t(i)$ est fixée à 1, elle appartient aux k combinaisons choisies, et la diminution de la somme des multiplicateurs lagrangiens sera ainsi compensée par l'accroissement de $S[t(i)]$. Par conséquent, nous avons $\texttt{LB}(\bar{u}') \geq \texttt{LB}(\bar{u})$. Or $\bar{u}$ était solution optimale du problème dual lagrangien, donc $\bar{u}'$ l'est aussi. En réitérant ce processus, nous obtenons un vecteur de multiplicateurs vérifiant $\bar{u}_i \leq c_{t(i)}d_i$ pour tout $i \in V \setminus V^1$.
2. S'il existe un multiplicateur $\bar{u}_i$ tel que $\bar{u}_i < c_{j(i)}d_i$, alors définissons $\bar{u}'$ par $\bar{u}'_j = \bar{u}_j$, pour tout $j \neq i$, et $\bar{u}'_i = c_{j(i)}d_i$. D'une part, la somme des multiplicateurs croît de $c_{j(i)}d_i - \bar{u}_i$, et d'autre part, $S[i]$ décroît de la même valeur. En remarquant que les $S[j]$ reste inchangées, pour toutes les combinaisons $j \succ i$, nous en déduisons que $\texttt{LB}(\bar{u}') \geq \texttt{LB}(\bar{u})$. La solution $\bar{u}$ étant optimale du problème dual lagrangien, la solution $\bar{u}'$ l'est donc aussi. En réitérant ce processus, nous obtenons un vecteur de multiplicateurs vérifiant $\bar{u}_i \geq c_{j(i)}d_i$ pour tout $i \in V \setminus V^1$.

$\diamondsuit$

Le type de relaxations lagrangiennes avec lesquelles nous allons travailler étant choisi, nous nous intéressons maintenant à la recherche du meilleur vecteur

de multiplicateurs lagrangiens.

2.3 RÉSOUDRE LE PROBLÈME DUAL LAGRANGIEN

La recherche du vecteur de multiplicateurs $\bar{u}$ qui nous donnera la plus grande borne inférieure $\texttt{LB}(\bar{u})$ est un problème en soit. Il s'agit de ce que nous avons appelé *problème dual lagrangien* (PDL).

L'algorithme général permettant d'exploiter les relaxations lagrangiennes se décompose suivant le schéma suivant :

Algorithme 1 Algorithme général des relaxations lagrangiennes

Initialiser le vecteur des multiplicateurs $\bar{u}^0$
$t \leftarrow 0$
répéter
  Résoudre $LR(\bar{u}^t)$
  si la solution $x^{\bar{u}^t}$ est réalisable pour $(\mathcal{P}_k)$ **alors**
    $x^{\bar{u}^t}$ est optimale pour $(\mathcal{P}_k)$, STOP
  fin si
  Exploiter $\bar{u}^t$
  Mise à jour des multiplicateurs : calculer $\bar{u}^{t+1}$
  $t \leftarrow t + 1$
jusqu'à ce que le test d'arrêt soit vérifié

Exploiter la solution $\bar{u}^t$ consiste à

1. mettre à jour GLB, la meilleure borne inférieure trouvée jusqu'à présent, si $\texttt{LB}(\bar{u}^t) > \texttt{GLB}$
2. construire une solution réalisable de $(\mathcal{P}_k)$ à partir de $\bar{u}^t$ (voir chapitre 4). Si nécessaire, mettre à jour GUB, valeur de la meilleure solution réalisable de $(\mathcal{P}_k)$ trouvée jusqu'à présent.
3. essayer de fixer des variables (voir chapitre 3)

Le test d'arrêt dépend de la méthode employée pour calculer $\bar{u}^{t+1}$.

Les trois méthodes principales que nous employons sont décrites à la section suivante.

2.4 MISE À JOUR DES MULTIPLICATEURS LAGRANGIENS

Nous présentons dans cette section, trois méthodes heuristiques permettant de mettre à jour les multiplicateurs lagrangiens : la méthode des ajustements (section 2.4.1), celle du sous-gradient (section 2.4.2) puis celle des faisceaux (section 2.4.3).

2.4.1 MÉTHODE DES AJUSTEMENTS

La méthode des ajustements consiste à définir le vecteur des multiplicateurs $\bar{u}^{t+1}$ en ne changeant qu'une seule composante de $\bar{u}^t$. En d'autres termes, à chaque itération nous ne nous intéressons qu'à un seul multiplicateur u_i. De plus, un changement n'est effectué que s'il est profitable, c'est-à-dire si la valeur $\texttt{LB}(\bar{u}^{t+1})$ est strictement supérieure à $\texttt{LB}(\bar{u}^t)$. L'algorithme 1 se termine si plus aucune amélioration n'est détectée.

Notons $S^t[.]$, et $S^{t+1}[.]$ les valeurs de $S[.]$ associées respectivement à $\bar{u}^t$ et à $\bar{u}^{t+1}$. Notons aussi R^t et R^{t+1} les ensembles des k combinaisons associés à la définition respectivement de $x^{\bar{u}^t}$ et $x^{\bar{u}^{t+1}}$.

Si nous considérons les valeurs $S[.]$, en supposant tous les multiplicateurs fixés, sauf u_i, nous obtenons les trois types de fonctions représentées à la figure 2.1. En effet, par définition des $S[j]$, nous avons trois cas possibles : soit $j = i$, et $S[j]$ est alors une fonction linéaire de pente -1, soit $j \succ i$, et alors $S[j]$ est une fonction constante tant que u_i est inférieur à $c_j d_i$, car pour de telles valeurs de u_i nous avons $(c_j d_i - u_i)^- = 0$, puis linéaire décroissante de pente -1 au delà de ce seuil, car alors $(c_j d_i - u_i)^- = c_j d_i - u_i$, soit j n'est pas supérieure ou égale à i, et u_i n'intervient pas dans le calcul de $S[j]$.

Remarquer que ces trois fonctions sont décroissantes, au sens large. De plus, quelle que soit la nouvelle valeur $\bar{u}_i^{t+1}$ du multiplicateur u_i, nous avons

1. $S^{t+1}[i] = (S^t[i] + \bar{u}_i^t) - \bar{u}_i^{t+1}$
2. pour tout $j \succ i$,

$$S^{t+1}[j] = \begin{cases} (S^t[j] - (c_j d_i - \bar{u}^t)^-) & \text{si } \bar{u}_i^{t+1} \leq c_j d_i \\ (S^t[j] - (c_j d_i - \bar{u}^t)^- + c_j d_i) - \bar{u}_i^{t+1} & \text{si } \bar{u}_i^{t+1} > c_j d_i \end{cases}$$

3. pour les autres combinaisons j, $S^{t+1}[i] = S^t[i]$

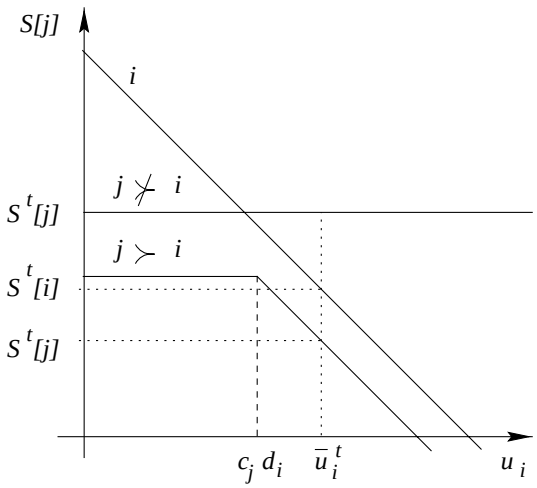


FIG. 2.1 – S *fonction de* u_i

L'ensemble R^t associé à la solution optimale de $LR(\bar{u}^{t+1})$ a été défini comme étant l'ensemble des k combinaisons possédant les plus petits $S^t[.]$ (ou fixée à 1, d'après la remarque 4). Il est représenté à la figure 2.2 par des points noirs. Nous utiliserons les notations $Smax1^t$ et $Smin0^t$ pour désigner respectivement la plus grande valeur des $S^t[j]$ pour $j \in R^t$, et la plus petite valeur des $S^t[j]$ pour $j \notin R^t$.

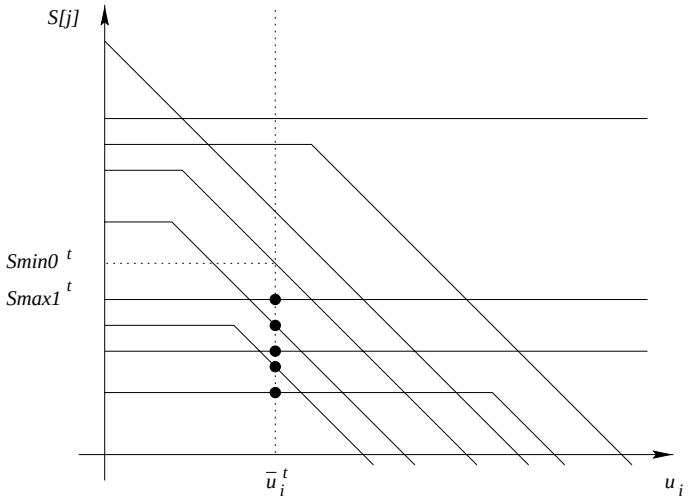


FIG. 2.2 – *Solution optimale de* $LR(\bar{u}^t)$

Nous allons étudier dans les sections suivantes l'influence, sur R^t et sur $\texttt{LB}(\bar{u}^t)$, d'une augmentation ou d'une diminution du multiplicateur u_i.

2.4.1.1 AUGMENTER UN MULTIPLICATEUR

Nous supposons que le multiplicateur ait été augmenté, c'est-à-dire que nous posons $\bar{u}_i^{t+1} = \bar{u}_i^t + \delta$, avec $\delta > 0$, les autres multiplicateurs restant inchangés.

La somme des multiplicateurs lagrangiens croît de δ.

Sachant que les $S[.]$ sont des fonctions décroissantes en fonction de u_i, nous avons $\texttt{LB}(\bar{u}^{t+1}) \leq \texttt{LB}(\bar{u}^t) + \delta$.

Notre but étant de trouver une valeur de δ maximisant $\texttt{LB}(\bar{u}^{t+1})$, nous ne considérons que des combinaisons i n'appartenant pas à R^t et telle que pour toute combinaison $j \succ i$ appartenant à R^t, on a $\bar{u}^t < c_j d_i$ (et j non fixée à 1). En effet, comme le montre la figure 2.2, si i ne vérifiait pas cette condition, alors le gain apporté par la somme des multiplicateurs lagrangiens serait compensée (au moins une fois) par la décroissance de $S[i]$ ou d'un tel $S[j]$.

Nous définissons δ comme étant la plus grande valeur telle que

1. $S^{t+1}[i] \geq Smax1^t$
2. pour tout $j \succ i,\ \ j \notin R^t,\ \ S^{t+1}[j] \geq Smax1^t$
3. pour tout $j \succ i,\ \ j \in R^t,\ \ S^{t+1}[j] = S^t[j]$

Au delà de cette valeur, nous serions à nouveau dans les conditions que nous énoncions plus haut (voir figure 2.3).

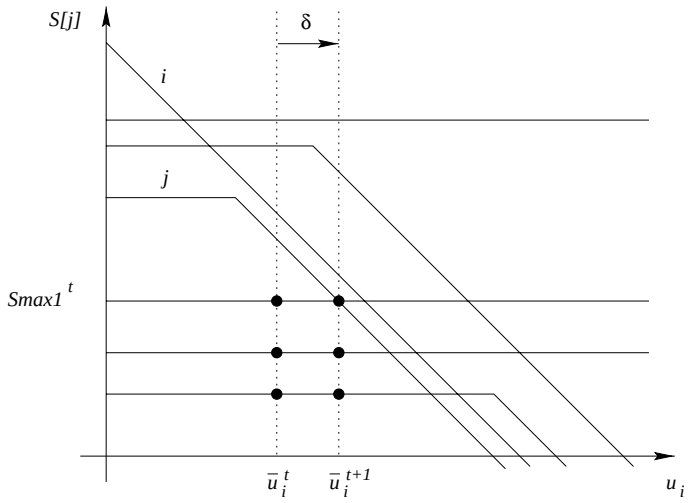


FIG. 2.3 – *Augmenter le multiplicateur* u_i

D'après ce que nous avons vu précédemment, la valeur de δ est donc le minimum entre les valeurs suivantes :

1. $S^t[i] - Smax1^t$

2. pour tout $j \succ i, \quad j \notin R^t$,
   (a) si $\bar{u}_i^t \leq c_j d_i$, $S^t[j] + c_j d_i - \bar{u}_i^t - Smax1^t$
   (b) si $\bar{u}_i^t > c_j d_i$, $S^t[j] - Smax1^t$
3. pour tout $j \succ i, \quad j \in R^t$, $c_j d_i - \bar{u}^t$

Dans l'exemple de la figure 2.3, la valeur de δ est été donnée par une combinaison $j \succ i$ n'appartenant pas à R^t et telle que $\bar{u}_i^t > c_j d_i$.

Remarque : par construction, nous avons $Smax1^{t+1} = Smax1^t$, et donc $R^{t+1} = R^t$. De plus, comme la somme des $S[j]$ pour $j \in R^{t+1}$ reste inchangée, nous avons $\texttt{LB}(\bar{u}^{t+1}) = \texttt{LB}(\bar{u}^t) + \delta$. Bien que R^{t+1} soit égal à R^t, la solution optimale a été modifiée, car les coûts réduits lagrangiens des arcs ij qui étaient égaux à $c_j d_i - \bar{u}_i^t$ sont devenus $c_j d_i - \bar{u}_i^t - \delta$.

2.4.1.2 DIMINUER UN MULTIPLICATEUR

Nous supposons que le multiplicateur ait été diminué, c'est-à-dire que nous posons $\bar{u}_i^{t+1} = \bar{u}_i^t - \delta$, avec $\delta > 0$, les autres multiplicateurs restant inchangés.

La somme des multiplicateurs lagrangiens décroît de δ.

Comme nous voulons obtenir une nouvelle valeur de la relaxation lagrangienne meilleure que l'actuelle, la diminution du multiplicateur u_i doit, non seulement, compenser la perte dûe à la somme des multiplicateurs, mais aussi générer un gain. Par conséquent, nous ne considérons que les combinaisons i vérifiant au moins l'une des deux propriétés suivantes (l'une servant à compenser la perte de δ, et l'autre à générer du gain) :

1. $i \in R^t$ et il existe au moins une combinaison $j \succ i$, telle que $j \in R^t$ et $c_j d_i < \bar{u}_i^t$
2. il existe deux combinaisons $j \succ i$, telle que $j \in R^t$ et $c_j d_i < \bar{u}_i^t$

Si n_i désigne le nombre de combinaisons j vérifiant la première propriété, et ce nombre moins une unité pour la seconde propriété, alors, pour les petites valeurs de δ, le gain est de $n_i \delta$.

Pour obtenir le plus grand gain, c'est-à-dire chercher la plus grande valeur $\texttt{LB}(\bar{u}^{t+1})$, nous devons choisir δ comme étant la plus grande valeur pour laquelle au moins l'une des deux propriétés reste vraie.

Nous définissons δ comme étant la seconde plus grande valeur de l'ensemble constitué de :

cas 1 : $i \in R^t$,
   (a) $Smin0^t - S^t[i]$
   (b) $\min\{\ \bar{u}_i^t - c_j d_i\ ,\ Smin0^t - S^t[j]\ \}$ pour les combinaisons $j \succ i$ telles que $j \notin V^1$, $j \in R^t$ et $c_j d_i < \bar{u}_i^t$

(c) $c_j d_i - \bar{u}_i^t$ pour toutes les combinaisons $j \succ i$ fixées à 1

cas 2 : $i \notin R^t$,

(a) $\min\{\ \bar{u}_i^t - c_j d_i\ ,\ Smin0^t - S^t[j]\ \}$ pour les combinaisons $j \succ i$ telles que $j \notin V^1$, $j \in R^t$ et $c_j d_i < \bar{u}_i^t$

(b) $c_j d_i - \bar{u}_i^t$ pour toutes les combinaisons $j \succ i$ fixées à 1

La valeur de la relaxation lagrangienne $LR(\bar{u}^{t+1})$ est alors supérieure ou égale à $\texttt{LB}(\bar{u}^t) + \delta$. Pour connaître sa valeur exacte nous résolvons complètement $LR(\bar{u}^{t+1})$. Noter que seule la valeur $S[.]$ de i, et de certains $j \succ i$ changent. Cette résolution peut donc être effectuée très facilement.

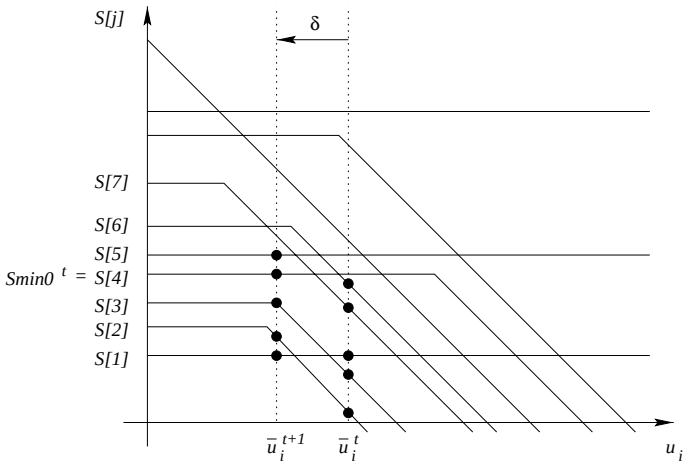


FIG. 2.4 – *Diminuer le multiplicateur* u_i

Dans l'exemple de la figure 2.4, nous sommes dans le cas 2 : $i \notin R^{t+1}$. Nous avons $R^t = \{2, 3, 1, 7, 6\}$, et toutes ces combinaisons j, sauf 2, peuvent remplacer i, et sont telles que $c_j d_i < \bar{u}_i^t$ (non les supposons non fixées à 1). Nous avons $Smin0^t = S^t[4]$. L'ensemble des valeurs qui nous permet de définir δ est, dans l'ordre croissant, $\{Smin0^t - S^t[6], Smin0^t - S^t[7], \bar{u}_i^t - c_3 d_i, \bar{u}_i^t - c_2 d_i\}$. Nous choisissons donc $\delta = \bar{u}_i^t - c_3 d_i$. Les deux combinaisons qui ont permis dans tout l'intervalle $[\bar{u}_i^t - \delta, \bar{u}_i^t]$ de compenser la perte dûe à la somme des multiplicateurs lagrangiens, et d'assurer un gain sont les combinaisons 2 et 3. En deçà de $\bar{u}_i^t - \delta = c_3 d_i$, la valeur $S[3]$ n'augmente plus et ne peut plus assurer le gain.

Remarque : Il peut arriver que plusieurs itérations successives de l'algorithme 1 (section 2.3), modifient le même multiplicateur, car dans le cas d'une diminution, nous n'avons pas l'assurance que la valeur de δ soit la meilleure possible. Ce phénomène se produit lorsque $Smin0^t$ correspond à une combinaison $j \succ i$ telle que $\bar{u}_i^{t+1} > c_j d_i$.

Cette méthode des ajustements est extrêmement rapide. Malheureusement, sa portée est aussi très limitée car elle définit un optimum local. Nous l'utilisons simplement pour améliorer la qualité des vecteurs lagrangiens obtenus par les deux autres méthodes suivantes, à commencer par celle du sous-gradient.

2.4.2 MÉTHODE DU SOUS-GRADIENT

La méthode consiste, comme celle des faisceaux que nous décrivons dans la section suivante, à changer simultanément tous les multiplicateurs lagrangiens.

Son nom provient de l'utilisation du vecteur sous-gradient G représentant la violation des contraintes relâchées. À chaque itération t, nous définissons un sous-gradient G^t par

$$G_i^t = 1 - \sum_{j \succeq i} x_{ij}^{\bar{u}^t}$$

Remarquer que G_i^t peut être égal soit à 0, si la contrainte est vérifiée, soit à 1 si i n'est ni produite ni remplacée, soit à un nombre entier strictement négatif indiquant que i est remplacée plusieurs fois, ou simultanément produite et remplacée.

Par conséquent, si le sous-gradient G^t est identiquement nul, la solution $x^{\bar{u}^t}$ est une solution réalisable pour $(\mathcal{P}_k)$. Vu que nous n'avons relâché que des équations, elle est optimale pour $(\mathcal{P}_k)$, et l'algorithme s'arrête.

Dans le cas contraire, nous définissons le nouveau vecteur de multiplicateurs par

$$\bar{u}^{t+1} = \bar{u}^t + \rho . G^t \tag{2.11}$$

où ρ est un réel reflétant non seulement la qualité de la solution optimale lagrangienne courante, mais aussi la violation des contraintes relâchées, il est défini par

$$\rho = \pi \, \frac{\texttt{GUB} - \texttt{LB}(\bar{u}^t)}{\sum_{i \in V} (G_i^t)^2}$$

où π est un paramètre diminuant régulièrement au cours de l'exécution de l'algorithme, et permettant de réduire la taille de l'intervalle dans lequel évoluent les multiplicateurs lagrangiens.

Ce paramètre π est mis à jour après un certain nombre d'itérations sans amélioration des meilleures bornes inférieures et supérieures. Dans la littérature (voir par exemple[Bea93b]), il est conseillé de l'initialiser à 2, et de le diviser par 2 toutes les 20 ou 30 itérations sans amélioration. Pour le problème de la gestion de la diversité, il est apparu que 80 voire 100 itérations étaient préférables. De plus,

nous avons testé avec succès une stratégie consistant à diminuer, au moment de la mise à jour de π, ce nombre d'itérations maximales sans amélioration. Nous l'initialisons à 120 ou 150, puis nous le multiplions par un facteur de réduction inférieur à 1 (égale à 0.95, par exemple). En effet, lorsque π est très petit, la vitesse d'augmentation de `GLB` est faible. Cette technique permet d'arrêter prématurément la boucle du sous-gradient, et d'appeler ensuite la procédure des ajustements beaucoup plus rapide.

Outre le cas où le sous-gradient est identiquement nul, il existe trois tests d'arrêt. Le premier est la limitation du nombre d'itérations. Le second est la définition d'une valeur minimale du paramètre π. Les deux étant très liés, il suffit souvent d'arrêter l'algorithme dès que π est inférieur à 0.001. Le troisième est un test d'optimalité : si $\texttt{LB}(\bar{u}^t) = \texttt{GUB}$ alors la solution réalisable de $(\mathcal{P}_k)$ qui est associée à `GUB` est une solution optimale.

Il peut paraître intéressant d'exploiter le théorème 2.2.2 afin de limiter l'intervalle d'étude des multiplicateurs, et donc, a priori, d'accélérer l'algorithme. Pour cela, nous avons effectué plusieurs tests, soit en tronquant les valeurs obtenues par la formule 2.11, soit en redéfinissant la valeur du coefficient ρ, mais dans les deux cas, les résultats ont été décevants, voire même, pour la seconde option, contraire à ce que nous espérions : l'amélioration de la borne inférieure `GLB` étant beaucoup plus lente qu'auparavant. Nous avons donc décidé de ne pas tenir compte de ce théorème pour la méthode du sous-gradient.

Cette méthode est relativement efficace. Cependant, il arrive parfois que des fixations de variables provoquent une telle réduction de la taille du problème que la meilleure borne inférieure atteignable désormais soit très supérieure à celle que nous pouvions espérer obtenir avant ces fixations. Malheureusement, ce phénomène arrive souvent lorsque π est petit, et l'algorithme s'arrête sans avoir réussi à approcher cette nouvelle valeur. Une première idée serait de faire un nouvel appel à l'algorithme lagrangien en réinitialisant tous les paramètres, mais la méthode du sous-gradient est relativement coûteuse en temps, et nous n'avons aucune garantie quant à la qualité du vecteur de multiplicateur qu'elle fournit. Nous faisons donc appel à la méthode des faisceaux pour confirmer que la valeur de `GLB` est la meilleure que l'on puisse obtenir, et si ce n'est pas le cas, elle va nous fournir une borne inférieure de qualité garantie. Cette méthode est le sujet de la section suivante.

2.4.3 MÉTHODE DES FAISCEAUX

La troisième méthode que nous avons utilisée pour mettre à jour les multiplicateurs lagrangiens est celle dites *des faisceaux*.

Cette méthode des faisceaux s'appuie sur le théorème suivant, qui est un résultat connu de relaxations lagrangiennes :

Théorème 2.4.1

La fonction qui a tout $u \in I\!R^n$ associe $\mathtt{LB}(u)$ est une fonction concave

Preuve de 2.4.1 : Soit $u, u' \in I\!R^n$, et $u" = (u + u')/2$. On a

$$\begin{aligned}
\mathtt{LB}(u") &= L(x^{u"}, u") \\
&= cx^{u"} + u"(b' - A'x^{u"}) \\
&= (cx^{u"} + u(b' - A'x^{u"}) + cx^{u"} + u'(b' - A'x^{u"}))/2 \\
&= (L(x^{u"}, u) + L(x^{u"}, u'))/2 \\
&\geq (\mathtt{LB}(u) + \mathtt{LB}(u'))/2
\end{aligned}$$

$\diamondsuit$

Cette méthode est très puissante dans le sens que, contrairement aux deux précédentes, elle est convergente, c'est à dire qu'il est possible de garantir la qualité du vecteur de multiplicateurs qu'elle fournit.

La singularité de cette méthode par rapport à celle du sous-gradient, est de garder un historique des solutions lagrangiennes. Le nouveau vecteur lagrangien $\bar{u}^{t+1}$ est en effet calculé à l'aide non seulement de $\bar{u}^t$ et du sous-gradient G^t, mais aussi de tous les vecteurs $\bar{u}^{t'}$ et $G^{t'}$ définis jusqu'à présent.

La théorie, sur laquelle s'appuie cette méthode, est trop complexe pour que nous l'abordions ici. Le lecteur intéressé pourra se reporter au livre de J.B. Hiriart-Urruty et C. Lemaréchal [HUL93], ainsi qu'à l'article de C. Lemaréchal et C. Sagastizábal [LS97].

Pour la tester, C. Lemaréchal nous a fournit un code que nous utilisons comme une *boîte noire*. Cette boite noire, qui est en faite une fonction que nous appellerons `faisceau`, communique avec notre programme à l'aide des entrées et sorties suivantes (nous ne mentionnons que les principales) :

entrées :

- un compteur t indiquant le numéro de l'itération en cours
- le vecteur des multiplicateurs $\bar{u}^t$
- la valeur de $\mathtt{LB}(\bar{u}^t)$
- le vecteur sous-gradient G^t

sorties :

- le nouveau vecteur des multiplicateurs $\bar{u}^{t+1}$

- un indicateur signalant si `faisceau` demande une nouvelle itération ou non

Il existe deux critères d'arrêt : soit le nombre d'appels à la procédure est supérieur au nombre d'itérations que nous nous étions fixé (analogue à celui de la méthode du sous gradient), soit la convergence du vecteur des multiplicateurs a été prouvée. Bien sûr, cette convergence ne peut être qu'approximative; des variables en entrée nous permettent de choisir l'approximation maximale souhaitée. Pour affiner la meilleure solution ainsi obtenue, nous faisons appel systématiquement à la méthode des ajustements.

La puissance de la méthode des faisceaux réside en sa convergence, malheureusement le code dont nous disposons ne permet pas la fixation des variables. La dimension de l'espace étudié doit en effet demeurer inchangée tout au long d'une boucle de faisceaux, du fait de la préservation de l'historique des solutions.

L'utilisation de cette méthode nous oblige donc, a priori, à n'appeler les procédures de fixations qu'une fois l'algorithme 1 vu à la section 2.3 terminé. Cependant, avec cette stratégie, le nombre de fixations réussies est relativement petit, car seule la dernière relaxation lagrangienne est étudiée. Nous avons décidé d'appeler les procédures de fixations de variables à chaque amélioration de la valeur `GLB`, c'est-à-dire à chaque fois que la borne $\texttt{LB}(u)$ est la meilleure trouvée, mais sans réellement effectuer ces fixations. En d'autres termes, nous gardons en mémoire toutes les fixations que nous aurions pu faire durant les différentes itérations, mais nous ne les appliquons pas immédiatement. Ce n'est qu'une fois l'algorithme terminé, que les dites fixations sont opérées. En procédant ainsi, les garanties que nous avions sur la qualité des multiplicateurs et sur celle de la dernière borne inférieure $\texttt{LB}(u)$ ne sont plus valables. En effet, ces garanties sont calculées sur les relaxations lagrangiennes définies sur l'espace initial, et non sur celui qui résulte des fixations. Si nous avons réussi à fixer des variables au cours du déroulement de l'algorithme, nous les effectuons, puis nous rappelons la méthode des faisceaux en utilisant, cette fois, le nouvel espace. Nous réitérons ce processus jusqu'à ce que plus aucune fixation ne soit possible. Le nombre de boucles nécessaires est souvent très petit, car, en pratique, seuls les premiers changement d'espace influent sur la valeur de la meilleure borne inférieure $\texttt{LB}(u)$.

La méthode des faisceaux est d'autant plus intéressante, que, si la convergence a été prouvée (i.e. que l'algorithme 1 de la section 2.3 s'est terminé au bout d'un nombre d'itérations inférieur à celui qui était autorisé), alors il existe une combinaison convexe des solutions $x^{\bar{u}^t}$ qui est une solution de la relaxation linéaire de $(\mathcal{P}_k)$. De plus, sous certaines conditions techniques que nous n'aborderons pas, la valeur des coefficients définissant cette combinaison convexe est fournie par la dernière procédure `faisceau` appelée.

Nous utiliserons ces renseignements pour initialiser l'ensemble des variables

initiales de l'algorithme *Branch-and-Cut*, dont le chapitre 5 est le sujet.

Enfin, notons l'utilité du théorème 2.2.2 pour la méthode des faisceaux. En effet, l'un des paramètres en entrée de la fonction `faisceau` est la précision absolue sur la violation des contraintes. Ce calcul nécessite une bonne connaissance du domaine de définition des multiplicateurs lagrangiens.

Comment et quand utiliser ces trois méthodes permettant de mettre à jours les coefficients lagrangiens ? Nous présentons dans la section suivante la stratégie que nous avons utilisée.

2.5 ALGORITHME UTILISÉ

Nous venons de décrire trois méthodes permettant de résoudre heuristiquement le problème dual lagrangien. Ces trois méthodes – les ajustements, le sous-gradient, et les faisceaux – ont des avantages et des inconvénients complémentaires :

ajustements

Avantage : rapide, dédié au problème étudié

Inconvénient : ne trouve qu'un optimum local

sous-gradient

Avantage : autorise les fixations

Inconvénient : convergence non garantie, peut être coûteux en temps de calcul si le nombre d'itérations autorisé est grand ou si la valeur minimale autorisé de π est très petite

faisceaux

Avantage : garantit la qualité de la borne inférieure, nombre d'itérations relativement réduit

Inconvénient : ne tient pas compte immédiatement des fixations

De ces observations, nous avons décidé de procéder en deux grandes phases : d'abord nous utilisons la méthode du sous-gradient comme une heuristique permettant d'obtenir un premier vecteur de multiplicateurs lagrangiens, vecteur qui est ensuite amélioré par la méthode des ajustements. Ensuite, nous entrons dans une phase basée sur la méthode des faisceaux. Contrairement à la méthode du sous-gradient, celle des faisceaux est très sensible à la qualité du vecteur de multiplicateurs initial $\bar{u}^0$. Le travail effectué par la phase du sous-gradient est donc

très intéressant, non seulement pour le nombre des fixations de variables qu'il autorise, mais aussi pour la rapidité des méthodes des faisceaux qui la suivent. Cependant, il est important de ne pas perdre trop de temps dans cette phase, car lorsque π est très petit, la portée des améliorations qu'il fournit est très limitée, alors que la phase des faisceaux qui la suit sera certainement plus efficace pour augmenter la qualité de la borne inférieure. Par conséquent, nous avons décidé d'utiliser une valeur minimale de π relativement grande (0.001), afin de l'arrêter prématurément.

L'algorithme que nous utilisons est décrit dans sa totalité par l'algorithme 2.

L'initialisation du premier vecteur de multiplicateurs $\bar{u}^0$ n'a que très peu d'importance, car la première méthode utilisée est celle du sous-gradient qui est très peu sensible à cette initialisation. Celle que nous avons choisie s'inspire du théorème 2.2.2 : pour toute combinaison i nous définissons $\bar{u}_i^0 = c_{j(i)}d_i$ avec $j(i)$ la combinaison la moins chère pouvant remplacer i.

2.6 CONCLUSION

Nous venons de décrire les phases principales de notre algorithme. Le but de cet algorithme lagrangien est triple : obtenir une borne inférieure la meilleure possible, fixer un grand nombre de variables, et construire une solution réalisable de $(\mathcal{P}_k)$ de très bonne qualité.

Nous avons exposé les différentes méthodes permettant d'atteindre le premier objectif. L'utilisation combinée du sous-gradient et des faisceaux s'avère très efficace, et n'avait, à notre connaissance, jamais été testée auparavant.

Le chapitre suivant est consacré au second objectif, à savoir la fixation des variables permettant de réduire la taille du problème.

Algorithme 2 Algorithme utilisé

Initialiser le vecteur des multiplicateurs $\bar{u}^0$
Initialiser GUB (pour les fixations)

Méthode du sous-gradient
$\Rightarrow$ mise à jour de GLB
$\Rightarrow$ mise à jour de GUB
$\Rightarrow$ fixation de variables

si la solution lagrangienne est réalisable pour $(\mathcal{P}_k)$ ou GLB=GUB **alors**
STOP, on a une solution optimale de $(\mathcal{P}_k)$
fin si

$\bar{u}^0 \leftarrow u$ tel que $\mathtt{LB}(u) = \mathtt{GLB}$

Méthode des ajustements
$\Rightarrow$ mise à jour de GLB
$\Rightarrow$ fixation de variables

si la solution lagrangienne est réalisable pour $(\mathcal{P}_k)$ ou GLB=GUB **alors**
STOP, on a une solution optimale de $(\mathcal{P}_k)$
fin si

répéter

$\bar{u}^0 \leftarrow u$ tel que $\mathtt{LB}(u) = \mathtt{GLB}$

Méthode des faisceaux
$\Rightarrow$ mise à jour de GLB
$\Rightarrow$ mise à jour de GUB
$\Rightarrow$ fixation de variables mémorisées mais non effectuées

Fixation des variables qui ont été mémorisées

si la solution lagrangienne est réalisable pour $(\mathcal{P}_k)$ ou GLB=GUB **alors**
STOP, on a une solution optimale de $(\mathcal{P}_k)$
fin si

$\bar{u}^0 \leftarrow u$ tel que $\mathtt{LB}(u) = \mathtt{GLB}$

Méthode des ajustements
$\Rightarrow$ mise à jour de GLB
$\Rightarrow$ fixation de variables

si la solution lagrangienne est réalisable pour $(\mathcal{P}_k)$ ou GLB=GUB **alors**
STOP, on a une solution optimale de $(\mathcal{P}_k)$
fin si

jusqu'à ce que convergence des faisceaux et plus aucune fixation

3

FIXATIONS DE VARIABLES

Sommaire

Dans ce chapitre, nous explorons les différentes voies qui nous permettent de fixer des variables du problème à 0 ou 1. Cette étude est particulièrement importante car elle a pour conséquence de réduire l'ensemble des solutions restant à considérer (par exemple, fixer une variable de sommet x_{jj} à 1 indique que désormais nous ne nous intéresserons qu'aux solutions où la combinaison j est produite).

Dans une première section, nous rappelons le théorème fondamental des fixations, ainsi que son utilisation dans la théorie de la programmation linéaire, et dans celle des relaxations lagrangiennes.

Dans une seconde section, nous définissons les notations et la terminologie que nous emploierons tout au long de ce chapitre.

A la troisième section, nous montrons les implications logiques qu'entraine toute fixation réussie à l'aide du théorème de la première section.

La section suivante est consacrée à l'exploitation efficace de ce théorème en utilisant les relaxations lagrangiennes.

Dans l'avant dernière section, nous expliquons pourquoi la méthode des fixations peut être l'une des meilleures façons d'introduire de nouvelles contraintes.

Nous concluons ce chapitre en détaillant une technique qui permet de réduire le problème sans fixer de variables.

3.1 THÉORÈME FONDAMENTAL DES FIXATIONS

Dans cette section, nous expliquons ce que nous entendrons par "fixer une variable", et nous détaillons un théorème très simple qui est la source des principaux critères de fixation que nous présentons dans ce chapitre.

Définition 3.1.1

Si nous avons la certitude qu'une variable x_{ij} vaut $\delta \in \{0,1\}$ dans toutes les solutions optimales que nous cherchons à obtenir, alors nous pouvons restreindre notre étude aux seules solutions réalisables du problème dans lesquelles cette variable vaut δ. Nous dirons alors que cette variable est fixée à δ.

Si une variable x_{ij} a été fixée à δ, nous dirons par abus de langage qu'une solution du problème est réalisable uniquement si elle possède cette variable à δ. Cela s'explique par le fait que fixer une variable x_{ij} à δ revient à ajouter la contrainte $x_{ij} = \delta$ dans tous les programmes que nous considérons.

Le théorème fondamental des fixations que nous allons énoncer repose sur l'idée très simple suivante : si nous réussissons à prouver que toutes les solutions réalisables du problème, dans lesquelles une variable x_{ij} vaut 1, ont un coût strictement supérieur au coût de la meilleure solution réalisable connue, alors nous pouvons fixer définitivement cette variable à 0. Et ceci est vrai aussi si nous échangeons les rôles de 0 et 1.

En pratique, montrer que le problème admet une solution optimale $\bar{x}$ avec $\bar{x}_{ij} = 1 - \delta$, $\delta \in \{0, 1\}$, n'est en général pas aisé. En revanche, il est parfois possible de montrer qu'il n'existe pas de solution optimale $\bar{x}$ avec $\bar{x}_{ij} = \delta$, et c'est ce principe que nous exploiterons dans la suite.

Théorème 3.1.2

Soit x_{ij} une variable binaire d'un problème de minimisation et $\delta \in \{0, 1\}$ une valeur possible de cette variable.
Soit $\mathtt{LB}_{ij}^{\delta}$ une borne inférieure du coût de toutes les solutions réalisables dans lesquelles la variable x_{ij} vaut δ.
Soit GUB *le coût d'une solution réalisable.*
Si $\mathtt{LB}_{ij}^{\delta} > \mathtt{GUB}$ alors la variable x_{ij} peut être fixée à $1 - \delta$

Preuve de 3.1.2 : $\mathtt{LB}_{ij}^{\delta}$ est une borne inférieure de toute solution réalisable possédant la variable x_{ij} à δ. Si l'on a $\mathtt{LB}_{ij}^{\delta} > \mathtt{GUB}$ alors nous avons la certitude qu'aucune solution optimale n'a la variable x_{ij} à δ. Cette variable étant binaire, nous en déduisons que pour toutes les solutions optimales du problème, cette variable est égale à $1-\delta$. Ainsi nous pouvons réduire l'étude du problème aux seules solutions ayant cette propriété. ♢

L'évaluation de la borne inférieure $\mathtt{LB}_{ij}^{\delta}$ est la clé de voûte d'une bonne utilisation de ce théorème. Dans la première section, cette borne inférieure est calculée à l'aide des relaxations linéaires, et dans la seconde, elle est calculée à l'aide des relaxations lagrangiennes.

3.1.1 APPLICATION AUX RELAXATIONS LINÉAIRES

Une première voie bien connue pour calculer la borne inférieure $\mathtt{LB}_{ij}^{\delta}$ est d'utiliser les résultats de la programmation linéaire.

La relaxation linéaire (LP) d'un programme linéaire en nombres binaires (P) est obtenue en relâchant les contraintes d'intégralité :

$$(P)\ \begin{cases} \text{minimiser} & cx & & \\ \text{sachant} & Ax & \leq b & \\ & x & \in \{0,1\}^n \end{cases} \quad \Rightarrow \quad (LP)\ \begin{cases} \text{minimiser} & cx & \\ \text{sachant} & Ax & \leq b \\ & x & \in [0,1]^n \end{cases}$$

Une solution $\bar{x}$ de base est une solution optimale dès que, pour chaque variable x_{ij} hors base, le coût réduit $\bar{c}_{ij} = c_{ij} - \bar{y}^t A^{ij}$ est positif ou nul si $\bar{x}_{ij} = 0$, et négatif ou nul si $\bar{x}_{ij} = 1$, avec A^{ij} la colonne de A associée à la variable x_{ij}, et $\bar{y}$ la solution optimale du problème dual, qui, rappelons le, est disponible dès que le programme linéaire a été résolu par l'algorithme du simplexe.

Le théorème suivant est un théorème classique de la programmation linéaire, sa démonstration s'appuie sur la théorie de la dualité et sur le fait que les variables considérées sont entières. Nous reprenons les notations du théorème 3.1.2 pour montrer qu'il ne s'agit que d'une application.

Théorème 3.1.3

Soit $\bar{x}$ une solution optimale de base, x_{ij} une variable hors base, et $\bar{c}_{ij}$ son coût réduit.

- *si $\bar{x}_{ij} = 0$, i.e. $\bar{c}_{ij} \geq 0$, alors $\mathtt{LB}^1_{ij} = c\bar{x} + \bar{c}_{ij}$ est une borne inférieure de toutes les solutions optimales de (P) dans lesquelles x_{ij} vaut 1.*
  Si $\mathtt{LB}^1_{ij} > \mathtt{GUB}$ alors nous pouvons fixer x_{ij} à 0.
- *si $\bar{x}_{ij} = 1$, i.e. $\bar{c}_{ij} \leq 0$, alors $\mathtt{LB}^1_{ij} = c\bar{x} - \bar{c}_{ij}$ est une borne inférieure de toutes les solutions optimales de (P) dans lesquelles x_{ij} vaut 0.*
  Si $\mathtt{LB}^0_{ij} > \mathtt{GUB}$ alors nous pouvons fixer x_{ij} à 1.

Ce résultat étant bien connu, nous ne le démontrerons pas ici.

Calculer les solutions optimales $\bar{x}$ et $\bar{y}$ est un problème facile en théorie, étant polynomial. Malheureusement, si les nombres de variables et de contraintes sont grands, il risque d'être en pratique très coûteux en temps, voire insoluble à cause de la limitation de la mémoire informatique.

Nous allons voir, à la section suivante, que l'application de ce théorème aux relaxations lagrangiennes nous offre beaucoup plus de souplesse.

3.1.2 APPLICATION AUX RELAXATIONS LAGRANGIENNES

Une autre façon d'obtenir une borne inférieure $\texttt{LB}_{ij}^{\delta}$ du coût de toutes les solutions réalisables possédant la variable x_{ij} à δ, consiste à ajouter à la relaxation lagrangienne courante la contrainte $x_{ij} = \delta$, voire aussi d'autres contraintes qui seraient satisfaites par toutes ces solutions réalisables.

En effet, le programme mathématique ainsi obtenu est une relaxation lagrangienne du programme initial (P) que l'on a complété par ces contraintes. La solution optimale de la nouvelle relaxation lagrangienne est donc une borne inférieure des solutions optimales du programme initial complété.

Plus nous ajoutons de contraintes, plus la résolution de la relaxation lagrangienne peut devenir complexe. Dans le cas du problème de la gestion de la diversité, la relaxation lagrangienne initiale est résolue en évaluant la quantité $S[.]$ associée à chaque combinaison, et en sélectionnant les k plus petites (voir chapitre 2). Fixer temporairement un sommet ou un arc à 0 ou 1, n'aura pour conséquence que la modification d'un certain nombre de valeurs $S[.]$. Ce nombre dépendant du nombre et du type de contraintes supplémentaires, il est important de bien gérer leur ajout. Ceci sera le sujet de la section 3.4.

3.2 NOTATIONS

Dans tout ce chapitre nous allons employer la terminologie de graphe que nous avons définie à la section 1.1.

Les ensembles V et A désignent respectivement l'ensemble des combinaisons et l'ensemble des substitutions possibles.

Ce chapitre étant consacré aux fixations, il devient nécessaire d'identifier clairement les combinaisons et les substitutions qui sont soit obligatoires (c'est-à-dire dont la variable est fixée à 1), soit interdites (c'est-à-dire fixées à 0), soit libres (= non fixées).

Par abus de langage, nous dirons qu'une combinaison ou une substitution est fixée à 0 ou 1 pour signifier que la variable associée est fixée à 0 ou 1.

Nous définissons V^1 et V^0 comme étant les ensembles des combinaisons respectivement fixées à 1 et à 0. Par conséquent, les combinaisons non fixées appartiennent à $V \setminus (V^1 \cup V^0)$. Parallèlement, nous définissons A^1 et A^0 les ensembles des arcs (= substitutions) fixés respectivement à 1 et à 0. Les arcs non fixés appartiennent donc à $A \setminus (A^1 \cup A^0)$.

3.3 FIXATIONS PAR IMPLICATION LOGIQUE

Fixer une variable à 0 ou 1 permet très souvent de fixer beaucoup d'autres variables. Par exemple, si nous réussissons à fixer une variable x_{jj} à 1, c'est-à-dire que nous ne nous intéressons désormais qu'aux solutions où j est produite, nous pouvons fixer définitivement toutes les variables $x_{j\ell}$ à 0 pour toutes les combinaisons $\ell \succ j$ car nous sommes certains que j ne sera pas substituée.

Les fixations par implication logique sont très nombreuses. Nous étudions à la section 3.3.2 celles qui sont dûes aux fixations des variables x_{jj}, et dans la section suivante, celles qui sont associées aux fixations des variables x_{ij}. À la section 3.3.4, nous exploitons une structure d'hypergraphe intrinsèquement liée au graphe partiel de $G = (V, A)$ engendré par les arcs non fixés.

Ces trois sections étant très dépendantes les unes des autres, par un effet que nous pourrions qualifier de "boule de neige", nous conseillons l'utilisation de la récursivité afin d'en tirer un profit maximum.

Mais avant d'énoncer ces différentes implications logiques, nous définissons le type de solution optimale que nous cherchons à obtenir.

3.3.1 PARTICULARITÉ DE CERTAINES SOLUTIONS OPTIMALES

D'après la définition du problème de la gestion de la diversité telle que nous l'avons donnée au chapitre 1, il peut arriver qu'il y ait plusieurs solutions optimales.

Ceci se produit notamment lorsque deux combinaisons produites possèdent non seulement le même coût, mais aussi peuvent remplacer la même combinaison. Cette dernière peut alors être substituée aussi bien par l'une que par l'autre sans altérer la qualité de la solution.

Pour lever ce type d'ambiguïté, nous définissons pour chaque combinaison i un *code décimal*, noté $code_i$, qui peut être par exemple le codage décimal des composantes du vecteur d'options qui lui est associé. Nous exigeons simplement que ce code décimal soit une fonction injective de l'ensemble des combinaisons V vers $I\!N$ afin de classer de façon univoque toutes les combinaisons.

Nous allons donc restreindre notre recherche au sous-ensemble OPT^{code} des solutions optimales telles que chaque combinaison i non produite est remplacée par la combinaison $j \succ i$ produite possédant non seulement un coût minimal, mais aussi, en cas d'ambiguïté, un code décimal minimal.

Il est clair que ce sous-ensemble n'est pas vide, car à partir de toute solution optimale nous pouvons en construire une autre possédant cette propriété.

Se restreindre à de telles solutions optimales va nous permettre d'énoncer des critères de fixation plus forts.

3.3.2 CONSÉQUENCES DES FIXATIONS DE SOMMETS

Dans cette section nous nous intéressons aux conséquences qu'entraine la fixation d'une variable x_{jj}. Dans une première partie, cette variable est supposée fixée à 1, et dans la seconde à 0.

3.3.2.1 FIXATION D'UN SOMMET À 1

Supposons qu'une variable x_{jj} soit fixée à 1. Cela signifie que dans toutes les solutions que nous étudions, la combinaison j est produite. Le théorème suivant regroupe toutes les conséquences qui en découlent.

Théorème 3.3.1

Si la variable x_{jj} est fixée à 1, alors

1. *Pour tout $\ell \succ j$, tel que $j\ell \notin A^0$ nous pouvons fixer $x_{j\ell}$ à 0, i.e. $A^0 \leftarrow A^0 + \{j\ell\}$*
2. *Pour tout $i \prec j$, pour tout $j' \succ i$ tel que $c_{j'} > c_j$ et $ij' \notin A^0$ nous pouvons fixer $x_{ij'}$ à 0, i.e. $A^0 \leftarrow A^0 + \{ij'\}$*

Preuve de 3.3.1 : Les premières fixations signifient que nous n'avons plus à considérer les substitutions de la combinaison j par une autre, vu qu'elle sera désormais produite.

Les secondes traduisent le fait qu'il ne serait pas optimale de remplacer une combinaison i que peut remplacer j par une autre j' plus chère. $\diamondsuit$

Le théorème suivant généralise simplement le second critère de fixation, en exploitant la structure des solutions optimales appartenant à OPT^{code}.

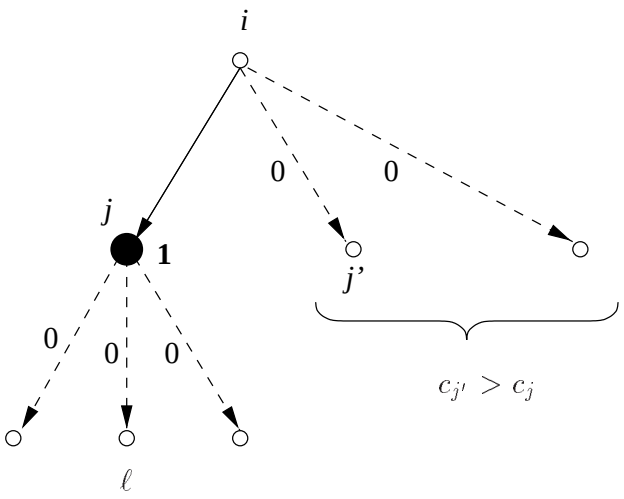


FIG. 3.1 – *Conséquences d'une fixation de sommet à 1*

Théorème 3.3.2

Si la variable x_{jj} est fixée à 1, alors pour tout $i \prec j$, pour tout $j' \succ i$ tel que
(a) $c_{j'} = c_j$
(b) $code_{j'} > code_j$
(c) $ij' \notin A^0$
nous pouvons fixer $x_{ij'}$ à 0, i.e. $A^0 \leftarrow A^0 + \{ij'\}$

Preuve de 3.3.2 : Une solution optimale dans laquelle la combinaison i serait remplacée par j' n'appartiendrait pas à l'ensemble des solutions optimales OPT^{code}, car nous savons d'après les hypothèses du théorème que j sera produite. Or, comme nous l'avons signalé à la section 3.3.1, nous ne nous intéressons qu'aux solutions optimales de OPT^{code}. ♢

3.3.2.2 FIXATION D'UN SOMMET À 0

Nous supposons dans cette section qu'une variable x_{jj} est fixée à 0, autrement dit, nous ne nous intéressons qu'aux solutions dans lesquelles la combinaison j n'est pas produite. Les conséquences d'une telle fixation sont regroupées dans le théorème suivant.

Théorème 3.3.3

Soit x_{jj} une variable fixée à 0. Les implications suivantes sont valides :

1. *Pour tout $i \prec j$, tel que $ij \notin A^0$ nous pouvons fixer x_{ij} à 0, i.e. $A^0 \leftarrow A^0 + \{ij\}$*
2. *S'il n'existe plus qu'une seule combinaison $\ell \succ j$ telle que $j\ell \notin A^0$, alors nous pouvons fixer $x_{j\ell}$ à 1, i.e. $A^1 \leftarrow A^1 + \{j\ell\}$*
3. *Soit $\ell \succ j$ telle que*
   (a) $j\ell \in A^0$
   (b) $j\ell' \in A^0$ pour tout $\ell' \succ j$ telle que $c_{\ell'} \leq c_\ell$

   Si $\ell \notin V^0$, alors nous pouvons fixer $x_{\ell\ell}$ à 0,
   i.e. $V^0 \leftarrow V^0 + \{\ell\}$

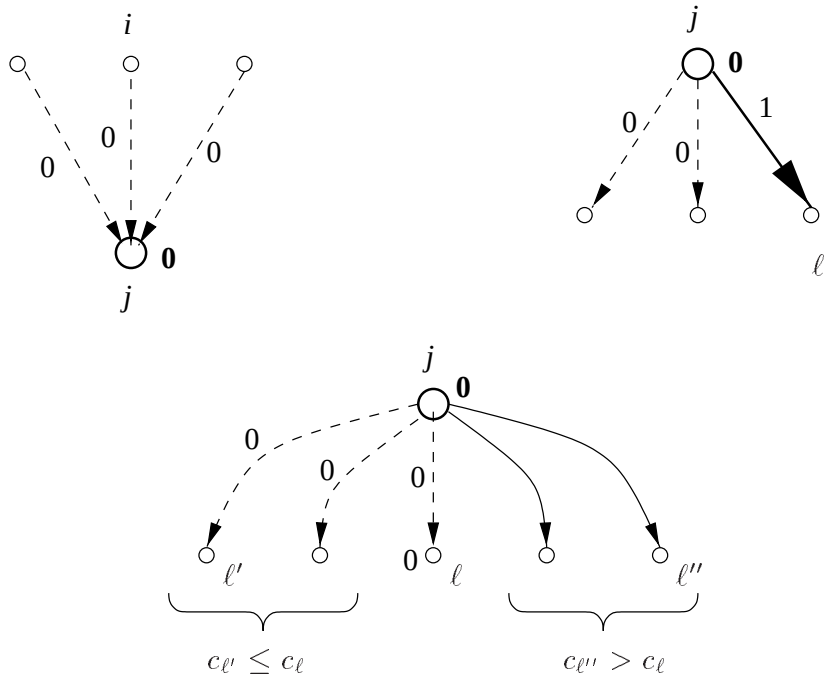


FIG. 3.2 – *Conséquences d'une fixation de sommet à 0*

Preuve de 3.3.3 : Les premières fixations traduisent le fait que la combinaison j n'étant pas produite, elle ne peut pas en remplacer une autre.

Sous les hypothèses de la seconde, nous savons que j n'est pas produite et qu'il ne reste plus qu'une seule combinaison candidate pour la remplacer. Il est donc clair que cette substitution devient alors obligatoire.

Pour prouver la validité du troisième critère, nous raisonnons par l'absurde. Supposons qu'il existe une solution optimale du problème dans laquelle une combinaison ℓ vérifiant les hypothèses du théorème est produite. Sachant qu'initialement j peut être substituée par ℓ, il vient que, dans cette solution optimale, la combinaison ℓ' qui remplace j possède un coût inférieur ou égal à celui de ℓ (sinon remplacer j par ℓ fournirait une solution réalisable de meilleure qualité que la solution courante, ce qui est contraire à l'optimalité de la solution). Or, par hypothèse, nous savons que les substitutions de j par toute combinaison ℓ' de coût inférieur ou égal à celui de ℓ sont interdites (les arcs $j\ell'$ appartenant à A^0). Nous en déduisons que j n'est ni produite ni remplacée, d'où la contradiction avec la réalisabilité de la solution. ♢

Comme à la section précédente, nous pouvons augmenter la puissance du dernier critère de fixation de ce théorème en exploitant la structure des solutions optimales de OPT^{code}.

Théorème 3.3.4

Soit x_{jj} une variable fixée à 0. Les implications suivantes sont valides :
3'. Soit $\ell \succ j$ telle que
(a) $j\ell \in A^0$
(b) $j\ell' \in A^0$ pour tout $\ell' \succ j$ telle que $c_{\ell'} < c_\ell$
(c) $j\ell' \in A^0$ pour tout $\ell' \succ j$ telle que $c_{\ell'} = c_\ell$ et $code_{\ell'} < code_\ell$
Si $\ell \notin V^0$, alors nous pouvons fixer $x_{\ell\ell}$ à 0,
i.e. $V^0 \leftarrow V^0 + \{\ell\}$

Preuve de 3.3.4 : Pour démontrer ce théorème nous reprenons la preuve précédente. Supposons qu'il existe une solution optimale du problème appartenant à OPT^{code} dans laquelle une combinaison ℓ vérifiant les hypothèses du théorème est produite. Sachant qu'initialement j peut être substituée par ℓ, il vient que, dans cette solution optimale, la combinaison ℓ' qui remplace j possède un coût qui est soit strictement inférieur à c_ℓ, soit égal à c_ℓ mais alors le code décimal de cette combinaison ℓ' doit être strictement inférieur à celui de ℓ, par définition de OPT^{code}. Or, par hypothèse, nous savons que toutes ces substitutions sont interdites. Nous en déduisons que j n'est ni produite ni remplacée, d'où la contradiction avec la réalisabilité de la solution. ♢

Ces derniers critères sont très proches d'un critère que nous décrirons à la section 3.3.3.2. Il n'a d'utilité que si la fixation de x_{jj} à 0 intervient après la fixation des substitutions de j par les combinaisons ℓ les moins chères, et si ces combinaisons ℓ ne sont pas déjà fixées.

3.3.3 CONSÉQUENCES DES FIXATIONS D'ARCS

Cette section est dédiée à l'étude des conséquences qu'entraîne la fixation d'une variable d'arc à 0 ou 1.

Fixer un arc ij à 0 ou 1 revient à décider que désormais la combinaison i ne sera jamais ou sera toujours remplacée par la combinaison j.

Nous décrivons dans les deux sections suivantes les principales implications logiques qu'entraine la fixation d'un arc. Cependant, nous verrons à la section 3.5, qu'il en existe d'autres. Ces dernières sont isolées dans une section particulière, étant un exemple de traduction de contraintes en critères de fixation.

3.3.3.1 FIXATION D'UN ARC À 1

Nous supposons ici que la variable x_{ij} est fixée à 1. Ceci signifie que toutes les solutions que nous allons considérer dans la suite sont telles que i sera toujours remplacée par j. Cette fixation est certainement la fixation qui a le plus de conséquences directes et indirectes parmi toutes celles décrites dans ce chapitre.

Théorème 3.3.5

Soit x_{ij} une variable fixée à 1, avec $i \prec j$.

1. *Si $j \notin V^1$, alors nous pouvons fixer x_{jj} à 1, i.e. $V^1 \leftarrow V^1 + \{j\}$*
2. *Si $i \notin V^0$, alors nous pouvons fixer x_{ii} à 0, i.e. $V^0 \leftarrow V^0 + \{i\}$*
3. *Pour tout $j' \succ i$, $j' \neq j$,*
   (a) *Si $ij' \notin A^0$, alors nous pouvons fixer $x_{ij'}$ à 0, i.e. $A^0 \leftarrow A^0 + \{ij'\}$*
   (b) *Si $c_{j'} < c_j$ et $j' \notin V^0$, alors nous pouvons fixer $x_{j'j'}$ à 0, i.e. $V^0 \leftarrow V^0 + \{j'\}$*

Preuve de 3.3.5 : Les deux premières fixations traduisent le fait que remplacer la combinaison i par j présuppose que la combinaison j est produite, et que la combinaison i ne l'est pas.

La validité de la fixation suivante provient du fait que si nous savons que i est remplacée par j, alors nous savons que i ne le sera pas par une autre combinaison.

Pour démontrer que la dernière fixation est légale, il suffit de raisonner par l'absurde : supposons qu'il existe une solution optimale du problème dans laquelle une combinaison produite j' est strictement moins chère que j et peut, elle aussi, remplacer i. Il est clair que dans une telle solution, i ne sera pas remplacée par j, dans le pire des cas elle sera remplacé par j'. D'où la contradiction avec le fait

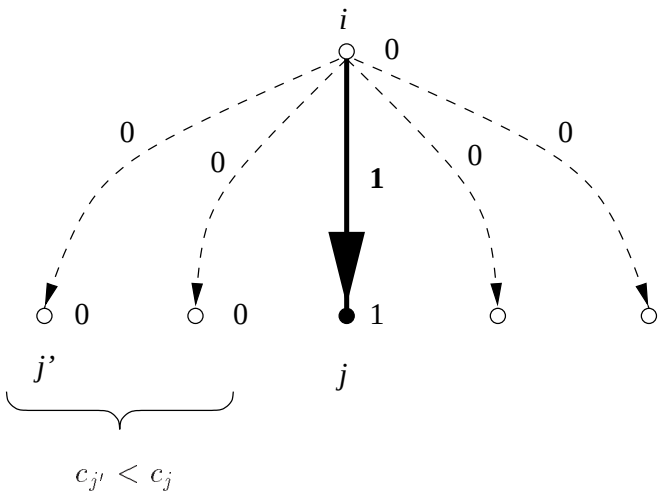


FIG. 3.3 – *Conséquences d'une fixation d'arc à 1*

que la substitution de i par j est obligatoire. ♢

Si nous tenons compte du fait que seules les solutions optimales de OPT^{code} nous intéressent, nous obtenons un renforcement qui s'énonce comme suit :

Théorème 3.3.6

Soit x_{ij} une variable fixée à 1, avec $i \prec j$.

3. Pour tout $j' \succ i$, $j' \neq j$,

(c) Si $c_{j'} = c_j$ et $code_{j'} < code_j$ et $j' \notin V^0$, alors nous pouvons fixer $x_{j'j'}$ à 0, i.e. $V^0 \leftarrow V^0 + \{j'\}$

Preuve de 3.3.6 : Si nous produisions une telle combinaison j', cela signifierait qu'une solution optimale dans laquelle i est remplacée par j ne pourrait pas appartenir à OPT^{code}, par définition de cet ensemble. Or, nous savons que la variable x_{ij} est fixée à 1, cela signifie que nous ne nous intéressons désormais qu'aux solutions où i est justement remplacée par j. ♢

3.3.3.2 FIXATION D'UN ARC À 0

Fixer un arc ij à 0 n'a quasiment aucune conséquence, car cela revient à dire que nous sommes certains que la combinaison i ne sera pas remplacée par j, mais cela n'implique ni la production, ni la non production de i ou de j, et encore moins la désignation de la combinaison qui remplacera i le cas échéant. Le théorème

suivant décrit l'une des rares situations où des implications logiques sont quand même possibles.

Théorème 3.3.7

Soit x_{ij} une variable fixée à 0, avec $i \prec j$, telle que $i \in V^0$

1. *s'il existe $j' \succ i$ telle que*
   (a) *ij' n'est pas fixé*
   (b) *$ij'' \in A^0$ pour tout $j'' \succ i$, $j'' \neq j'$*

   Alors nous pouvons fixer $x_{ij'}$ à 1, i.e. $A^1 \leftarrow A^1 + \{ij'\}$
2. *si $j \notin V^0$ et si $ij' \in A^0$ pour tout $j' \succ i$ tels que $c_{j'} \leq c_j$, alors nous pouvons fixer x_{jj} à 0, i.e. $V^0 \leftarrow V^0 + \{j\}$*

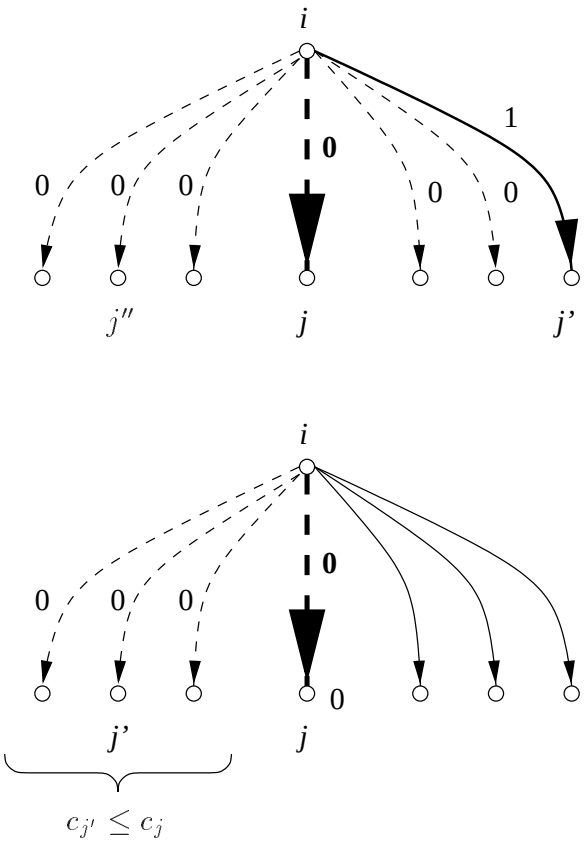


FIG. 3.4 – *Conséquences d'une fixation d'arc à 0*

Preuve de 3.3.7 : Pour prouver ce théorème il suffit de reprendre les raisonnements développés pour les deux derniers critères de fixation du théorème 3.3.3. ♢

Le théorème suivant renforce le dernier critère en utilisant la particularité des solutions optimales de OPT^{code}.

Théorème 3.3.8

Soit x_{ij} une variable fixée à 0, avec $i \prec j$, telle que $i \in V^0$

3. *si $j \notin V^0$ et si $ij' \in A^0$ pour tout $j' \succ i$ tels que $c_{j'} < c_j$ ou tels que $c_{j'} = c_j$ et $code_{j'} < code_j$, alors nous pouvons fixer x_{jj} à 0, i.e. $V^0 \leftarrow V^0 + \{j\}$*

Preuve de 3.3.8 : analogue à celle de 3.3.4. ♢

Comme nous l'avons déjà signalé, les situations décrites dans ce théorème sont peu fréquentes. En fait nous pouvons être raisonnablement moins pessimistes quant au nombre d'implications logiques entraînées par une fixation d'arc à 0. La section suivante en explique la raison.

3.3.4 FIXATIONS PAR HYPERGRAPHE

Lorsque plusieurs fixations de sommets ou d'arcs ont été effectuées, nous pouvons construire ce que nous appellerons *l'hypergraphe des sommets non fixés.*

La notion d'hypergraphes est une généralisation du concept de graphes. Pour plus de détails, le lecteur est invité à se reporter à l'ouvrage de C. Berge ([Ber70]). Comme pour un graphe, un hypergraphe $\mathcal{H}$ est défini par un couple $(\mathcal{V}, \mathcal{E})$, où $\mathcal{V}$ est un ensemble sommets, et où $\mathcal{E}$ est un ensemble de sous-ensembles de $\mathcal{V}$ appelés *arêtes* de $\mathcal{H}$. Contrairement à un graphe traditionnel, les cardinalités des arêtes d'un hypergraphe sont quelconques.

Les sommets de l'hypergraphe $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ que nous allons utiliser sont les combinaisons qui ne sont pas encore fixées. Pour chaque combinaison $i \in V$ non fixée à 1, nous étudions l'ensemble E_i des combinaisons qui sont candidates pour la remplacer (par abus de langage, cet ensemble contient i si celle-ci n'est pas encore fixée à 0). Si E_i ne contient aucune combinaison fixée à 1, et n'est contenu dans (et n'est égale à) aucun autre, alors il définit une arête de l'hypergraphe $\mathcal{H}$.

Cet hypergraphe $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ est donc caractérisé par :

– $\mathcal{V} = V \setminus (V^1 \cup V^0)$

– $\mathcal{E} = \{\ E_i \ \mid \ E_i \cap V^1 = \emptyset,\ \ i \in V \ \text{ et } E_i \not\subset E_j\ \forall j \in V\}$

avec $E_i = \{\ j \succ i \ \mid\ ij \notin A^0\ \}$ si $i \in V^0$
et $E_i = \{\ j \succ i \ \mid\ ij \notin A^0\ \} \cup \{i\}$ si $i \notin V^0$

Par mesure d'efficacité, nous ne construisons cet hypergraphe que lorsque que toutes les fixations décrites aux sections précédentes ont été exploitées, et ont donné lieu à un nombre substantiel de fixations. Ceci a pour conséquence que chaque arête $E_i \in \mathcal{E}$ est de cardinalité au minimum 2. En effet, si nous avions $E_i = \{j\}$ alors nous aurions pu fixer la variable x_{ij} à 1, que i soit différent de j ou non.

L'intérêt de cette structure d'hypergraphe est que, dans toute solution optimale de OPT^{code}, et dans toutes les arêtes $E_i \in \mathcal{E}$, nous avons la certitude qu'il existe au moins une combinaison produite qui n'est pas encore fixée.

Nous allons exploiter dans cette section la notion d'hypergraphe pour élaborer d'autres critères de fixation par implication logique. Nous utiliserons à nouveau ce concept à la section 3.7 pour *identifier* certains sommets et certains arcs, puis à la section 5.5 pour générer des règles de branchement dans un algorithme de Branch&Cut, et enfin à la section 5.3.4.3 pour générer efficacement certaines contraintes.

Nous énonçons le théorème suivant dans une formulation où nous privilégions directement les solutions optimales de OPT^{code}. Ce théorème est appliqué lors de la construction de $\mathcal{H}$, au moment où nous détectons qu'une arête est incluse dans une autre.

Théorème 3.3.9

Soient deux combinaisons i et j telles que $E_i \subset E_j$, et $\ell \in E_j \setminus E_i$. Si l'une des propriétés suivantes est vrai :

1. *$c_\ell \geq c_{\ell'}$ pour tout $\ell' \in E_i$, et $code_\ell > code_{\ell'}$ pour tout $\ell' \in E_i$ tels que $c_\ell = c_{\ell'}$*
2. *$i \prec \ell$*

Alors nous pouvons fixer la variable $x_{j\ell}$ à 0, i.e. $A^0 \leftarrow A^0 + \{j\ell\}$

La figure 3.5 illustre ces nouveaux critères dans le cas où les combinaisons i et j sont fixées à 0.

Ce théorème, tel que nous venons de l'énoncer, présuppose que $\ell \neq j$. Cela s'explique par les deux observations suivantes :

cas 1 : les hypothèses impliquent immédiatement que $\ell \neq j$, car pour tout $l' \succ j$, et notamment pour tout $l' \in E_i$, nous avons $c_j < c_{l'}$

cas 2 : si $j \in E_j \setminus E_i$ et $i \prec j$, cela signifie que non seulement la combinaison j n'est pas encore fixée, contrairement à l'arc ij qui est fixé à 0, mais aussi que i est fixée à 0 (sinon nous aurions simultanément $i \in E_i \subset E_j$, ce

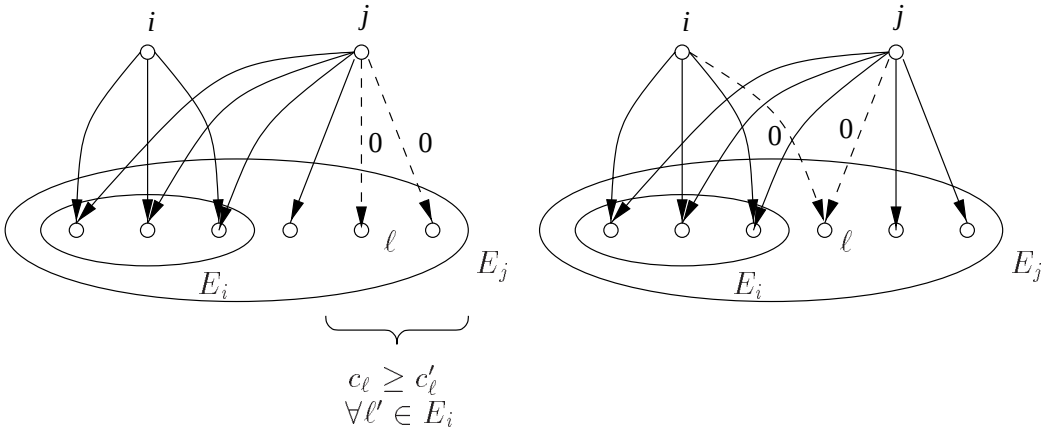


FIG. 3.5 – *Fixation d'arc par hypergraphe*

qui implique $i \succ j$, et $i \prec j$) . Sachant que $c_j < c_{\ell'}$ pour tout $\ell' \succ j$, et notamment pour tout $\ell' \in E_i$, nous aurions pu déjà fixer j à 0 grâce aux critères des théorèmes 3.3.3 et 3.3.7.

Preuve de 3.3.9 : La preuve de ce théorème s'appuie essentiellement sur le fait que dans chaque arête de l'hypergraphe nous devons produire au moins une combinaison. L'arête E_i étant incluse dans E_j, nous avons donc la certitude qu'une combinaison de E_i pourra remplacer j le cas échéant, et donc qu'a priori, certaines substitutions de j par des éléments de $E_j \setminus E_i$ sont inutiles.

Soit $\ell \in E_j \setminus E_i$, $\ell \neq j$,

1. Si $c_\ell \geq c_{\ell'}$ pour tout $\ell' \in E_i$, deux cas se présentent :
   - soit $c_\ell > c_{\ell'}$ pour tout $\ell' \in E_i$, et alors remplacer j par ℓ ne serait pas optimale vu que toute combinaison $\ell' \in E_i$ est meilleure candidate et qu'au moins l'une d'entre elles sera produite.
   - soit il existe des combinaisons $\ell' \in E_i$ aussi chères que ℓ. Si les codes décimaux de ces combinaisons sont tous strictement inférieurs à celui de ℓ, alors ou bien aucune d'elles n'est produite, et nous nous retrouvons dans la situation précédente, ou bien l'une d'elle est produite et une solution optimale où j serait remplacée par ℓ ne pourrait pas appartenir à OPT^{code}.
2. Si $i \prec \ell$, alors $i\ell \in A^0$, car $\ell \notin E_i$. Supposons qu'il existe une solution optimale de OPT^{code} dans laquelle j est remplacée par ℓ, ce qui implique que ℓ est produite. Soit $\ell' \in E_i$ la combinaison qui "remplace" i dans cette solution (ℓ' peut être égal à i si celle-ci est produite). Si $c_\ell > c_{\ell'}$ alors la

solution n'est pas optimale car la substitution de j par ℓ' aurait été plus profitable. Même raisonnement si $c_\ell < c_{\ell'}$. Il vient que $c_\ell = c_{\ell'}$. Si nous refaisons ce raisonnement non plus sur les coûts, mais sur les codes décimaux, nous en déduisons que l'appartenance de la solution optimale à l'ensemble OPT^{code} implique $code_\ell = code_{\ell'}$, et donc, par injectivité du code décimal, que $\ell = \ell'$, ce qui contredit le fait que $\ell \notin E_i$ et $\ell' \in E_i$

♢

Jusqu'à présent, toutes les implications logiques que nous avons développées ne sont que les conséquences plus ou moins directes de fixations réussies.

Le critère que nous présentons ici à ceci de particulier qu'il est fondé sur un raisonnement par l'absurde. De part cette caractéristique, il se situe à mi-chemin entre les fixations par implications logiques traditionnelles et les fixations par coût réduit dont nous verrons quelques exemples à la section 3.4.

Nous définissons, pour chaque combinaison j non fixée, un hypergraphe $\mathcal{H}_j^\delta$. Cet hypergraphe est l'hypergraphe $\mathcal{H}$ que nous aurions obtenu si j avait été fixée à δ.

Nous rappelons ce qu'on entend par *transversal* d'un hypergraphe.

Définition 3.3.10

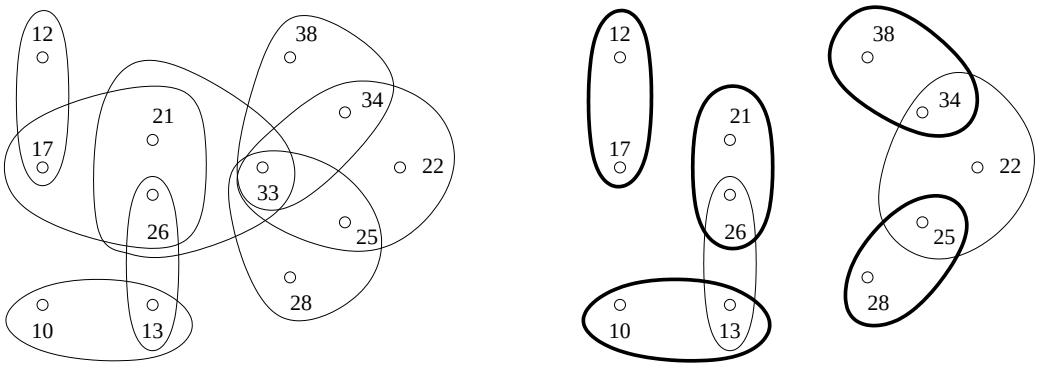
Soit $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ un hypergraphe et $\mathcal{T} \subseteq \mathcal{V}$.
$\mathcal{T}$ est un transversal de $\mathcal{H}$ si chaque arête de $\mathcal{E}$ contient au moins un sommet de $\mathcal{T}$.
Nous noterons $\tau(\mathcal{H})$ la cardinalité minimum des transversaux de $\mathcal{H}$

Théorème 3.3.11

Soit $j \notin V^0 \cup V^1$ et $\delta \in \{0, 1\}$
Si $\tau(\mathcal{H}_j^\delta) > k - |V^1|$ alors nous pouvons fixer x_{jj} à $1 - \delta$, i.e. $V^{1-\delta} \leftarrow V^{1-\delta} + \{j\}$

Preuve de 3.3.11 : trivial, d'après la définition des transversaux. ♢

Trouver, pour un hypergraphe donné, la valeur de τ est un problème NP-difficile (voir [GJ79]). Nous avons donc choisi d'exhiber un nombre α d'arêtes 2 à 2 disjointes de l'hypergraphe $\mathcal{H}_j^\delta$. Sachant que $\alpha \leq \tau(\mathcal{H}_j^\delta)$ (car tous les transversaux possèdent au moins un élément dans chaque arête), si nous prouvons que $\alpha > k - |V^1|$, alors nous pouvons fixer x_{jj} à $1 - \delta$.
Exemple : La figure 3.6 illustre un exemple où ce type de fixation peut réussir.



Hypergraphe $\mathcal{H}$ *Hypergraphe $\mathcal{H}^0_{33}$*

FIG. 3.6 – *Exemple de fixation par stabilité d'hypergraphe*

Il s'agit d'une instance réelle comportant 42 combinaisons demandées, et pour laquelle le nombre de combinaisons à produire est $k = 10$. Nous avons déjà fixé $|V^1| = 6$ combinaisons à 1, il n'en reste donc plus que 4 à choisir. Le dessin de gauche représente l'hypergraphe $\mathcal{H}$, et celui de droite l'hypergraphe $\mathcal{H}^0_{33}$ construit en supposant que la combinaison 33 n'est pas produite. Comme le montre cette figure, il existe alors $\alpha = 5$ arêtes qui ont une intersection vide deux à deux, ce qui signifie qu'il est nécessaire de produire au minimum 5 combinaisons, en plus des 6 déjà fixées à 1. Nous en déduisons que ne pas produire cette combinaison 33 est impossible, et donc qu'elle peut être fixée à 1.

Remarque : nous aurions pu énoncer un théorème analogue en considérant non plus les variables de sommets mais les variables d'arcs, cependant le nombre d'arcs étant très élevé et les situations pour lesquelles ce théorème est applicable très rares, ce nouveau théorème n'aurait eu qu'une valeur théorique.

Le problème qui consiste à calculer la valeur maximale de α est lui-même NP-difficile (voir [GJ79]), nous avons donc choisi d'évaluer ce nombre à l'aide d'une heuristique gloutonne rapide (voir algorithme 3).

Pour la rendre encore plus rapide, nous pouvons évaluer dynamiquement le degré d'intersection $D_{\mathcal{H}^\delta_j}(E_i)$, qui correspond aux nombres d'arêtes intersectées par E_i. De plus, si plusieurs arêtes minimisent cette quantité $D_{\mathcal{H}^\delta_j}(.)$, il est préférable de choisir celle pour laquelle cette valeur était la plus petite dans l'hypergraphe initial, et si le choix est toujours multiple, de prendre celle dont la cardinalité est minimale. Ces règles permettent en effet de limiter le nombre d'arêtes que l'on va éliminer.

Algorithme 3 Heuristique d'évaluation de α

$\alpha \leftarrow 0$
tant que il reste des arêtes dans l'hypergraphe $\mathcal{H}_j^\delta$ **faire**
  pour tout arête E_i de l'hypergraphe $\mathcal{H}_j^\delta$ **faire**
    calculer le nombre $D_{\mathcal{H}_j^\delta}(E_i)$ d'arêtes E_j de $\mathcal{H}_j^\delta$ telles que $E_i \cap E_j \neq \emptyset$
  fin pour
  Choisir une arête E_i possédant le plus petit $D_{\mathcal{H}_j^\delta}(.)$
  Retirer l'arête E_i et toutes les arêtes intersectant E_i
  Incrémenter α
fin tant que

En évaluant $\tau(\mathcal{H}_j^\delta)$ à l'aide de α, nous ne tenons malheureusement pas compte du fait que tout transversal possède au minimum $n+1$ sommets dans un cycle composé de $2n+1$ arêtes. La figure 3.7 nous montre un tel exemple. Le schéma de gauche correspond à l'hypergraphe $\mathcal{H}_j^\delta$, et celui du centre représente la solution calculée par l'algorithme 3. Nous voyons que $\alpha = 2$. Celui de droite montre un transversal de cardinalité minimum, ici $\tau(\mathcal{H}_j^\delta) = 3$. Si $k - |V^1| = 2$ nous aurions pu fixer la variable x_{jj} à $1-\delta$ grâce à $\tau(\mathcal{H}_j^\delta)$, mais α ne nous le permet pas.

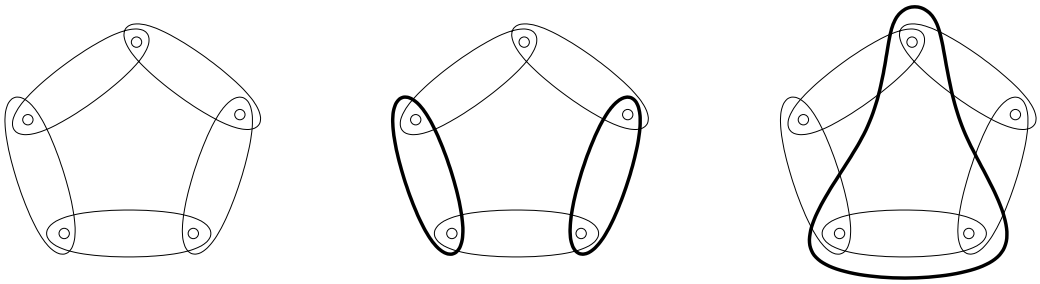


FIG. 3.7 – *Limite de l'algorithme 3*

Nous avons choisi de garder cette heuristique malgré tout, car celle-ci est appelé fréquemment et les situations où le théorème 3.3.11 permet de fixer des variables sont rares. Ralentir cette heuristique, et donc a fortiori tout l'algorithme, en exploitant les cycles impairs, nous ferait perdre plus de temps qu'il n'en nous ferait gagner.

3.4 FIXATIONS PAR COÛT RÉDUIT LAGRANGIEN

Quelques fixations par implication logique peuvent être effectuées avant même de pénétrer dans les boucles de l'algorithme lagrangien, mais elles sont très limitées. Il s'agit principalement de la fixation à 1 des combinaisons terminales dont la demande est strictement positive, car elles ne peuvent être remplacées par aucune autre.

Les principales sources de fixations sont celles qui sont issues du théorème 3.1.2, et que nous appellerons *fixations par coût réduit*. Nous avons vu à la section 3.1 les idées générales qui nous permettent d'exploiter ce théorème. Nous allons approfondir dans cette nouvelle section son application aux relaxations lagrangiennes.

Pour évaluer les différentes bornes inférieures $\texttt{LB}_{ij}^{\delta}$, nous allons ajouter aux relaxations lagrangiennes quelques contraintes provenant des implications logiques décrites précédemment.

Plus les contraintes supplémentaires sont nombreuses, meilleure sera l'évaluation de $\texttt{LB}_{ij}^{\delta}$, et plus grandes seront les chances de fixer la variable x_{ij} à $1 - \delta$. En effet, en ajoutant des contraintes, nous diminuons l'espace des solutions étudiées. Par conséquent, la valeur de la solution optimale du nouveau programme ne peut être que supérieure ou égale à la valeur de la relaxation lagrangienne courante. Cependant, ajouter simultanément un grand nombre de contraintes peu ralentir considérablement le calcul de la nouvelle relaxation lagrangienne, et donc l'évaluation de $\texttt{LB}_{ij}^{\delta}$. Nous allons décrire dans les sections suivantes différentes fixations efficaces par coût réduit.

Dans la première section, consacrée à ce nous appellerons les fixations *simples*, nous ajoutons au programme simplement la contrainte $x_{ij} = \delta$ sans tenir compte de toutes les conséquences logiques qu'entraine le fait que x_{ij} est à δ. Dans la seconde, en revanche, nous ajoutons non seulement $x_{ij} = \delta$ mais aussi quelques autres contraintes traduisant quelques implications logiques. Nous parlerons alors de fixations *avancées*. Nous verrons comment il est possible de calculer par étape une solution optimale de la nouvelle relaxation lagrangienne. Mais là encore, nous ne prendrons en compte qu'une infime partie des implications logiques que nous pourrions déduire du fait que x_{ij} est à δ. Contrairement aux deux précédentes, nous considérons à la troisième section, absolument toutes les conséquences logiques. De telles fixations seront alors dites *fortes*.

Pour ne pas alourdir les démonstrations qui vont suivre, nous noterons $\texttt{LB}$, au lieu de $\texttt{LB}(\bar{u}^t)$, la valeur à l'optimum de la relaxation lagrangienne courante. Plus généralement, nous n'indiquerons plus le t de l'itération courante de l'algorithme

lagrangien.

3.4.1 FIXATIONS SIMPLES

La plus élémentaire des évaluations de $\texttt{LB}_{ij}^{\delta}$ consiste à n'ajouter à la relaxation lagrangienne courante que la contrainte $x_{ij} = \delta$, en ne prenant en compte aucune des conséquences logiques que nous pourrions en déduire. Dans cette section nous supposerons toujours que la variable x_{ij} vaut $1 - \delta$ dans la solution optimale $\bar{x}$ de la relaxation lagrangienne courante. Ajouter la contrainte $x_{ij} = \delta$ dans le cas contraire serait inutile, car la solution $\bar{x}$ de valeur LB, serait encore optimale pour le nouveau programme, et par conséquent la valeur $\texttt{LB}_{ij}^{\delta}$ serait égale à LB que l'on sait être une borne inférieure de GUB.

Les deux théorèmes ci-dessous nous montrent comment exploiter la valeur LB de la solution $\bar{x}$ pour calculer $\texttt{LB}_{ij}^{\delta}$. Le premier est consacré aux variables de sommets, et le second aux variables d'arcs.

En plus de R qui désigne l'ensemble des combinaisons produites, nous définissons deux nouvelles notations :

- $Smax1 = \max\{\ S[j]\ \mid\ j \in R\ \text{ et }\ j \notin V^1\}$
- $Smin0 = \min\{\ S[j]\ \mid\ j \notin R\ \text{ et }\ j \notin V^0\}$

Comme nous l'avons vu au chapitre 2, la solution optimale $\bar{x}$ de la relaxation lagrangienne courante est construite en choisissant les k combinaisons qui, ou bien sont fixées à 1, ou bien possèdent les plus petits $S[.]$.

Ainsi la valeur $Smax1$ désigne la valeur $S[.]$ de la pire combinaison produite non fixée, et $Smin0$ désigne la valeur $S[.]$ de la meilleure combinaison non fixée qui n'est pas produite.

De plus, nous rappelons que $\bar{x}_{ij}$ vaut 1, avec $j \neq i$ et $ij \notin A^0$, si et seulement si soit $j \in R$ et $\bar{\gamma}_{ij} = c_j d_i - \bar{u}_i < 0$, soit $ij \in A^1$, sinon $\bar{x}_{ij}$ vaut 0.

Théorème 3.4.1

Soit x_{jj} une variable non fixée.

1. *Si $\bar{x}_{jj} = 1$, et si $\texttt{LB}_{jj}^{0} = \texttt{LB} - S[j] + Smin0 > \texttt{GUB}$ alors nous pouvons fixer la variable x_{jj} à 1, i.e. $V^1 \leftarrow V^1 + \{j\}$*
2. *Si $\bar{x}_{jj} = 0$, et si $\texttt{LB}_{jj}^{1} = \texttt{LB} + S[j] - Smax1 > \texttt{GUB}$ alors nous pouvons fixer la variable x_{jj} à 0, i.e. $V^0 \leftarrow V^0 + \{j\}$*

Théorème 3.4.2

Soit x_{ij} une variable non fixée, $j \succ i$.

1. *Si $\bar{x}_{ij} = 1$, et $j \in V^1$, et si $\mathtt{LB}^0_{ij} = \mathtt{LB} - \bar{\gamma}_{ij} > \mathtt{GUB}$ alors nous pouvons fixer la variable x_{ij} à 1, i.e. $A^1 \leftarrow A^1 + \{ij\}$*
2. *Si $\bar{x}_{ij} = 0$, et $\bar{x}_{jj} = 1$, et si $\mathtt{LB}^1_{ij} = \mathtt{LB} + \bar{\gamma}_{ij} > \mathtt{GUB}$ alors nous pouvons fixer la variable x_{ij} à 0, i.e. $A^0 \leftarrow A^0 + \{ij\}$*

Preuve de 3.4.1 : Soit x_{jj} une variable non fixée.

1. Nous supposons que $\bar{x}_{jj} = 1$, i.e. i est produite dans la solution lagrangienne courante. Si nous exigeons que j ne soit pas produite, ce qui revient à ajouter la contrainte $x_{jj} = 0$, alors une solution optimale de la nouvelle relaxation lagrangienne consiste à produire toutes les combinaisons de R, excepté j, et la combinaison j' qui nous a permis de définir $Smin0$, étant la meilleure combinaison non produite jusqu'à présent. La démonstration de l'optimalité de cette solution est triviale, et son coût est $\mathtt{LB}^0_{jj} = \mathtt{LB} - S[j] + Smin0$. Cette valeur $\mathtt{LB}^0_{jj}$ est une borne inférieure du coût de toutes les solutions optimales dans lesquelles j n'est pas produite. D'après le théorème 3.1.2, si $\mathtt{LB}^0_{jj} > \mathtt{GUB}$, alors nous pouvons fixer à 1 la variable x_{jj}.
2. Supposons $\bar{x}_{jj} = 0$, i.e. i n'est pas produite dans la solution lagrangienne courante. Nous ajoutons la contrainte $x_{jj} = 1$. Une solution optimale de ce nouveau programme est obtenue en sélectionnant les combinaisons de R, excepté celle qui a permis de définir $Smax1$ afin de libérer un place pour la combinaison j. La valeur de cette solution $\mathtt{LB}^1_{jj}$, est égale à $\mathtt{LB} + S[j] - Smax1$. Donc, d'après le théorème 3.1.2, si $\mathtt{LB}^1_{jj} > \mathtt{GUB}$, alors nous pouvons fixer à 0 la variable x_{jj}.

♢

Preuve de 3.4.2 : Soit x_{ij} une variable non fixée, $j \succ i$.

1. Supposons $\bar{x}_{ij} = 1$, avec j fixée à 1. Noter que $\bar{x}_{ij} = 1$ implique que $\bar{\gamma}_{ij} < 0$, d'après la définition de $\bar{x}$. Nous ajoutons la contrainte $x_{ij} = 0$ à la relaxation lagrangienne courante. Le coût réduit $\bar{\gamma}_{ij}$ n'intervenant plus dans le calcul de $S[j]$, celui-ci devient $S[j] - \bar{\gamma}_{ij}$. Mais, quelle que soit la nouvelle valeur de $S[j]$, nous avons l'assurance que la combinaison j sera toujours produite, étant fixée à 1. La solution $\bar{x}$ reste donc optimale pour le nouveau programme, après avoir posé $\bar{x}_{ij} = 0$. Son coût $\mathtt{LB}^0_{ij}$, égal à $\mathtt{LB} - \bar{\gamma}_{ij}$, est une borne inférieure du coût de toutes les solutions optimales du programme initial dans lesquelles x_{ij} vaut 0. D'après le théorème 3.1.2, si $\mathtt{LB}^0_{ij} > \mathtt{GUB}$, alors nous pouvons fixer à 1 la variable x_{ij}.

2. Nous supposons ici que $\bar{x}_{ij} = 0$ et que $\bar{x}_{jj} = 1$. Cela implique, par définition de $\bar{x}$, que $\bar{\gamma}_{ij} \geq 0$. Nous ajoutons la contrainte $x_{ij} = 1$ à la relaxation lagrangienne courante, c'est-à-dire que nous exigeons que i soit remplacée par j. Le coût réduit $\bar{\gamma}_{ij}$ doit désormais intervenir dans le calcul de $S[j]$, qui devient $S[j]+\bar{\gamma}_{ij}$. En posant $\bar{x}_{ij} = 1$, la solution lagrangienne courante reste optimale pour le nouveau programme, car, quelle que soit la nouvelle valeur de $S[j]$, nous devons produire j (nous avons exigé que cette combinaison remplace i). Il vient que le coût $\texttt{LB}^1_{ij}$ de cette solution optimale est $\texttt{LB} + \bar{\gamma}_{ij}$. Donc, d'après le théorème 3.1.2, si $\texttt{LB}^1_{ij} > \texttt{GUB}$, alors nous pouvons fixer à 0 la variable x_{ij}.

$\diamondsuit$

Pour utiliser ces critères de fixation, nous avons évalué la borne inférieure $\texttt{LB}^\delta_{ij}$ simplement en ajoutant la contrainte $x_{ij} = \delta$ au programme et aucune autre. Comme nous venons de le voir, en procédant ainsi, le calcul de cette borne $\texttt{LB}^\delta_{ij}$ est particulièrement simple et rapide.

Nous allons étudier dans la section suivante comment ajouter d'autres contraintes issues de l'interprétation des implications logiques.

3.4.2 FIXATIONS AVANCÉES

Dans cette section, nous développons des critères de fixations plus coûteux en temps que les précédents. Bien sûr, nous les appliquons uniquement dans le cas où les critères de fixations simples ont échoué.

Ces fixations, que nous appellerons *avancées*, sont toutes des applications du théorème 3.1.2. Leur philosophie est la suivante : si le calcul de $\texttt{LB}^\delta_{ij}$ n'a pas permis de fixer la variable x_{ij} à $1 - \delta$, peut-être pouvons nous le compléter en ajoutant encore de nouvelles contraintes traduisant des implications logiques, et l'utiliser soit pour essayer de fixer encore cette variable, soit pour en fixer une autre.

Nous donnons dans cette section un exemple de telles fixations. Nous définissons quatre critères, notés de FA1 à FA4, dont les significations sont les suivantes :

FA1 Nous considérons une variable x_{jj} non fixée qui vaut 1 dans l'actuelle solution optimale lagrangienne. Nous essayons de la fixer à 0. Nous calculons une borne inférieure $\texttt{LB}^1_{jj}$ en partant du principe que si nous imposons la production de j, alors nous pouvons imposer que j ne soit remplacée par aucune combinaison $\ell \succ j$.

FA2 idem que FA1 sauf qu'ici nous supposons que x_{jj} vaut 0 dans l'actuelle solution optimale lagrangienne

FA3 Nous utilisons tous les calculs faits en FA1 et FA2, pour essayer de fixer une variable x_{ij} à 0, avec $i \prec j$. Nous exploitons le fait qu'imposer la substitution de i par j présuppose que la combinaison j soit produite.

FA4 Nous tentons encore une fois de fixer la variable x_{ij} à 0, en utilisant les calculs faits en FA3. Nous complétons la borne inférieure $\mathtt{LB}^1_{ij}$ en tenant compte du fait qu'imposer la substitution de i par j implique que i ne sera remplacée par aucune combinaison $t \succ i$, $t \neq j$

La figure 3.8 représente l'enchaînement de ces critères avancées.

Par conséquent, les différentes contraintes ajoutées à la relaxation lagrangienne courante sont successivement :

- Pour FA1 et FA2 :
  - a) $x_{jj} = 1$
  - b) $x_{j\ell} = 0$ pour tout $\ell \succ j$
- Pour FA3, en plus de a) et b) :
  - c) $x_{ij} = 1$
- Pour FA4, en plus de a), b) et c) :
  - d) $x_{it} = 0$ pour tout $t \succ i$, $t \neq j$

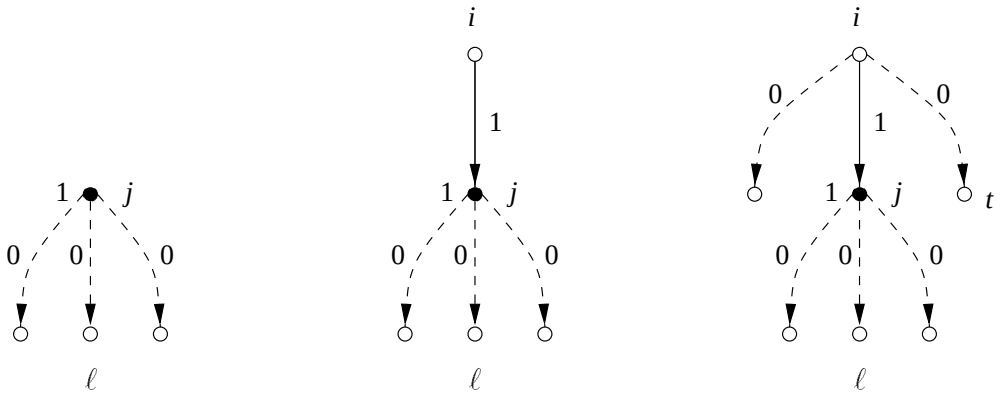


FIG. 3.8 – *Enchaînement des fixations avancées*

Comme nous l'avons déjà signalé dans l'introduction de cette section, l'ajout successif de ces contraintes va nous permettre d'obtenir des bornes inférieures

de plus en plus élevées – dû au fait qu'un plus grand nombre de valeurs $S[.]$ augmentera – et donc d'augmenter nos chances de fixations.

Nous détaillons pour chacune de ces fixations avancées comment évaluer efficacement les bornes inférieures qui leur sont associées.

Fixation avancée FA1 : Soit j une combinaison non fixée, produite dans la solution lagrangienne courante $\bar{x}$, i.e. $\bar{x}_{jj} = 1$, et telle que $\sum_{\ell \succ j} \bar{x}_{j\ell} > 1$. En d'autres termes, il existe au moins un arc sortant de j ayant une valeur 1. Noter que, par construction de $\bar{x}$, le coût réduit de ces arcs est strictement négatif. Nous allons nous intéresser aux solutions optimales appartenant à OPT^{code} dans lesquelles j est choisie. De telles solutions ont toutes les variables $x_{j\ell}$ à 0, car j ne sera substituée par aucune autre combinaison. Nous ajoutons donc à la relaxation lagrangienne courante non seulement la contrainte $x_{jj} = 1$, mais aussi $x_{j\ell} = 0$ pour tout $\ell \succ j$. Les valeurs de $S[.]$ restent inchangées, sauf pour les $\ell \succ j$, tels que $\bar{x}_{j\ell} = 1$ (i.e. $\bar{\gamma}_{j\ell} < 0$ et $j\ell \notin A^0$). Pour ces combinaisons ℓ, $S[\ell]$ croît de $-\bar{\gamma}_{j\ell}$. Pour calculer la valeur $\texttt{LB}^1_{jj}$ du nouveau programme, deux cas se présentent : ou bien, pour tout $\ell \succ j$ tels que $\bar{x}_{j\ell} = 1$, on a $\ell \in V_1$ ou $S[\ell] - \bar{\gamma}_{j\ell} \leq Smin0$, ou bien, il existe $\ell \succ j$, $\ell \notin V_1$ tel que $\bar{x}_{j\ell} = 1$ et $S[\ell] - \bar{\gamma}_{j\ell} > Smin0$.

Dans le premier cas, il est facile de voir que la solution $\bar{x}'$, construite à partir de $\bar{x}$ en posant $\bar{x}'_{jj} = 1$ et $\bar{x}'_{j\ell} = 0$ pour tout $\ell \succ j$, est une solution optimale du nouveau programme, et donc que $\texttt{LB}^1_{jj} = \texttt{LB} - \sum\{\bar{\gamma}_{j\ell} : \bar{x}_{j\ell} = 1\}$. Si $\texttt{LB}^1_{jj} > \texttt{GUB}$, nous pouvons fixer x_{jj} à 0.

Dans le second cas, la solution change plus radicalement, car les combinaisons choisies dans la solution lagrangienne courante risquent de ne plus être celles qui sont associées aux k plus petits $S[.]$. Nous recalculons alors totalement la solution optimale du nouveau programme, et sa valeur $\texttt{LB}^1_{jj}$. Ceci est très simple à faire : il suffit de reprendre l'optimisation standard des relaxations lagrangiennes, vu à la section 2.2, mais cette fois sans les arcs $j\ell$, et en considérant que j est imposée. C'est une tâche d'autant plus simple que seules vont changer les valeurs $S[\ell]$, pour $\ell \succ j$ avec $\bar{\gamma}_{j\ell} < 0$. Si $\texttt{LB}^1_{jj} > \texttt{GUB}$, nous pouvons fixer x_{jj} à 0.

Remarquer que, dans ce dernier cas, la valeur $\texttt{LB}^1_{jj}$ sera supérieure à LB d'au plus $-\sum\{\bar{\gamma}_{j\ell} : \bar{x}_{j\ell} = 1\}$. Donc, nous ne pouvons espérer fixer x_{jj} à 0 que si $\texttt{LB} - \sum\{\bar{\gamma}_{j\ell} : \bar{x}_{j\ell} = 1\} > \texttt{GUB}$. Cependant, nous verrons qu'en cas d'échec, que ce soit dans le premier ou le second cas, nous pouvons quand même utiliser $\texttt{LB}^1_{jj}$ pour fixer d'autres variables. C'est le sujet de la fixation avancée FA3.

Noter aussi que si nous réussissons à fixer x_{jj} à 0, c'est-à-dire à la valeur "opposée" de celle qu'elle avait dans la solution courante, alors il est capital de recalculer complètement la solution de la relaxation lagrangienne avant de passer à un autre calcul, si ce n'est les fixations que celle-ci pourrait impliquer. Ce type de situation sera le sujet de la section 3.6.

Fixation avancée FA2 : Nous nous intéressons à un cas similaire de FA1, excepté le fait que, cette fois, $\bar{x}_{jj} = 0$. Nous voulons voir ce qui se passerait si nous imposions la production du modèle j, c'est-à-dire $x_{jj} = 1$, et donc aussi $x_{j\ell} = 0$ pour tout $\ell \succ j$. Nous gardons l'hypothèse $\sum_{\ell \succ j} \bar{x}_{j\ell} > 1$, car, si nous pouvons fixer la variable x_{jj} à 0 ici, alors nous aurions pu déjà la fixer à l'aide du second critère du théorème 3.4.1. Comme précédemment, nous ajoutons à la relaxation lagrangienne courante les contraintes $x_{jj} = 1$ et $x_{j\ell} = 0$ pour tout $\ell \succ j$. Le calcul d'une solution optimale du nouveau programme, ainsi que de sa valeur $\mathtt{LB}^1_{jj}$, est très proche de celui effectué en FA2, à ceci près que le modèle j appartiendra désormais à la solution alors qu'il en était exclu avant. Faire entrer j dans l'ensemble des combinaisons produites implique que le rôle de $Smin0$ va être tenu par la valeur de $Smax1$. Ceci étant noté, nous reprenons point par point le même raisonnement.

Les valeurs de $S[.]$ restent inchangées, sauf pour les $\ell \succ j$, tels que $\bar{x}_{j\ell} = 1$ (i.e. $\bar{\gamma}_{j\ell} < 0$ et $j\ell \notin A^0$), pour lesquels $S[\ell]$ croît de $-\bar{\gamma}_{j\ell}$. Pour calculer $\mathtt{LB}^1_{jj}$, deux cas se présentent : ou bien, pour tout $\ell \succ j$ tels que $\bar{x}_{j\ell} = 1$, on a $\ell \in V^1$ ou $S[\ell] - \bar{\gamma}_{j\ell} \leq Smax1$, ou bien, il existe $\ell \succ j$, $\ell \notin V^1$ tel que $\bar{x}_{j\ell} = 1$ et $S[\ell] - \bar{\gamma}_{j\ell} > Smax1$.

Dans le premier cas, la solution $\bar{x}'$, construite à partir de $\bar{x}$ en posant $\bar{x}'_{jj} = 1$ et $\bar{x}'_{j\ell} = 0$ pour tout $\ell \succ j$, est une solution optimale du nouveau programme, et donc que $\mathtt{LB}^1_{jj} = \mathtt{LB} - \sum\{\bar{\gamma}_{j\ell} : \bar{x}_{j\ell} = 1\}$.

Dans le second cas, cette solution risque de ne plus être optimale, et nous devons recalculer entièrement $\mathtt{LB}^1_{jj}$. Les remarques faites en FA1, quant à la simplicité des calculs, restent valables.

Si $\mathtt{LB}^1_{jj} > \mathtt{GUB}$, nous pouvons fixer x_{jj} à 0, sinon, nous utiliserons ce nouveau programme pour tenter de fixer d'autres variables grâce au critère de fixation suivant.

Fixation avancée FA3 : Nous sommes dans le cas où, à la fin de FA1 ou de FA2, nous n'avons pas réussi à fixer x_{jj} à 0. Nous pourrions aussi être dans les conditions du second critère du théorème 3.4.1.

Nous allons essayer d'utiliser tous les calculs que nous avons faits précédemment pour fixer des variable x_{ij}. Nous travaillerons donc avec la relaxation lagrangienne complétée par la contrainte $x_{jj} = 1$, ainsi que $x_{j\ell} = 0$ pour tout $\ell \succ j$ si ce critère est appelé après FA1 ou FA2.

Pour alléger les notations, nous désignerons ce programme complété comme étant la relaxation lagrangienne courante, $\bar{x}'$ désignera sa solution optimale, $\mathtt{LB}^1_{jj}$ le coût de cette solution, et $S[.]$ les valeurs remises à jour.

Soit x_{ij}, $j \succ i$, une variable non fixée telle que $\bar{x}'_{ij} = 0$. Par construction, nous avons $\bar{\gamma}_{ij} > 0$. Sachant que substituer i par j implique la production de j, nous pouvons essayer de fixer x_{ij} à 0 en ajoutant la contrainte $x_{ij} = 1$ à la relaxation

lagrangienne courante. Si nous posons $\bar{x}'_{ij} = 1$, la solution $\bar{x}'$ reste optimale pour le nouveau programme. Seule la valeur $S[j]$ change, et croît de $\bar{\gamma}_{ij}$. Le coût de cette solution optimale $\texttt{LB}^1_{ij}$ est donc de $\texttt{LB}^1_{ij} = \texttt{LB}^1_{jj} + \bar{\gamma}_{ij}$.

Si $\texttt{LB}^1_{ij} > \texttt{GUB}$, alors nous pouvons fixer x_{ij} à 0.

En cas d'echec, nous pourrons, à nouveau, tenter de fixer cette variable grâce au critère de fixation suivant.

Fixation avancée FA4 : Supposons que nous soyons dans les conditions d'échec de FA3. Nous pourrions aussi être dans les conditions d'échec du second critère de fixation du théorème 3.4.2. Par convention, $\bar{x}''$ désignera la solution optimale associée à $\texttt{LB}^1_{ij}$, et $S[.]$ les valeurs remises à jour.

Supposons que l'on ait $\sum_{t \succ i}\{\bar{x}''_{it} : \bar{x}''_{it} = 1 \text{ et } t \neq j\} \geq 1$. Sachant que nous ne considérons ici que les solutions optimales de OPT^{code} dans lesquelles i est remplacée par j, et donc par aucune autre combinaison t, nous ajoutons à la dernière relaxation lagrangienne les contraintes $x_{it} = 0$ pour tout $t \succ i$, $t \neq j$.

Il est clair que toute solution optimale de ce nouveau programme aura un coût $new\texttt{LB}^1_{ij}$ supérieur à $\texttt{LB}^1_{ij}$ d'au plus $\sum\{-\bar{\gamma}_{it} : \bar{x}''_{it} = 1 \text{ et } t \neq j\}$. D'où, si $\texttt{LB}^1_{ij}$ augmentée de cette valeur est strictement supérieure à GUB, nous recalculons entièrement $new\texttt{LB}^1_{ij}$. Encore une fois, ce calcul est relativement rapide, vu que seules les valeurs $S[t]$, avec $\bar{x}''_{it} = 1$ et $t \neq j$, vont changer. Si $new\texttt{LB}^1_{ij} > \texttt{GUB}$ alors nous pouvons fixer x_{ij} à 0.

Nous pouvons imaginer beaucoup d'autres fixations avancées. Les quatre qui ont été énoncées ci-dessus décrivent une utilisation rusée du théorème 3.1.2, au sens que les calculs effectués sont simples et rapides. Plus nous ajoutons de nouvelles contraintes, plus nous avons des chances de fixer des variables. Mais en contre-partie, il ne faut pas oublier qu'une mauvaise gestion de l'ajout de ces contraintes entraîne un sur-coût exagéré en temps de calcul.

À la section suivante, nous abordons une dernière méthode de fixation par coût réduit permettant d'exploiter toutes les implications logiques que nous avons décrites à la section 3.3, et ce, de façon récursive.

3.4.3 FIXATIONS FORTES

Nous avons vu à la section 3.3 que la fixation d'une seule variable pouvait entraîner par implication logique beaucoup d'autres fixations. La section précédente donnait un exemple où de telles implications pouvaient être utilisées progressivement pour améliorer la qualité des bornes inférieures $\texttt{LB}^{\delta}_{ij}$. Mais il est évident que la portée de ces critères de fixation étaient limitée par le nombre d'implications logiques inexploitées.

La méthode de fixations que nous employons dans cette nouvelle section, et que nous nommerons *fixations fortes*, est beaucoup plus radicale, dans le sens que nous allons tenir compte de toutes les implications logiques décrites en 3.3.

Essayer de fixer une variable x_{ij} à $1-\delta$ par un critère fort est une tâche des plus simples. Il suffit de fixer temporairement cette variable à δ, puis d'appliquer récursivement toutes les fixations par implication logique.

Il arrive parfois que des contradictions apparaissent immédiatement. Cela se produit si nous devons fixer une variable à 0 par une première implication logique, puis à 1 par une seconde, ou si le nombre de combinaisons fixées est trop grand ($|V^1| > k$ ou $|V^0| > |V| - k$). Dans ce cas, il est clair que nous pouvons fixer la variable x_{ij} à δ.

Si aucune contradiction n'est détectée, nous calculons une solution optimale de la nouvelle relaxation lagrangienne, et si son coût $\texttt{LB}_{ij}^{\delta}$ est strictement supérieur à GUB, alors nous pouvons fixer encore une fois la variable x_{ij} à δ.

Par soucis d'efficacité, toutes les fixations temporaires sont mémorisées dans une liste afin de leur rendre leur valeur initiale en cas d'échec de la fixation. Cependant cette méthode demeure très coûteuse en temps de calcul. Pour l'utiliser, nous attendons que le pourcentage de variables fixées dépasse un certain seuil. Nous commençons par appliquer cette méthode aux variables de sommets x_{jj}, puis, à des moments cruciaux de l'algorithme lagrangien, tels que la fin de la phase du sous-gradient ou après la convergence d'un faisceau (voir section 2.5), nous l'appliquons aux variables d'arcs x_{ij}, avec $i \prec j$. Pour les grandes instances, le nombre d'arcs peut être extrêmement élevé. Il est conseillé de ne tester la fixation forte que d'un nombre limité de variables d'arcs (5000 semble donner de bons résultats), afin de ne pas ralentir exagérément l'algorithme. De plus, il est inutile d'essayer de fixer à 1 ces variables d'arcs, car leur fixation à 0 n'a que très peu d'implications logiques. Pour choisir les arcs que nous devons tester, nous évaluons la valeur $\texttt{LB}_{ij}^{1}$ en ajoutant à $\texttt{LB}_{jj}^{1}$ le coût réduit de l'arc ij s'il est positif, et en retranchant à $\texttt{LB}_{ii}^{0}$ les coûts réduits de tous les arcs ij' négatifs, sauf celui de ij. Ceci n'est qu'une évaluation, car il peut arriver que ces coûts réduits soient déjà pris en compte dans le calcul de $\texttt{LB}_{jj}^{1}$ et de $\texttt{LB}_{ii}^{0}$. Les arcs retenus sont ceux pour lesquels cette évaluation est maximale.

3.5 TRADUIRE DE NOUVELLES CONTRAINTES EN CRITÈRE DE FIXATIONS

La connaissance de nouvelles contraintes valides pour toutes les solutions optimales du problème peut être particulièrement bénéfique. Une fois ajoutées au

programme, ces contraintes permettent non seulement de restreindre l'ensemble des solutions réalisables, ce qui facilite la recherche d'une solution optimale, mais aussi, a priori, d'améliorer la qualité des bornes inférieures obtenues par les différentes relaxations, qu'elles soient linéaires ou lagrangiennes.

Si l'on souhaite tenir compte de ces nouvelles contraintes pour améliorer la qualité des relaxations lagrangiennes, trois possibilités s'offrent à nous :

1. soit nous les ajoutons directement dans le corps même des relaxations lagrangiennes
2. soit nous les relâchons en leur associant de nouveaux multiplicateurs lagrangiens
3. soit nous les exploitons pour déduire de nouveaux critères de fixations

Des trois possibilités, la première est celle qui fournira a priori la meilleure borne inférieure. Cependant, un algorithme lagrangien n'est efficace que si, pour un ensemble de multiplicateurs lagrangiens donné, l'évaluation de la valeur de la fonction objectif lagrangienne est rapide. L'ajout de ces nouvelles contraintes peut s'avérer catastrophique, voire rendre le problème lagrangien NP-complet. Seule une étude poussée de l'effet de ces contraintes sur la qualité de la nouvelle borne inférieure lagrangienne d'une part, et de la complexité du nouveau problème lagrangien d'autre part, permettra de décider si cette possibilité est envisageable.

La seconde alternative possible qui consiste à relâcher les nouvelles contraintes, présente l'avantage de préserver la complexité du problème lagrangien, mais en revanche le très grand désavantage de générer de nouveaux multiplicateurs qui devront être ajustés au même titre que les autres. En pratique, cette méthode ne peut être employée que si le nombre de nouvelles contraintes est très restreint, ce qui n'arrive que très rarement.

La troisième possibilité est donc de déduire toutes ou une partie des conséquences qu'elles entraînent. Nous allons illustrer cette méthode avec le problème de la diversité et les contraintes dites *de chemin*, qui sont valides pour toutes les solutions optimales de l'ensemble OPT^{code} défini à la section 3.3.1.

Définition 5.3.9

Soient trois combinaisons i, j et ℓ telles que $i \prec j \prec \ell$. Nous appellerons "contrainte de chemin associée à (i, j, ℓ)" la contrainte :

$$x_{i\ell} \leq x_{j\ell}$$

Théorème 3.5.2

Les contraintes de chemin sont valides pour toutes les solutions optimales du problème de la gestion de la diversité appartenant à OPT^{code}.

Preuve de 3.5.2 : Soient $i \prec j \prec \ell$ et $\bar{x}$ une solution optimale appartenant à OPT^{code}. Nous démontrons ce théorème en prouvant la validité de l'implication suivante : $\bar{x}_{i\ell} = 1 \Rightarrow \bar{x}_{j\ell} = 1$

Supposons que $\bar{x}_{i\ell} = 1$, c'est-à-dire que i est remplacée par ℓ dans cette solution optimale. Il est clair que j n'est pas produite, sinon remplacer i par j serait plus profitable que l'actuelle substitution. Supposons par l'absurde que j soit remplacée par une combinaison $\ell' \neq \ell$. Par transitivité de l'ordre partiel $\succ$, il vient que i aurait pu être remplacée par ℓ'. La solution $\bar{x}$ étant optimale, nous en déduisons que $c_{\ell'} \geq c_\ell$, de plus, j pouvant être remplacée par ℓ il vient que $c_{\ell'} \leq c_\ell$, et donc que $c_{\ell'} = c_\ell$. Or, par injectivité du code décimal, nous savons que $code_{\ell'} \neq code_\ell$, d'où la contradiction avec le fait que la solution optimale $\bar{x}$ appartient à OPT^{code}. ♢

La traduction de ces contraintes en critères de fixations s'exprime de la façon suivante :

Théorème 3.5.3

Soient $i \prec j \prec \ell$

1. *Si la variable* $x_{i\ell}$ *est fixée à 1 et si* $j\ell \notin A^1$, *alors nous pouvons fixer à 1 la variable* $x_{j\ell}$, *i.e.* $A^1 \leftarrow A^1 + \{j\ell\}$
2. *Si la variable* $x_{j\ell}$ *est fixée à 0 et si* $i\ell \notin A^0$, *alors nous pouvons fixer à 0 la variable* $x_{i\ell}$, *i.e.* $A^0 \leftarrow A^0 + \{i\ell\}$

Preuve de 3.5.3 : trivial d'après le théorème 3.5.2 ♢

3.6 UNE ERREUR À ÉVITER

L'algorithme traditionnel des relaxations lagrangiennes consiste à itérer la séquence ci-dessous :

– Résoudre la relaxation lagrangienne
– Essayer d'améliorer `GUB`

– Appliquer la phase de fixations de variables
– Mettre à jour les multiplicateurs

Avec, dans la fonction “Mettre à jour les multiplicateurs”, un test qui arrête l’algorithme si la solution lagrangienne est réalisable, et donc optimale (ne pas oublier que nous n’avons relâché que des équations).

Les fixations que nous avons développées dans ce chapitre rendent cet algorithme incorrect, comme nous allons le montrer dans un exemple.

Dans la figure 3.9, il ne reste, disons, que trois combinaisons non fixées, i, j et ℓ, dont deux vont devoir être produites. Supposons que $S[i] = -9$, $S[j] = -7$ et $S[\ell] = -6$. Ainsi, les combinaisons i et j apparaissent dans la solution lagrangienne toutes les deux avec des arcs ti et tj de coûts réduits $\bar{\gamma}_{ti} = -4$ et $\bar{\gamma}_{tj} = -2$. Admettons que nous réussissions à fixer x_{ti} à 1, i.e. ti entre dans A^1, et donc, par implication logique, la variable x_{tj} est fixée à 0.

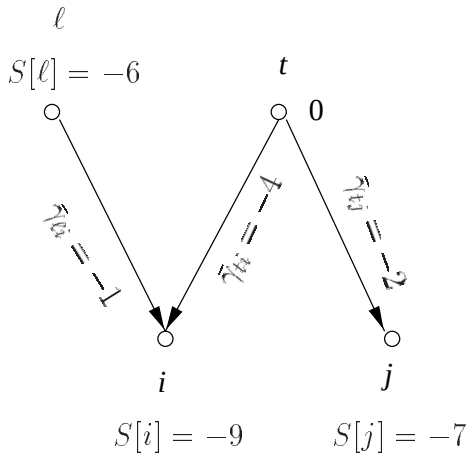


FIG. 3.9 – *Exemple d’une erreur à éviter*

Maintenant, supposons qu’une fois ces fixations effectuées, toutes les combinaisons soient, ou bien choisies, ou bien remplacées par exactement une autre. Nous allons à l’étape “Mettre à jour les multiplicateurs” qui va immédiatement arrêter l’algorithme, la solution étant réalisable. Or, celle-ci n’est pas optimale, car la fixation de tj à 0 a changé la valeur de $S[j]$, qui vaut désormais (-7)-(-2)=-5, et est donc devenue supérieure à $S[\ell]$. Par conséquent, la solution lagrangienne optimale n’est plus $\{i, j\}$ et les autres combinaisons fixées à 1, mais $\{i, \ell\}$ et ces mêmes combinaisons.

Il est donc primordial, quand la fixation des variables risque de changer la solution lagrangienne, de revenir à l’étape “Résoudre la relaxation lagrangienne”, et non pas de se rendre à l’étape “Mettre à jour les multiplicateurs”. Ceci est

important, non seulement parce que nous risquons d'arrêter l'algorithme avec une solution supposée optimale alors qu'elle ne l'est pas, mais aussi parce que nous ne mettrons pas à jour correctement les multiplicateurs.

Ce phénomène se produit uniquement lorsqu'on fixe à δ une variable qui avait la valeur "opposée", $1 - \delta$, dans la solution lagrangienne courante.

3.7 AGGLOMÉRATION DE SOMMETS ET D'ARCS

Dans cette section, nous abordons une méthode très différente de réduction de problème qui permet d'éliminer des variables sans les fixer.

Cette technique, que nous nommerons *agglomération de variables*, consiste à éliminer des sommets et des arcs en exploitant la structure d'hypergraphe décrite à la section 3.3.4. Elle s'appuie sur le théorème suivant :

Théorème 3.7.1

Soient i et j deux combinaisons telles que $E_i = E_j$.
Les deux propositions suivantes sont vraies :

1. *Au moins une des deux combinaisons i et j est fixée à 0*
2. *Dans toute solution optimale $\bar{x}$ appartenant à OPT^{code}, $\bar{x}_{i\ell} = \bar{x}_{j\ell}$ pour tout $\ell \in E_i = E_j$*

Preuve de 3.7.1 : Soient i et j deux combinaisons telles que $E_i = E_j$, c'est à dire que i et j peuvent être remplacées par les mêmes combinaisons.

Sachant que ces deux combinaisons ne peuvent pas appartenir simultanément à $E_i = E_j$ (sinon nous aurions $i \succ j$ et $j \succ i$), au moins l'une des deux, j par exemple, est fixée à 0.

Soit $x^\star$ une solution optimale de OPT^{code}.

Nous savons que i sera substituée par une combinaison $\ell \in E_i$ dans cette solution optimale. Remarquer que ℓ peut être égale à i si celle-ci n'est pas fixée à 0, et dans ce cas substituer i par elle-même revient à la produire.

La combinaison $\ell' \in E_i$ qui remplace j dans cette solution $x^\star$ est de coût inférieur ou égal à celui de ℓ, sinon remplacer j par ℓ serait plus profitable, et la solution ne serait pas optimale. En inversant les rôles de ℓ et ℓ' nous en déduisons que ℓ est de coût inférieur à celui de ℓ', et donc que ces deux combinaisons ont un coût égal. Par construction des solutions optimales appartenant à OPT^{code}, nous pouvons conclure que $\ell = \ell'$, d'où le résultat annoncé. $\diamondsuit$

Sachant que, pour toute combinaison $\ell \in E_i = E_j$, les variables $x_{i\ell}$ et $x_{j\ell}$ auront les mêmes valeurs dans toutes les solutions optimales de OPT^{code}, il devient inutile de toutes les garder. Nous pouvons donc agglomérer les sommets i et j en un seul sommet, que nous nommerons encore i. Le fait d'appeler i ce nouveau sommet n'est pas un hasard, dans le sens que cette opération peut être vu comme une "absorption" de j par i : le nouveau sommet i possédera le même coût, le même état de fixation, et les mêmes arcs entrant et sortant que l'ancien i. En revanche il aura une demande égale à $d_i + d_j$. Le fait d'additionner les demandes est logique vu que cette nouvelle combinaison représentera les deux combinaisons simultanément.

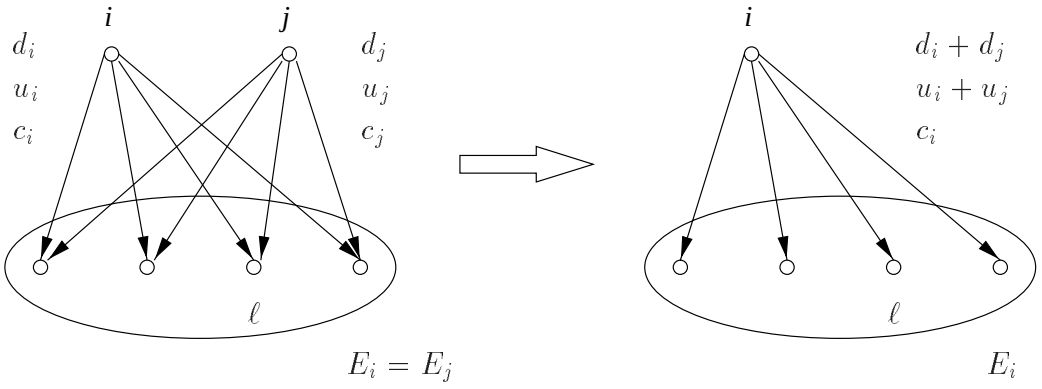


FIG. 3.10 – *Agglomération de variables*

De plus, nous définissons le multiplicateur lagrangien de cette nouvelle combinaison comme étant égal à $u_i + u_j$. En procédant ainsi, les valeurs $S[\ell]$, pour $\ell \in E_i$, peuvent augmenter, et donc améliorer peut-être la qualité de la borne inférieure lagrangienne. Ceci est dû aux observations suivantes :

- $S[\ell] = c_\ell d_\ell - u_\ell + \sum_{\ell' \prec \ell} \min(0, c_\ell d_{\ell'} - u_{\ell'})$
- $\min(0, c_\ell(d_i + d_j) - (u_i + u_j)) \geq \min(0, c_\ell d_i - u_i) + \min(0, c_\ell d_j - u_j)$, pour le cas où $\ell \neq i$,
- $c_\ell d_j - u_j \geq \min(0, c_\ell d_j - u_j)$ pour le cas où $\ell = i$.

Nous nous retrouvons ainsi dans l'une des situations décrites à la section 3.6 où les valeurs des $S[.]$ peuvent changer. Si la solution lagrangienne courante est réalisable pour le problème initial, nous ne pouvons pas conclure qu'elle est optimale pour ce problème, car nous n'avons plus la certitude que cette solution est encore optimale pour la relaxation lagrangienne.

3.8 CONCLUSION

Nous avons vu dans ce chapitre comment la fixation d'une seule variable peut entraîner une réduction substantielle de la taille du problème. Cette réduction est d'autant plus conséquente que les implications logiques sont nombreuses.

Seule une étude poussée du problème de la gestion de la diversité nous a permis de déduire les conséquences logiques qu'implique la production d'une combinaison ou sa substitution. L'extension de cette technique de fixation à d'autres problèmes passe donc obligatoirement par une telle étude. Noter que de nombreux résultats présentés dans ce chapitre sont facilement généralisables aux problèmes de localisation traditionnels.

Nous verrons au chapitre 6 que pour bon nombre d'instances, cette méthode de fixation nous a permis de réduire considérablement la taille du problème, voire de prouver l'optimalité des solutions. Cependant, nous verrons aussi qu'il existe des instances pour lesquelles elle n'a eut qu'une portée très limitée. Cela s'explique par son effet "boule de neige" dûe à la très grande récursivité des fixations : paradoxalement, moins nous réussissons à fixer des variables, et moins nous avons de chances d'en fixer à nouveau. La réduction de la taille du problème implique la réduction de l'ensemble des solutions étudiées, et donc l'augmentation des valeurs lagrangiennes. Si cette réduction n'est pas suffisante, les valeurs lagrangiennes resteront peu élevées.

Il est évident que la réussite de cette méthode est très dépendante de la qualité de la meilleure solution trouvée, c'est-à-dire celle qui nous a permis de définir la valeur `GUB`. La recherche d'une telle solution est développée dans le chapitre suivant.

4

ALGORITHMES DE RÉSOLUTION APPROCHÉE

Sommaire

Dans ce chapitre, nous essayons de construire une “bonne” solution réalisable du problème. Une grande majorité des heuristiques présentées vont exploiter les différents renseignements donnés par les relaxations lagrangiennes.

Dans les algorithmes lagrangiens, l'heuristique qui est le plus souvent proposée consiste à exploiter directement les solutions optimales lagrangiennes (voir par exemple [Bea93b]). Elle revient à construire une solution réalisable satisfaisant une à une les contraintes relâchées. Dans notre cas, elle revient à choisir les k combinaisons de la solution lagrangienne courante, et à leur affecter, de façon optimale, les combinaisons non produites.

Cette heuristique est particulièrement mauvaise pour notre problème. La situation suivante fournit un début d'explication. Supposons qu'il existe deux combinaisons très proches, que ce soit par rapport à leur coût, à l'ensemble des combinaisons qu'elles peuvent remplacer ou à la valeur de leur multiplicateur lagrangien. Les valeurs $S[.]$ qui leur sont associées seront alors quasiment identiques. Cela implique que ces deux combinaisons auront le même comportement par rapport à la solution lagrangienne. En particulier, si l'une y apparaît, l'autre y apparaîtra probablement aussi, bien qu'elle n'apporte rien à la valeur de la solution.

Cette remarque démontre l'importance d'élaborer des heuristiques plus adaptées à notre problème. Dans la première section, nous présentons les heuristiques gloutonnes que nous avons mis au point afin d'obtenir des solutions réalisables du problème. Dans une seconde section, nous discutons de l'utilité d'une procédure d'amélioration par recherche de voisinage, dite `2-OPT`. Dans la section suivante, nous traitons du bon emploi de toutes ces heuristiques, de leurs avantages, de leurs inconvénients et de leurs limites. Enfin, nous concluons ce chapitre sur une remarque quant à l'exploitation des fixations d'arcs obtenues au chapitre 3.

4.1 HEURISTIQUES GLOUTONNES

Nous présentons dans cette section les différentes heuristiques que nous employons tout au long de notre algorithme lagrangien. Le principal point commun qui les relie est d'être des algorithmes *gloutons*. Par définition, un algorithme est dit glouton s'il permet de construire un ensemble R en suivant les deux règles suivantes :

Règle 1 : à chaque étape, l'ensemble R est complété par le meilleur élément candidat pour y entrer

Règle 2 : tout choix fait à une certaine étape ne peut pas être remis en cause aux étapes suivantes

Pour le problème de la gestion de la diversité, l'ensemble R contiendra l'ensemble des combinaisons que nous produirons. Il est possible de l'initialiser avec certaines combinaisons que l'on sait être produites (ou au moins que l'on espère être produites) dans des solutions optimales du problème. Toutes les heuristiques que nous allons présenter suivent donc le schéma ci-dessous :

Algorithme 4 Algorithme glouton générique

initialiser l'ensemble R
tant que $|R| < k$ **faire**
  chercher la "meilleure" combinaison $j \notin R$ non fixée à 0
  $R \leftarrow R \cup \{j\}$
fin tant que

La notion de "meilleur candidat" doit être définie très précisément. Elle dépend, en général des choix précédents. Les heuristiques que nous allons décrire diffèrent dans la définition de "meilleur candidat" et dans l'initialisation de R.

La solution obtenue est ensuite améliorée par une procédure, dite `2-OPT`, que nous décrivons plus loin (section 4.2).

La rapidité de ces algorithmes est assurée par le nombre très limité de boucles effectuées : il y en a autant que de combinaisons à introduire dans R pour obtenir le nombre k désiré. Le coût en temps de calcul de chacune de ces boucles provient de l'évaluation du critère qui permet de définir la meilleure combinaison à introduire dans R. Ce coût est infime, même pour les quatre premières heuristiques, car le calcul du gain qu'apporte la production d'une combinaison supplémentaire $j \notin R$ peut s'effectuer très efficacement : d'une part nous parcourons les combinaisons n'appartenant pas à R que j peut remplacer, et nous calculons le gain que nous pourrions tirer si nous les substituions par j et non par les éléments de R, et d'autre part, nous ajoutons à ce gain celui qui provient de la non substitution de j.

La particularité des quatre premières heuristiques est de définir dynamiquement ce critère de classement. En effet, le gain que nous pouvons tirer de la production d'une combinaison dépend très fortement des combinaisons qui sont déjà produites. Ces quatre heuristiques se différencient les unes des autres par l'initialisation de l'ensemble R.

4.1.1 HEURISTIQUE GLOUTONNE NAÏVE

La plus simple initialisation de l'ensemble R est de ne prendre que les combinaisons fixées à 1. C'est donc aussi l'heuristique gloutonne qui demandera le plus

de temps de calcul, car c'est celle où le nombre de combinaisons restant à trouver est le plus important. A chaque étape, la combinaison choisie est celle qui fait le plus diminuer le coût total de la production.

Excepté la fixation des combinaisons à 1, cette heuristique ne tient pas compte des différents renseignements que l'on a pu recueillir tout au long de l'algorithme des relaxations lagrangiennes. Nous avons décidé de restreindre son utilisation uniquement à la recherche d'une première solution réalisable, c'est-à-dire à l'initialisation de la valeur de GUB que nous avons définie dans les chapitres précédents.

Ce choix est motivé aussi par le fait que les heuristiques suivantes sont tout aussi performantes, voire très souvent meilleures, et de complexité comparable.

4.1.2 HEURISTIQUE GLOUTONNE DES 2 PRÉSENCES

L'heuristique des *2 présences* que nous décrivons dans cette section est basée sur une exploitation directe de la solution optimale de la relaxation lagrangienne courante.

Comme nous l'avons signalé précédemment, considérer que toutes les combinaisons qui constituent cette solution sont de bons candidats à appartenir à une solution optimale de notre problème, n'est pas une bonne idée. En effet, cela risque d'entraîner le choix d'une combinaison superflue, dans le sens qu'elle ne remplacera aucune autre combinaison.

Pour éviter cette situation gênante, nous filtrons dans cette heuristique cette solution lagrangienne courante. Pour cela nous construisons l'ensemble R à l'aide des combinaisons qui sont présentes simultanément dans cette solution courante et dans la meilleure solution lagrangienne trouvée jusqu'à présent, c'est-à-dire celle qui a servi à définir GLB. Ainsi, la probabilité d'une apparition fortuite d'une mauvaise combinaison dans cet ensemble initial R diminue sensiblement. Sachant que dans chacune des deux solutions lagrangiennes qui définissent R, le nombre de combinaisons produites est égal à k, le nombre de combinaisons produites simultanément dans ces deux solutions, c'est-à-dire la cardinalité initiale de R, est inférieur à k. Il est très rare que R soit dès le début de cardinalité k, à moins que la preuve de l'optimalité de la meilleure solution réalisable ne soit proche. Nous pouvons donc légitimement espérer que les combinaisons qui vont compléter R rectifieront une éventuelle erreur, tout au moins relativement. Pour compléter R nous choisissons, comme à la section précédente, la combinaison dont la production fournit le meilleur profit.

L'utilisation systématique d'une procédure d'amélioration par voisinage après cette heuristique s'avère souvent efficace. Ceci s'explique aisément par le fait qu'une telle procédure 2-OPT, qui consiste à échanger une combinaison de R

par une qui n'appartient pas à R, nous offre la possibilité d'exclure une combinaison superflue pour l'échanger par une autre. Cependant, la principale limite de cette heuristique apparaît lorsque plus de deux combinaisons quasiment identiques sont produites, car dans ce cas un seul échange ne suffit pas à les exclure toutes de R. L'heuristique décrite à la section suivante va nous permettre de palier, tout au moins partiellement, à ce problème.

4.1.3 HEURISTIQUE GLOUTONNE DES 3 PRÉSENCES

La philosophie de l'heuristique gloutonne des *3 présences* est très proche de celle des 2 présences développée à la section précédente. En effet, les combinaisons qui composent initialement l'ensemble R doivent encore appartenir simultanément à la solution optimale de la relaxation lagrangienne courante et à celle qui a permis de définir la valeur de GLB. La particularité de cette nouvelle heuristique est que nous demandons, en plus, que ces combinaisons initiales soient produites dans la meilleure solution réalisable trouvée jusqu'à présent, c'est-à-dire celle qui est associée à la valeur GUB actuelle.

Exiger qu'une combinaison soit présente dans la meilleure solution réalisable est particulièrement restrictif, car cela assure qu'aucune combinaison superflue n'appartiendra à R, ou, si elle venait à y appartenir, ce serait parce qu'elle fut la meilleure candidate lors d'une des boucles de l'algorithme générique 4 décrit dans l'introduction de cette section, et donc nous pouvons la considérer comme ayant une utilité non négligeable.

La force de ce troisième critère de présence peut être aussi un inconvénient, car il peut arriver qu'aucune combinaison ne satisfasse les trois présences simultanément, hormis les combinaisons fixées à 1. Dans ce cas, cette heuristique serait équivalente à l'heuristique gloutonne naïve présentée à la section 4.1.1. Cependant, cela ne se produit que très rarement. Il suffit qu'une seule combinaison non fixée satisfasse ces trois critères, pour que la solution réalisable obtenue puisse être radicalement différente de celle fournie par cette première heuristique gloutonne.

4.1.4 HEURISTIQUE GLOUTONNE AVEC EXCLUSION

Le point commun entre les heuristiques que nous venons de décrire est de privilégier les combinaisons qui appartiennent fréquemment aux solutions optimales des différentes relaxations lagrangiennes rencontrées, ou aux solutions réalisables trouvées. Pour certaines instances, cette stratégie peut mener systématiquement à un échec. L'exemple suivant illustre une telle instance.

Soit $V = \{a, b, c, d, e, f\}$, et $A = \{ad, bd, cd, df, ef\}$, ce qui signifie notamment que f est une combinaison terminale, elle a donc été fixée à 1 dès le début de l'algorithme lagrangien. Les coûts unitaires de a, b et c sont de 1, ceux de d et e de 2, et celui de f de 3. Les demandes sont de 3 pour a, b et c, de 4 pour e, et de 1 pour d et f. Enfin, nous supposons que le nombre de combinaisons que l'on souhaite produire est de $k = 4$. La solution gloutonne sera presque toujours construite en choisissant de produire f, puis d puis e et enfin a, b ou c. Un telle solution aura un coût total de 28. Or la solution optimale est celle qui consiste à produire $\{a, b, c, f\}$. Cette solution optimale unique est très difficilement atteignable par les algorithmes gloutons définis dans les sections précédentes, même si nous les faisons suivre d'une procédure 2-OPT.

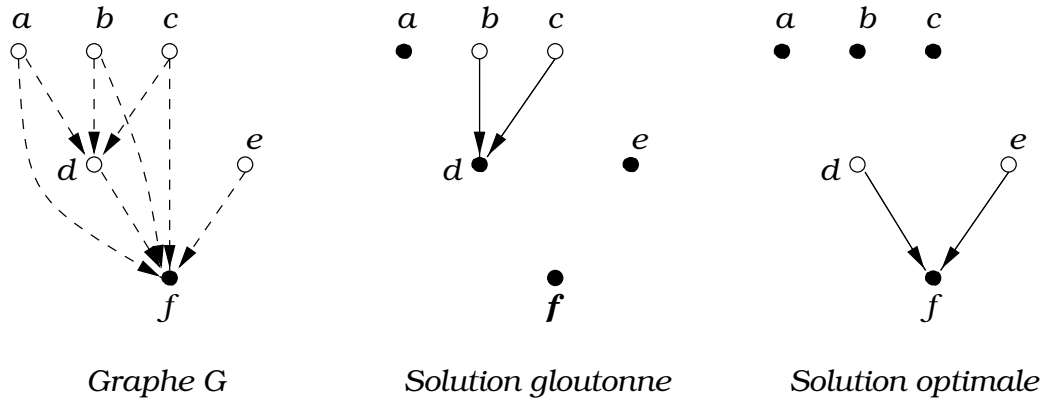


FIG. 4.1 – *Exemple d'instance pouvant poser problème*

L'heuristique gloutonne proposée dans cette section initialise l'ensemble R en suivant trois étapes. D'abord nous classons les combinaisons suivant les nombres croissants d'apparition dans toutes les solutions réalisables construites jusqu'à présent. Nous limitons ces nombres d'apparition aux seules solutions issues de procédures 2-OPT (voir section 4.2) afin d'éliminer des apparitions fortuites. Puis nous construisons l'ensemble initial R à l'aide des k combinaisons ayant obtenu les plus grands scores. Et enfin, nous excluons de R un certain nombre de combinaisons en tête de ce classement.

Les places ainsi libérées sont comblées itérativement par les combinaisons dont la production génère le plus grand profit.

Le choix du nombre de combinaisons supprimées est absolument arbitraire, et doit être affiné par des tests numériques. Pour le problème de la diversité, les tests nous ont prouvé qu'écarter 2 combinaisons à la fois était le meilleur choix.

Une alternative serait de n'exclure aucune combinaison. Nous construirions

alors immédiatement un ensemble R de cardinalité k, qui serait le mélange des solutions heuristiques précédentes. Après quelques tests décevants, nous avons décidé de l'abandonner. Ces mauvais résultats peuvent s'expliquer par le fait qu'une solution issue d'un 2-OPT est un optimum local. Les présences des combinaisons dans de telles solutions sont donc très dépendantes les unes des autres. Vouloir créer une nouvelle solution en ne prenant que des fragments de solutions optimisées est a priori illusoire.

Afin de ne pas rejeter toujours les mêmes combinaisons, il peut être judicieux de les mémoriser, et de s'interdire de les exclure pendant un certain nombre d'appels de cette heuristique.

Enfin, il est important de noter que cette heuristique gloutonne est l'une des plus rapides, car le nombre de combinaisons restant à trouver est très limité.

Dans la section suivante, nous proposons deux autres heuristiques dans lesquelles nous choisissons en une seule itération les k combinaisons destinées à être produites.

4.1.5 HEURISTIQUE DES FRÉQUENCES

Le critère de classement qui nous permet de désigner la "meilleure" combinaison à chaque étape de l'algorithme 4, est ici le nombre de solutions optimales lagrangiennes auxquelles appartient chaque combinaison. Pour que ce critère soit significatif, nous nous restreignons aux solutions qui ont été les meilleures solutions lagrangiennes connues lors de leur calcul, c'est-à-dire celles qui ont amélioré la valeur de GLB. De plus, nous considérons comme identiques, et donc comptées une seule fois, les solutions lagrangiennes qui consistent à produire le même ensemble de combinaisons, même si les valeurs des arcs sont différentes (nous rappelons que la variable associée à un arc ij vaut 1 si le coût réduit de celui-ci est strictement négatif et si j est produite).

Cette heuristique s'est souvent avérée être l'une des plus efficaces. Son succès provient certainement du fait qu'elle exploite simultanément toutes les "bonnes" solutions lagrangiennes rencontrées. Comme nous l'avons remarqué précédemment, exploiter une seule solution lagrangienne ne donne généralement pas de bons résultats. En exploiter plusieurs à la fois nous permet en revanche de privilégier les combinaisons dont l'utilité est prouvée. Ceci est d'autant plus vrai que l'on avance dans l'algorithme lagrangien.

4.1.6 HEURISTIQUES DES FIXATIONS

Nous avons vu au chapitre 3 comment fixer certaines variables. L'idée maître qui nous permet de fixer une variable à 0 (respectivement à 1) réside en l'évaluation d'une borne inférieure de la valeur de toutes les solutions optimales du problème initial dans lesquelles cette variable vaut 0 (respectivement 1). La définition de "meilleure" combinaison (voir l'algorithme 4) que nous utilisons dans cette section nous permet de tirer profit de ces calculs en cas d'échec de la fixation.

Notons, pour chaque combinaison i non fixée, $best\mathtt{LB}_{ii}^0$ et $best\mathtt{LB}_{ii}^1$ comme étant les valeurs des meilleures bornes inférieures calculées dans les procédures de fixations lors de l'essai de fixation de i respectivement à 1 et à 0. La valeur $best\mathtt{LB}_i^0$ (respectivement $best\mathtt{LB}_{ii}^1$) est donc la meilleure borne inférieure des solutions optimales ayant x_{ii} à 0 (respectivement à 1) que l'on ait trouvée jusqu'à présent. Le critère de classement que nous utilisons ici est la valeur de $best\mathtt{LB}_{ii}^0 - best\mathtt{LB}_{ii}^1$ pour chaque combinaison i. Il est en effet logique de choisir en priorité une combinaison pour laquelle cette valeur est grande, car cela signifie que $best\mathtt{LB}_{ii}^0 \gg best\mathtt{LB}_{ii}^1$, autrement dit, i a plus de chance d'être fixée à 1 qu'à 0.

Cette heuristique est d'autant plus efficace que les valeurs $best\mathtt{LB}_{ii}^0$ et $best\mathtt{LB}_{ii}^1$ sont de bonne qualité. Il est donc tout à fait approprié de l'appeler après chaque fixation forte (c.f. 3.4.3).

4.2 PROCÉDURES 2-OPT

Très souvent, les solutions réalisables fournies par les heuristiques décrites à la section précédente sont de qualité moyenne. Nous proposons dans cette section une procédure d'amélioration par voisinage, nommée `2-OPT`.

L'idée principale d'une procédure `2-OPT` est d'échanger une combinaison produite par une qui ne l'est pas dès que cela peut nous procurer du gain. Ainsi, cette procédure nous fait passer d'une solution réalisable, à une autre, jusqu'à nous fournir un optimum local. Par conséquent, nous définissons la notion de *voisinage* d'une solution comme étant l'ensemble de toutes les solutions réalisables dans lesquelles sont produites $k-1$ combinaisons de la première solution, plus une $k^{\text{ème}}$ combinaison qui n'était pas produite initialement.

Dans la section 4.2.1 nous étudions les trois stratégies d'échanges possibles, en exhibant notamment leurs avantages et leurs inconvénients. Dans la section 4.2.2, nous développons une version bridée de cette procédure afin de limiter son

principal handicap, son coût en temps de calcul, sans perdre trop en qualité de solution. Cela revient, comme nous le verrons plus tard, à restreindre la notion de voisinage définie précédemment.

4.2.1 DIFFÉRENTES STRATÉGIES D'ÉCHANGES

Une itération de 2-OPT consiste à intervertir une combinaison i qui était produite par une combinaison j qui ne l'était pas. Nous effectuons cet échange de i par j uniquement si la nouvelle solution ainsi obtenue est de coût strictement inférieur à la solution de départ. Mais le choix d'un tel échange n'est généralement pas unique. Nous traitons dans cette section de ces différents choix.

L'algorithme général d'une procédure 2-OPT peut s'énoncer comme suit :

Algorithme 5 Procédure 2-OPT générale

$R := \{i | i$ est produite dans la solution courante$\}$
tant que il existe un échange profitable $(i, j) \in R \times (V \setminus R)$ **faire**
  $R \leftarrow R \cup \{j\} \setminus \{i\}$
fin tant que

Une idée élémentaire est de définir les combinaisons $i \in R$ et $j \in V \setminus R$ que nous allons intervertir, comme étant les premières rencontrées dont l'échange fournit un profit. L'avantage de cette stratégie est la rapidité de la recherche de i et j, mais le très grand désavantage est le nombre d'échanges que nous risquons d'effectuer. En effet, avec une telle stratégie, il arrive souvent qu'au lieu de permuter directement une combinaison i par une combinaison j nous soyons obligés de réaliser une suite d'échanges intermédiaires (i, u_0), (u_0, u_1), (u_1, u_2), ..., (u_n, j), ce qui fait perdre beaucoup de temps de calcul. Pour limiter ce type de désagrément, nous avons décidé de parcourir d'abord l'ensemble R, puis, pour une combinaison $i \in R$ donnée, de chercher la combinaison $j \in V \setminus R$. Ce choix s'explique par le fait que nous nous intéressons principalement aux petites valeurs de $k = |R|$ (voir chapitre 1), ce qui implique que $|R| \ll |V \setminus R|$. Le choix inverse (parcourir $V \setminus R$ puis R) aurait a priori fait entrer dans R bon nombre de combinaisons qui en seraient ressortis aux itérations suivantes. Malgré cela, cette procédure 2-OPT est demeurée bien trop lente, nous interdisant de l'appeler fréquemment. Cette stratégie a donc été rapidement abandonnée.

Une seconde idée, à l'opposé de la première, est de chercher à chaque itération la transposition $(i, j) \in R \times V \setminus R$ qui donne la plus grande diminution du coût total de production. Cette stratégie nécessite que peu d'itérations, mais chacune

d'elles est extrêmement coûteuse en temps de calcul car il faut tester l'ensemble de tous les échanges possibles. Cependant, la qualité de la solution obtenue est quelquefois inégalable. Ce type de procédure a donc été retenu, mais utilisé avec parcimonie.

La troisième idée, celle que nous avons utilisée le plus fréquemment, se situe à mi chemin entre les deux précédentes. Nous parcourons l'ensemble des combinaisons $i \in R$ produites, puis, pour i donnée, nous cherchons la combinaison $j \in V \setminus R$ dont l'échange avec i donne le plus grand gain. Cette procédure nécessite beaucoup moins d'itérations que la première, les échanges intermédiaires étant pour la plupart éliminés, et la qualité de la solution finale est souvent proche, parfois même meilleure que celle de la seconde stratégie. Elle est de plus relativement rapide car dès que nous rencontrons une combinaison i pour laquelle il existe un échange (i, j), nous l'effectuons.

Il est important de noter que l'ordre de visite des éléments de R a une très grande influence non seulement sur la qualité mais aussi et surtout sur la rapidité d'exécution de la procédure. Pour optimiser le parcours des combinaisons de R, nous les classons de façon à visiter en premier celles dont l'utilité de production est la moins évidente. Le classement que nous utilisons est intrinsèquement lié à l'heuristique qui nous a permis de construire la solution réalisable initiale. En effet, dans toutes les heuristiques que nous avons développées dans les sections précédentes, autant les premières combinaisons choisies sont d'excellentes candidates pour être produites, autant les dernières peuvent susciter quelques réserves. Ainsi, les ordres retenus sont respectivement :

- quelconque pour l'heuristique gloutonne naïve
- les nombres de participation croissants à toutes les solutions réalisables trouvées jusqu'à présent, pour les autres heuristiques gloutonnes sauf celles des fréquences et des fixations
- les nombres de participation croissants à des solutions lagrangiennes ayant amélioré GLB, pour l'heuristique des fréquences
- les valeurs croissantes de $best\mathtt{LB}^0_{ii} - best\mathtt{LB}^1_{ii}$, pour l'heuristique des fixations

Cependant, comme nous le verrons à la section 4.3, nous appelons très souvent cette procédure, et malgré l'optimisation que nous venons de citer, elle reste l'une des dépenses majeures en temps de calcul. Il apparaît donc nécessaire d'élaborer une nouvelle procédure d'amélioration, ou au moins d'apporter quelques modifications aux voisinages que nous utilisons, afin de rendre ce 2-OPT beaucoup plus performant. C'est le sujet de la section suivante.

4.2.2 2-OPT BRIDÉ

La procédure `2-OPT` que nous avons décrite à la section précédente est très performante, mais peut se révéler particulièrement lente. Nous détaillons dans cette section une nouvelle procédure `2-OPT` que nous qualifierons de *bridée*.

La lenteur observée de l'algorithme 5 résulte principalement du nombre d'échanges effectués, mais aussi et surtout du nombre d'échanges testés. En effet, si l'on veut exclure une combinaison de R, nous devons évaluer le gain que l'on pourrait tirer de son remplacement par l'une de celles qui appartient à $V \setminus R$. Or, très souvent, beaucoup de ces dernières n'ont aucune chance d'améliorer la solution courante.

L'idée que nous développons ici consiste simplement à restreindre les voisinages utilisés, c'est-à-dire à restreindre l'ensemble des combinaisons pouvant sortir de R, et l'ensemble des combinaisons pouvant y entrer. En d'autres termes, nous définissons deux ensembles $P \subseteq R$ et $Q \subseteq V \setminus R$ comme étant, pour le premier, les pires combinaisons produites, et pour le second, les meilleures combinaisons non produites. Puis, nous réutilisons l'algorithme 5 de la façon suivante :

Algorithme 6 Procédure `2-OPT` bridée

$P := \{i | i$ est produite et de qualité relativement mauvaise$\}$
$Q := \{i | i$ n'est pas produite mais de qualité relativement bonne$\}$
répéter
  pour tout élément i de P, parcouru dans l'ordre croissant des qualités **faire**
    parcourir totalement l'ensemble Q,
    soit $j \in Q$ l'élément de Q dont l'échange avec i est le plus profitable
    si l'échange (i, j) améliore la solution **alors**
      $P \leftarrow P + \{j\} \setminus \{i\}$
      $Q \leftarrow Q + \{i\} \setminus \{j\}$
    fin si
  fin pour
jusqu'à ce que aucun échange $(i, j) \in P \times Q$ n'améliore plus la solution

Pour construire les ensembles P et Q, nous classons dans une première étape les éléments de R et de $V \setminus R$ afin ne retenir que ceux qui sont respectivement en tête dans chaque classement.

Nous avons déjà défini à la section précédente quels étaient les classements que nous utilisons pour ranger R. Il parait tout à fait logique d'utiliser le critère

de classement inverse pour ranger $V \setminus R$. Ainsi les combinaisons qui se trouvent en tête de ces deux classements sont bien respectivement les pires combinaisons parmi celles qui sont produites (ayant des qualités minimales), et les meilleures combinaisons parmi celles qui ne sont pas produites (ayant des qualités maximales).

Il ne nous reste plus qu'à déterminer la cardinalité des ensembles P et Q. Ce paramètre est certainement la clé de voûte de cette procédure, car plus ces cardinalités sont grandes, meilleure sera la qualité de la solution finale, mais en contre partie, plus lent sera l'algorithme. Les tests numériques nous ont montré que choisir des cardinalités de 50 nous fournissait suffisamment d'échanges pour espérer obtenir une solution finale de bonne qualité, et surtout suffisamment peu pour être très rapide et fréquemment appelée. Nous avons testé des cardinalités fonctions de la taille de l'instance, mais les résultats étaient sensiblement identiques, et pour les grandes instances, les temps d'exécution beaucoup plus importants.

Dans la section suivante nous allons étudier comment et quand optimiser ces appels.

4.3 APPEL DES HEURISTIQUES

Vu leur nombre, il n'est pas concevable d'appeler à chaque itération de l'algorithme des relaxations lagrangiennes chacune des heuristiques développées dans ce chapitre. Cette section trace la liste des avantages et des inconvénients de chacune d'elle, afin de savoir quand les appeler, et sous quelles conditions.

Dans la littérature, il est recommandé de chercher des solutions réalisables à chaque itération de l'algorithme lagrangien. Nous avons opté pour une utilisation des heuristiques beaucoup plus réduite afin de ne pas ralentir notre programme. En contre partie, nous faisons suivre systématiquement chaque heuristique par une procédure `2-OPT`. Nous recourons à celle-ci toujours sous sa version bridée (voir 4.2.2) sauf si la solution obtenue est la meilleure rencontrée jusqu'alors.

L'utilisation "réduite" des heuristiques consiste à ne faire appel à elles que lorsque la solution lagrangienne courante est la meilleure rencontrée jusqu'à présent, ou lorsque nous mettons à jour le paramètre π (voir chapitre 2).

De part leur nature, il n'existe a priori pas de critère objectif qui nous permette de décider qu'une heuristique donnera une meilleure solution qu'une autre. Cependant, pour nous aider dans notre choix, nous pouvons remarquer que

1. le glouton naïf fournira presque toujours la même solution (il est sûr que cette solution ne variera pas entre deux fixations réussies)

2. les heuristiques des fréquences et des fixations se démarquent des autres par leur rapidité, et ceci est d'autant plus vrai que le nombre de combinaisons à produire k est grand
3. l'heuristique des fixations est surtout efficace quand la différence entre les bornes inférieures $best\texttt{LB}_{ii}^0$ et $best\texttt{LB}_{ii}^1$ est significative

Nous avons donc choisi d'utiliser ces différentes heuristiques suivant les règles suivantes :

- pour obtenir une première solution réalisable : l'heuristique gloutonne naïve
- lorsque la solution lagrangienne courante est la meilleure rencontrée jusqu'à présent :
  1. après de nombreuses fixations réussies ou après un appel aux fixations fortes : l'heuristique des fixations,
  2. après un échec des fixations : une des heuristiques gloutonnes des 2 ou 3 présences, ou avec exclusion, choisie par roulement
  3. systématiquement : l'heuristique des fréquences
- si π est mis à jour : toutes les heuristiques sauf la naïve, et celle des fréquences

Il peut paraître quelque peu dangereux de limiter l'emploi de la procédure `2-OPT` non bridée aux seules meilleures solutions atteintes. Cependant, son coût prohibitif en temps de calcul nous freine considérablement, et il s'est avéré plus judicieux de tester beaucoup de `2-OPT` bridées qu'une seule non bridée de temps en temps. Néanmoins, pour "confirmer" que la meilleure solution réalisable connue est, si ce n'est optimale, tout au moins d'excellente qualité, il est possible de l'utiliser à des phases cruciales de notre algorithme (voir chapitre 2), telles qu'à la fin de la phase du sous gradient, ou après un faisceau ayant convergé. Ainsi, nous recourrons à elle uniquement si nous n'avons pas réussi à prouver, avant, l'optimalité de la solution. Cet emploi étant exceptionnel, le choix de l'heuristique qui précédera ce `2-OPT` complet est éminemment délicat. Malheureusement, ni la théorie ni la pratique ne nous ont permis de désigner la meilleure candidate. Nous avons donc opté pour celle qui nous a traditionnellement donné les meilleurs résultats : l'heuristique des fréquences. Cet appel est bien sûr annulé si la meilleure solution réalisable connue est issue de cette heuristique, et si la meilleure solution lagrangienne est restée inchangée depuis, car alors un `2-OPT` complet a déjà été effectué sur la solution qu'elle pourrait fournir.

Dans tout ce chapitre, nous avons exploité la fixation des combinaisons, dans le sens que nous produisions systématiquement celles qui étaient fixées à 1, et que nous bannissions toujours de R celles qui étaient fixées à 0. Cependant, nous n'avons jamais explicitement utilisé la fixation des arcs. Peut-on tirer profit de telles fixations ? La section suivante tente de répondre à cette question.

4.4 FIXATION D'ARCS : INCONVÉNIENT OU AVANTAGE ?

Nous avons vu au chapitre 3 qu'il était possible de fixer non seulement des sommets, mais aussi des arcs. Jusqu'à présent nous avons toujours exploité la fixations des combinaisons en produisant toujours celle qui étaient fixées à 1 et jamais celles qui étaient fixées à 0. Nous n'avons pas encore utilisé le renseignement que nous fournit la fixation d'un arc ij, à savoir que i doit toujours être substituée par j si cette fixation est à 1, et jamais si elle est à 0. Est-il intéressant d'en tenir compte ?

D'un côté, il peut paraître étrange de s'opposer à la substitution d'une combinaison i par une combinaison j si celle-ci est la moins chère de toutes les combinaisons produites pouvant remplacer i. Il est clair qu'en procédant ainsi nous n'obtiendrons pas la meilleure solution possible, tout au moins au moment où se produit ce phénomène.

D'un autre côté, limiter le nombre de bonnes solutions réalisables accessibles aura pour conséquence d'affiner le choix des combinaisons que l'on va produire dans les heuristiques gloutonnes, ainsi que de limiter le nombre d'échanges dans les procédures `2-OPT`. La principale conséquence sera l'amélioration sensible de la rapidité de ces algorithmes.

Nous avons testés ces deux optiques, et il s'est avéré que les qualités des solutions obtenues étaient équivalentes. Nous avons donc privilégié l'exploitation des fixations d'arcs. Cependant il est important de noter que la fixation à 0 des arcs reliants les combinaisons non fixées à 1 à celles qui le sont, peut générer quelques problèmes. En effet, elle peut rendre les solutions obtenues non réalisables, des combinaisons non produites ne pouvant plus être remplacées. Nous avons donc décider d'augmenter exagérément le coût des substitutions fixer à 0 sans les interdire réellement, afin d'assurer la réalisabilité de toutes les solutions. Ce nouveau coût est égal à une grande constante, et pénalise le recours à une solution qui ne ferait pas partie des solutions qui nous intéressent. Le principal avantage de cette méthode est d'amener de façon naturelle toutes les heuristiques, `2-OPT` y compris, à procéder en deux phases : d'abord elles cherchent à minimiser le nombre de fixations violées, puis, si ce nombre est nul, elles minimisent le coût réel de la solution. En effet, si le coût des substitutions interdites est suffisamment grand, le coût de toutes les solutions intermédiaires sera quasiment proportionnel au nombre de ces fixations violées, et ce n'est que lorsque toutes les substitutions empruntées seront légales, que le coût calculé reflétera réellement le coût de la production.

4.5 CONCLUSION

Chercher la meilleure solution réalisable possible en des temps raisonnables était l'un de nos buts premiers.

Les heuristiques que nous avons décrites dans ce chapitre ont rempli pleinement cet objectif pour les instances que nous avons testées : comme nous le verrons au chapitre 6, elles nous ont fourni des solutions optimales du problème pour un très grand nombre de ces instances, et lorsqu'elles n'étaient pas optimales, ces solutions demeuraient d'excellente qualité, avec une garantie inférieure à 5 %.

Ce succès est dû non seulement aux qualités intrinsèques de chacune de ces heuristiques, mais aussi et surtout à leur complémentarité et à l'utilisation systématique de procédures rapides d'échanges pour affiner leurs solutions.

Cependant, il serait certainement plus confortable sur le plan théorique et plus efficace sur le plan pratique, de disposer d'une seule heuristique, polyvalente, elle aussi de complexité polynômiale, donnant des solutions de qualité comparable à celle des solutions que nous avons obtenues avec notre collection d'heuristiques, et ce, quelle que soit l'instance testée.

Ceci est a priori utopique, le problème de la diversité étant NP-complet (voir section 1.4). Une issue pourrait consister à restreindre ce problème à un sous cas, en exhibant et en exploitant une structure commune à toutes les instances réelles. Voulant développer des heuristiques génériques pour les adapter facilement à des problèmes de localisation classiques, nous n'avons pas mené à terme cette étude, qui reste donc un problème ouvert.

5

ALGORITHME BRANCH AND CUT

Sommaire

Dans ce chapitre, nous décrivons l'algorithme qui nous permet de trouver une solution optimale du problème de la gestion de la diversité. Cet algorithme n'est appelé, bien sûr, que dans le cas où l'algorithme des relaxations lagrangiennes n'a pas réussi à prouver l'optimalité de la solution qu'il fournit.

L'algorithme que nous utilisons pour résoudre jusqu'à l'optimalité le problème de la gestion de la diversité est de type "*Branch and Cut and Price*", c'est-à-dire qu'il est basé sur une énumération implicite des solutions réalisables du problème, en utilisant à chaque nœud de l'arbre d'énumération les méthodes de génération de contraintes et de variables.

Dans une première section, nous rappelons les principales phases de ces algorithmes "Branch and Cut" ainsi que la terminologie de la théorie polyédrale. Puis, nous consacrons les sections suivantes à détailler les points principaux de notre algorithme, à savoir la génération des contraintes (ce qui implique une étude du polyèdre associé à notre problème), la génération des variables, et les règles de branchement.

5.1 ALGORITHME *Branch and Cut*

La résolution d'un programme linéaire en nombres entiers s'avère généralement être une tâche des plus difficiles, les problèmes qu'ils modélisent appartenant souvent à la classe des problèmes NP-difficiles.

Le programme linéaire en nombres entiers que nous considérons dans cette section sera décrit par

$$(\mathcal{P}) \quad \min\{c^T x \mid Ax \geq b, \;\; x \in \mathbb{N}^n\}$$

avec $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ et $b \in \mathbb{R}^m$

L'une des méthodes les plus connues pour résoudre jusqu'à l'optimalité un programme linéaire en nombres entiers est la méthode dite de *séparation et d'évaluation progressive*, de l'anglais *Branch and Bound*. Ce type d'algorithme consiste à construire un arbre d'énumération implicite des solutions réalisables du problème. Chaque nœud représente un sous ensemble de solutions. À chaque nœud de cet arbre, nous évaluons une borne inférieure du coût de la fonction objectif de toutes les solutions réalisables associées à ce nœud. Deux situations sont alors possibles : soit cette borne inférieure est strictement inférieure au coût de la meilleure solution réalisable connue, soit elle est supérieure ou égale. Dans le premier cas, nous partitionnons l'ensemble des solutions réalisables du nœud courant, en générant plusieurs *nœuds fils*, il s'agit de la phase de "branchement". Dans le second cas, aucune solution réalisable associée au nœud courant, n'est

de meilleure qualité que celle qui est déjà connue, nous pouvons donc *élaguer* ce nœud, c'est-à-dire arrêter la recherche dans sa branche à ce stade.

La génération des nœuds fils se fait à l'aide de règles de branchement spécifiques au problème étudié. La section 5.5 sera consacrée à la règle que nous avons utilisée pour le problème de la gestion de la diversité.

Pour évaluer la borne inférieure, plusieurs solutions sont possibles. Dans un algorithme *Branch and Bound* classique, cette évaluation s'effectue en calculant la valeur optimale de la relaxation linéaire de $(\mathcal{P})$, obtenue en remplaçant la contrainte $x \in \mathbb{N}^n$ par $x \in \mathbb{R}^n$.

Si, en pratique, le nombre de variables et/ou de contraintes est trop important pour résoudre les relaxations linéaires, une première alternative est l'utilisation des relaxations lagrangiennes (voir par exemple [Geo74] et [Bea93b]) si la résolution de celles-ci peut être effectuée efficacement. Malheureusement, la recherche des meilleurs multiplicateurs lagrangiens peut se révéler très coûteux en temps de calcul, et si les relaxations lagrangiennes possèdent la propriété d'intégralité, c'est-à-dire qu'il existe des solutions optimales entières pour ces relaxations, alors le théorème de Geoffrion nous assure que la borne inférieure ainsi obtenue ne sera jamais de meilleure qualité que celle fournie par les relaxations linéaires (voir [Geo74]). Cette propriété aura pour conséquence de produire un arbre d'énumération de grande taille, et donc un algorithme de résolution relativement lent.

Pour palier à ce problème, nous allons utiliser un algorithme de type *Branch and Cut and Price* (que nous appellerons pour alléger les notations simplement *Branch and Cut*). Le principe de ces algorithmes est le sujet de la section suivante. Nous rappellerons ensuite la terminologie de la théorie polyédrale.

5.1.1 PRINCIPE

D'après le résultat de Farkas, Weyl et Minkowsky (voir [Sch86]), nous savons que l'enveloppe convexe des solutions réalisables de $(\mathcal{P})$ peut être décrite par un système fini d'inéquations linéaires, i.e. $A'x \geq b'$, avec $x \in \mathbb{R}^n$. Si le nombre de variables n'est pas trop grand, nous pouvons donc d'abord résoudre un programme linéaire dont les contraintes ne constituent qu'un sous ensemble de petite taille du système $A'x \geq b'$, puis, si la solution optimale de cette relaxation est fractionnaire, nous générons des contraintes oubliées qui sont violées par cette solution et que nous ajoutons à la relaxation. En théorie, nous pourrions réitérer ce processus jusqu'à ce que la solution soit entière, et donc optimale. Malheureusement, Padberg et Rao ([PR82]), et Grötschel, Lovàsz et Shrijver ([GLS81]), ont montré qu'on ne peut pas résoudre en temps polynômial le problème, dit de *séparation*, consistant à chercher les contraintes violées à chaque itération si le problème $(\mathcal{P})$ est $\mathcal{NP}$-difficile, sauf si $\mathcal{P} = \mathcal{NP}$. En pratique, la résolution du problème de séparation

s'effectue à l'aide d'heuristiques, et si aucune contrainte violée n'a pu être générée pour éliminer la solution fractionnaire courante, la phase d'évaluation de la borne inférieure prend fin, et la phase de branchement commence.

De tels algorithmes *Branch and Bound*, où l'évaluation s'effectue par séparation, furent exploités pour la première fois par Grötschel, Jünger et Reinelt ([GJR84]). Cet algorithme fut dénommé *Branch and Cut* par Padberg et Rinaldi ([PR87],[PR91]) qui l'utilisèrent pour résoudre le problème du voyageur de commerce.

Si la résolution du programme linéaire est impraticable du fait du nombre de variables, alors que le nombre de contraintes est raisonnable, une éventualité consiste à optimiser la relaxation linéaire en utilisant la méthode dite *de génération de colonnes* introduite par P.C. Gilmore et R.E. Gomory ([GG61]). Elle revient à ne considérer qu'un sous ensemble de variables suffisamment restreint pour rendre possible la résolution de la relaxation linéaire. Puis, grâce à la solution optimale du programme dual associé à cette relaxation restreinte, nous calculons le coût réduit des variables "oubliées", et nous complétons la relaxation linéaire en générant les variables dont le coût réduit est négatif. Un algorithme *Branch-and-Bound* où l'évaluation s'effectue grâce à la génération de colonnes est dit *Branch and Price* (voir pour plus de détails [BJN$^+$94], [VW94] et [Bri95]). Il fut exploité pour résoudre efficacement de nombreux problèmes tels que le problème de tournées de véhicules avec fenêtres de temps ([DDS92], [DSD84]), le problème de découpe binaire de rouleaux ([VBJN92]), ou le problème d'affectation généralisée ([Sav93]).

Ces deux variantes du *Branch and Bound*, à savoir le *Branch and Cut* et le *Branch and Price*, peuvent être conjugués pour résoudre des programmes linéaires en nombres entiers possédant non seulement beaucoup de contraintes, mais aussi beaucoup de variables. Nous parlons alors d'algorithmes *Branch and Cut and Price*. La figure 5.1 illustre les principales phases de ce type d'algorithme. S. Thienel et M. Jünger ont développé un progiciel écrit en C++, nommé Abacus ([Thi95],[JT96],[JT97]), rendant très aisée l'utilisation des algorithmes *Branch and Cut and Price* pour résoudre les problèmes d'optimisation. Nous avons adopté ce progiciel pour le problème de la gestion optimale de la diversité.

Utiliser un algorithme *Branch and Cut and Price* consiste notamment à identifier quelles sont les premières variables et les premières contraintes que nous prenons en compte au nœud racine, et à redéfinir comment nous générons les variables, comment nous générons les colonnes et comment nous branchons. Tous ces points seront les sujets des sections de ce chapitre. Mais avant de les détailler, nous rappelons quelques notions de théorie polyédrale nécessaires à la phase de séparation.

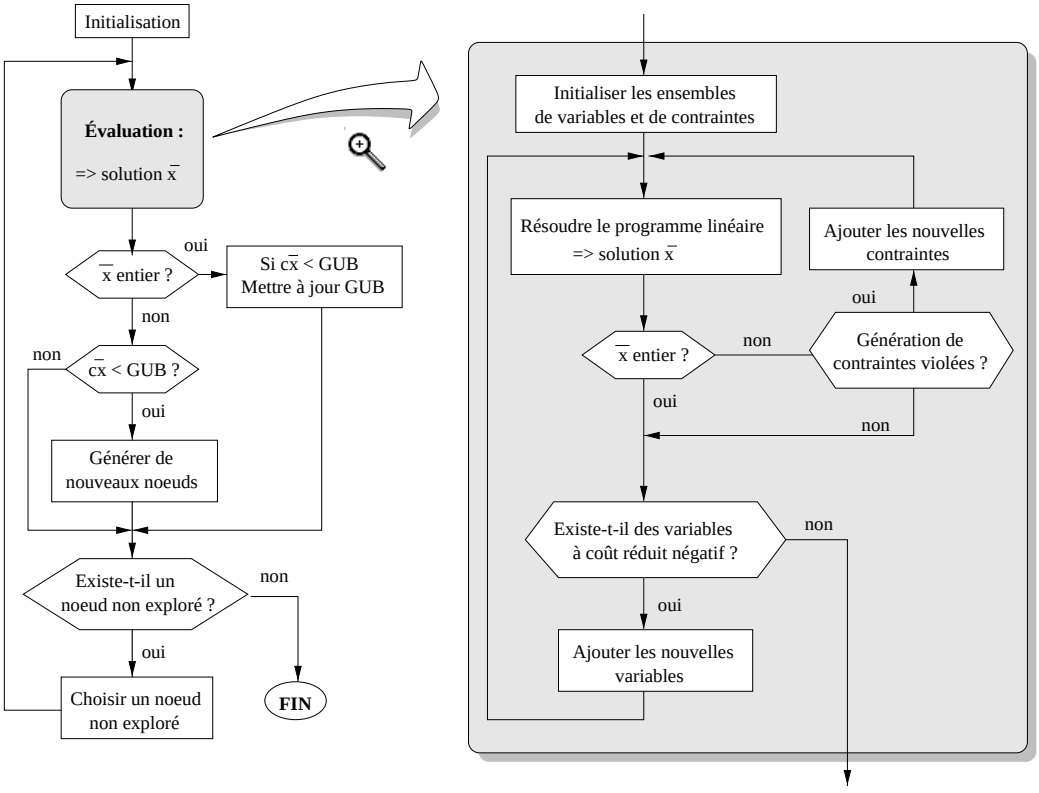


FIG. 5.1 – *Algorithme Branch and Cut and Price*

5.1.2 TERMINOLOGIE DE LA THÉORIE POLYÉDRALE

Cette section est consacrée à quelques rappels de base de la théorie polyédrale. Pour plus de détails, le lecteur est invité à consulter l'ouvrage de Schrijver ([Sch86]), ainsi que celui de Nemhauser et Wolsey ([NW88b]).

Définition 5.1.1 : indépendance affine

Un ensemble $\{x^i \mid 1 \leq i \leq m\}$ *d'éléments de* $\mathbb{R}^n$ *est dit affinement indépendant si*

$$\sum_{i=1}^{m} \lambda_i x^i = 0 \ \textit{et} \ \sum_{i=1}^{m} \lambda_i = 0 \quad \textit{impliquent} \quad \lambda_i = 0, \ \ \forall i = 1, ..., m$$

Définition 5.1.2 : indépendance linéaire

Un ensemble $\{x^i \mid 1 \leq i \leq m\}$ *d'éléments de* $\mathbb{R}^n$ *est dit linéairement indépendant si*

$$\sum_{i=1}^{m} \lambda_i x^i = 0 \quad \textit{implique} \quad \lambda_i = 0, \ \ \forall i = 1, ..., m$$

Définition 5.1.3 : enveloppe convexe

L'enveloppe convexe d'un ensemble $\{x^i \mid 1 \leq i \leq m\}$ *d'éléments de* $\mathbb{R}^n$ *est l'ensemble*

$$\left\{ \sum_{i=1}^{m} \lambda_i x^i \mid \sum_{i=1}^{m} \lambda_i = 1 \quad \textit{et} \quad \lambda_i \in \mathbb{R}^+, \ \ \forall i = 1, ..., m \right\}$$

Théorème 5.1.4 : équivalence des indépendances affines et linéaires

Soit $S = \{x^0, x^1, ..., x^m\}$ *un ensemble d'éléments de* $\mathbb{R}^n$.
Si le vecteur nul de $\mathbb{R}^n$ *n'appartient pas à* S*, alors les deux propositions suivantes sont équivalentes :*

a) les éléments de S *sont affinement indépendants*

b) les éléments de S *sont linéairement indépendants*

Définition 5.1.5 : Dimension

La dimension d'un ensemble P *de* $\mathbb{R}^n$*, notée* $dim(P)$ *est le nombre maximum d'éléments de* P *affinement indépendant diminué d'une unité. Si* $dim(P) = n$*, l'ensemble* P *est dit de pleine dimension*

Définition 5.1.6 : Polyèdre et polytope

Un polyèdre est un ensemble $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$*, avec* A *une* $m \times n$*-matrice à coefficients réels, et* $b \in \mathbb{R}^m$.
Un polyèdre est appelé polytope s'il est borné.

Définition 5.1.7 : hyperplan

Un hyperplan de $\mathbb{R}^n$ est un ensemble d'éléments x de $\mathbb{R}^n$ vérifiant $ax = b$.

Définition 5.1.8 : inéquation valide - violée - serrée

Une inéquation $fx \geq f_0$ est dite valide pour un polyèdre P si, pour tout $x' \in P$, on a $fx' \geq f_0$.
Une inéquation $fx \geq f_0$ est dite violée par un élément $x' \in \mathbb{R}^n$ si $fx' < f_0$.
Une inéquation $fx \geq f_0$ est dite serrée par un élément $x' \in \mathbb{R}^n$ si $fx' = f_0$.

Définition 5.1.9 : face

Si une inéquation $fx \geq f_0$ est valide pour un polyèdre P, alors l'ensemble $F = \{x \in P \mid fx = f_0\}$ est appelée face de P.
Si $F \neq \emptyset$ et $F \neq P$, alors la face F est dite propre.

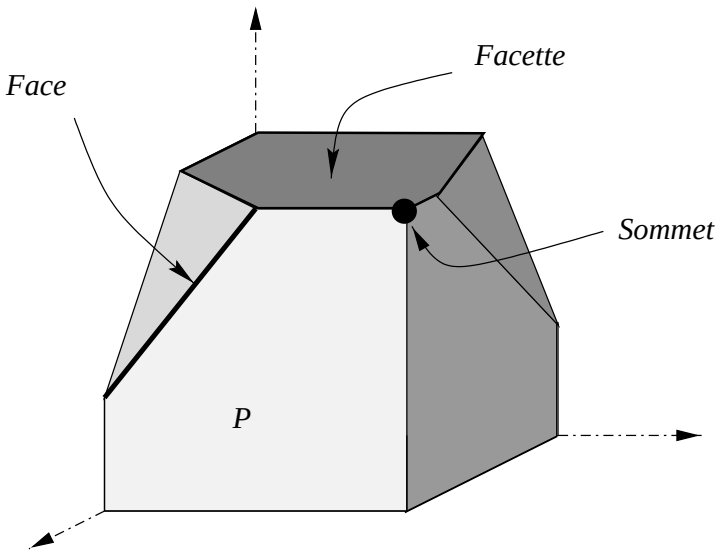


FIG. 5.2 – *Notions de théorie polyédrale*

Définition 5.1.10 : facette

Une face F d'un polyèdre P est dite facette de P si $dim(F) = dim(P) - 1$.
Par abus de langage, la contrainte valide $fx \geq f_0$ définissant F sera elle-aussi appelée facette de P.

Définition 5.1.11 : sommet d'un polyèdre

Si une face d'un polyèdre P est de dimension nulle, c'est-à-dire si elle est réduite à un élément $x \in P$, alors x est appelé sommet de P.

5.2 INITIALISATIONS AU NOEUD RACINE

Comme pour tout algorithme d'énumération implicite, la phase d'initialisation d'un algorithme *Branch and Cut* a une très grande influence sur le nombre de nœuds explorés et sur sa rapidité. Les paramètres que nous devons initialiser sont d'abord l'ensemble des variables et des contraintes qui définissent le premier programme linéaire que nous devons résoudre, puis la valeur d'une borne supérieure qui nous permettra d'élaguer certaines branches, c'est-à-dire de ne pas générer de nouveaux fils.

Comme nous l'avons vu au chapitre 2, les dernières phases de l'algorithme lagrangien que nous utilisons consiste à faire appel à la méthode des faisceaux. Non seulement, cette méthode nous offre une garantie quant à la qualité des multiplicateurs lagrangiens ainsi obtenus, mais encore elle nous fournit une solution de la relaxation linéaire, comme étant une combinaison convexe des solutions lagrangiennes. Nous initialisons donc l'ensemble des variables comme étant celles qui participaient à la définition de ces dernières solutions lagrangiennes. Malheureusement, en pratique, ces variables ne suffiront pas à trouver une solution optimale de base de la relaxation linéaire, nous devrons donc plus tard générer d'autres variables. Nous omettons bien sûr toutes les variables qui ont été précédemment fixées à 0 ou à 1.

L'ensemble des contraintes initiales est défini par

contrainte (a) : $$\sum_{j \in V \setminus (V^0 \cup V^1)} x_{jj} = k - |V^1|$$

contrainte (b) : $\sum_{j \succeq i} x_{ij} = 1$ pour toute combinaison $i \in V$ non fixée à 1

contrainte (c) : $x_{ij} \leq x_{jj}$ pour toute variable x_{ij} appartenant à l'ensemble de variables initiales telles que x_{jj} n'est pas fixée à 1

contrainte (d) : $\sum_{i \prec j} x_{ij} \leq M x_{jj}$ pour toute combinaison $i \in V$ non fixée à 1

Dans les contraintes (b) nous ne prenons en considération que les variables non fixées. Les contraintes (d) sont a priori redondantes lorsque nous considérons la relaxation linéaire complète, cependant nous les ajoutons afin d'accélérer notre algorithme. La constante M est choisie arbitrairement, nous demandons seulement qu'elle soit suffisamment grande pour être valide pour toutes les solutions réalisables du problème, par exemple égale au nombre total de combinaisons.

Après avoir résolu ce programme linéaire, nous vérifions si la solution est entière. Si c'est le cas, nous devons vérifier si des variables "oubliées" ont un coût réduit négatif, car sinon nous ne pouvons pas conclure à l'optimalité de la solution, ce sera le sujet de la section 5.4. Si la solution n'est pas entière, nous allons tenter de l'éliminer en générant de nouvelles contraintes. La section suivante est consacrée à la génération de ces nouvelles contraintes.

5.3 GÉNÉRATION DE CONTRAINTES ET ÉTUDE POLYÉDRALE

Nous avons vu au chapitre 1, section 1.5, différentes modélisations possibles du problème de la gestion de la diversité. Nous avons retenu dans tout ce document celle qui est décrite par le programme ($\mathcal{P}_k$). Nous noterons P_k l'enveloppe convexe des solutions réalisables de ($\mathcal{P}_k$). Le polyèdre P_k est donc l'enveloppe convexe des éléments x de $\{0,1\}^{|V|+|A|}$ vérifiant

$$\sum_{j \succeq i} x_{ij} = 1 \qquad \forall i \in V \tag{5.1}$$

$$\sum_{j \in V} x_{jj} = k \tag{5.2}$$

$$x_{ij} \leq x_{jj} \qquad \forall i,j \in V,\ i \prec j \tag{5.3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V,\ i \preceq j \tag{5.4}$$

Pour résoudre le problème de la diversité, nous cherchons un élément de ce polyèdre qui minimise la fonction économique :

$$\sum_{j \in V} c_j d_j x_{jj} \; + \; \sum_{i \in V} d_i \sum_{j \succ i} c_j x_{ij}$$

L'étude de ce polyèdre est particulièrement délicate. Nous exploiterons donc aussi le polyèdre P_K, enveloppe convexe des solutions x de $\{0,1\}^{|V|+|A|}$ vérifiant les contraintes (5.1), (5.3) et (5.4). Le polyèdre P_k étant la "section" de P_K suivant l'hyperplan $\sum_{j \in V} x_{jj} = k$ (voir figure 5.3), toute contrainte valide pour P_K est aussi valide pour P_k.

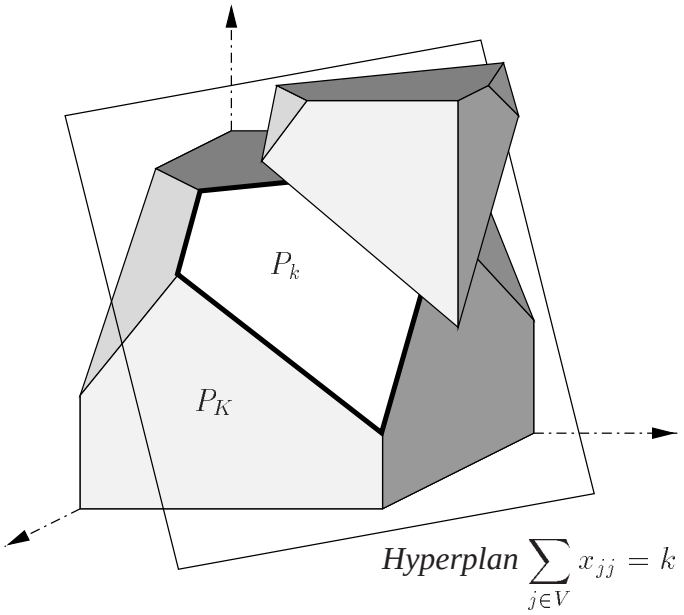


FIG. 5.3 – *P_k vu comme une section de P_K*

Dans une première section, nous allons étudier les dimensions de ces deux polytopes, puis, dans une seconde, nous nous intéressons à quelques facettes de P_k.

5.3.1 DIMENSION DES POLYTOPES P_k ET P_K

Pour pouvoir démontrer que des contraintes sont facettes des polytopes P_k et P_K, nous devons connaître leur dimension. Nous rappelons que T est l'ensemble des combinaisons terminales.

Théorème 5.3.1

Si $|T| < k < |V|$ *alors dim(*P_k*)* = $|A| - 1$

Preuve de 5.3.1 : Pour prouver ce théorème, nous montrons d'abord que tout élément de P_k vérifie $|V| + 1$ équations linéairement indépendantes. Puis nous exhibons $|A|$ éléments linéairement indépendants.

Lemme 5.3.2

Les équations (5.1) et (5.2) sont linéairement indépendantes.

Preuve de 5.3.2 : Les équations (5.1) associées aux combinaisons terminales i de T s'écrivent $x_{ii} = 1$. Nous pouvons donc réécrire (5.2) par l'équation (5.2') suivante $\sum_{j \in V \setminus T} x_{jj} = k - |T|$.

Remarquons que :

1. Dans toute équation (5.1) associée à un élément i non terminal (i.e. $i \in V \setminus T$), il existe au moins une variable x_{ij} associée à un arc $ij \in A$. Cette variable est absente des autres équations (5.1) et de (5.2').
2. L'équation (5.1) associée à un élément $i \in T$ est la seule à posséder la variable x_{ii}.

Nous en déduisons que les équations (5.1) et (5.2') sont linéairement indépendantes, et donc que les équations (5.1) et (5.2) le sont aussi. ♢

Sachant que le vecteur nul de $\mathbb{R}^{|A|+|V|}$ n'est pas solution de P_k, pour prouver le théorème 5.3.1, il suffit de construire $|A|$ solutions linéairement indépendantes. Pour cela, notons R' un ensemble de k éléments de V contenant l'ensemble T des combinaisons terminales. Nous construisons une première solution réalisable x' de P_k, en produisant toutes les combinaisons de R' et en remplaçant chaque combinaison $u \in V \setminus R'$ par une combinaison terminale $t_u \in T$. Nous savons que de telles substitutions sont possibles grâce à la transitivité de l'ordre partiel $\succ$.

Nous définissons donc x' de la façon suivante :

Pour tout $u \in R'$, $x'_{uu} = 1$

Pour tout $u \in V \setminus R'$, $x'_{uv} = \begin{cases} 1 & \text{si } v = t_u \\ 0 & \text{sinon} \end{cases}$

Le premier schema de la figure 5.4 illustre une telle solution.

Pour définir les autres solutions réalisables, nous allons faire jouer un rôle particulier à un arc bt avec $b \in R' \setminus T$ et $t \in T$. L'existence de bt est assurée par le fait que $|R'| = k$ et $|T| < k < |V|$.

Pour tout élément j de $V \setminus R'$, nous définissons des solutions z^j par :

$$\text{pour tout } u \in V \quad z^j_{uu} = \begin{cases} 1 & \text{si } u = j \\ 0 & \text{si } u = b \\ x'_{uu} & \text{sinon} \end{cases}$$

$$\text{pour tout } uv \in A \quad z^j_{uv} = \begin{cases} 0 & \text{si } uv = jt_j \\ 1 & \text{si } uv = bt \\ x'_{uv} & \text{sinon} \end{cases}$$

Excepté pour l'arc bt, nous définissons une solution y^{ij} de P_k pour chaque arc $ij \in A \setminus A'$, $ij \neq bt$. Nous procédons de la manière suivante (voir la figure 5.4) :

– si $i \in V \setminus R'$ et $j \in R'$, posons

$$\text{pour tout } u \in V \quad y^{ij}_{uu} = x'_{uu}$$

$$\text{pour tout } uv \in A \quad y^{ij}_{uv} = \begin{cases} 1 & \text{si } uv = ij \\ 0 & \text{si } uv = it_i \\ x'_{uv} & \text{sinon} \end{cases}$$

– si $i, j \in R'$, soit $a \in V \setminus R'$ quelconque, et posons

$$\text{pour tout } u \in V \quad y^{ij}_{uu} = \begin{cases} 0 & \text{si } u = i \\ 1 & \text{si } u = a \\ x'_{uu} & \text{sinon} \end{cases}$$

$$\text{pour tout } uv \in A \quad y^{ij}_{uv} = \begin{cases} 1 & \text{si } uv = ij \\ 0 & \text{si } uv = at_a \\ x'_{uv} & \text{sinon} \end{cases}$$

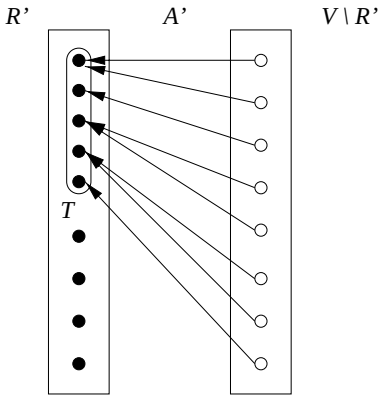
– si $i, j \in V \setminus R'$, posons

$$\text{pour tout } u \in V \quad y^{ij}_{uu} = \begin{cases} 1 & \text{si } u = j \\ 0 & \text{si } u = b \\ x'_{uu} & \text{sinon} \end{cases}$$

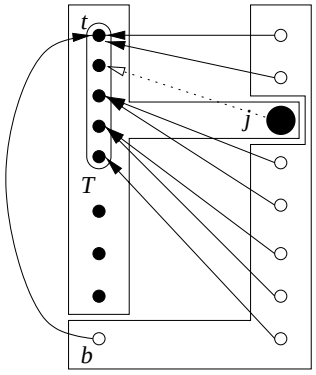
$$\text{pour tout } uv \in A \quad y^{ij}_{uv} = \begin{cases} 1 & \text{si } uv \in \{ij, bt\} \\ 0 & \text{si } uv \in \{it_i, jt_j\} \\ x'_{uv} & \text{sinon} \end{cases}$$

– si $i \in R'$, et $j \in V \setminus R'$, posons

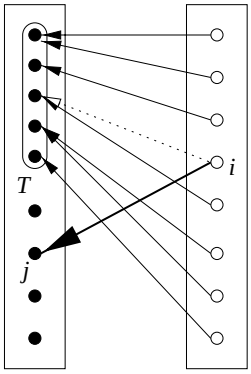
$$\text{pour tout } u \in V \quad y^{ij}_{uu} = \begin{cases} 1 & \text{si } u = j \\ 0 & \text{si } u = i \\ x'_{uu} & \text{sinon} \end{cases}$$



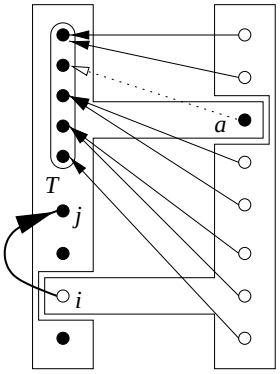
Solution x'



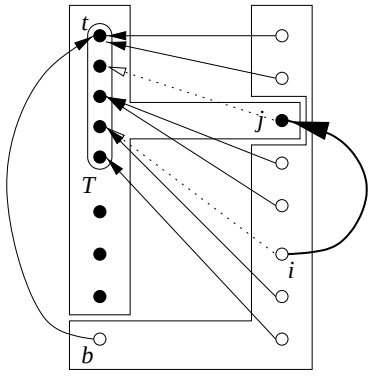
Solutions z^j *pour* $j \in V \setminus R'$



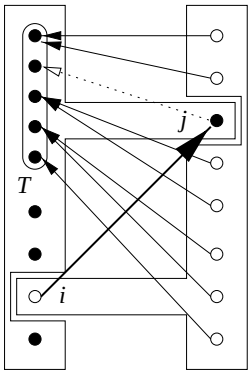
Solutions y^{ij} *pour* $i \in V \setminus R'$ *et* $j \in R'$



Solutions y^{ij} *pour* $i, j \in R'$



Solutions y^{ij} *pour* $i, j \in V \setminus R'$



Solutions y^{ij} *pour* $i \in R'$ *et* $j \in V \setminus R'$

FIG. 5.4 – *Solutions linéairement indépendantes de* P_k

$$\text{pour tout } uv \in A \quad y^{ij}_{uv} = \begin{cases} 1 & \text{si } uv = ij \\ 0 & \text{si } uv = jt_j \\ x'_{uv} & \text{sinon} \end{cases}$$

Lemme 5.3.3

x', y^{ij} et z^j sont des solutions distinctes de P_k et sont au nombre de $|A|$.

Preuve de 5.3.3 : Il est clair que se sont des solutions de P_k.

Le fait qu'elles sont distinctes découle de leur construction. N'ayant pas défini de solution y^{ij} pour $ij = bt$, la solution y^{ij} est la seule solution dont la composante associée à ij vaut 1. De plus, pour prouver qu'une solution z^j est distincte des autres $z^{j'}$ et de x', il suffit de remarquer qu'elle est la seule dans laquelle nous produisons la combinaison j.

En comptabilisant toutes ces solutions, on obtient un nombre de solutions distinctes égal à $|A \setminus (A' \cup \{bt\})| + |A'| + 1 = |A|$ ♢

Lemme 5.3.4

Les $|A|$ solutions x', y^{ij} et z^j sont linéairement indépendantes

Preuve de 5.3.4 : Démontrons que l'équation

$$\alpha x' + \sum_{ij \in A \setminus (A' \cup \{bt\})} \beta_{ij} y^{ij} + \sum_{j \in V \setminus R'} \gamma_j z^j = 0$$

implique

$$\alpha = 0, \qquad \beta_{ij} = 0 \quad \forall ij \in A \setminus (A' \cup \{bt\}) \qquad \text{et} \qquad \gamma_j = 0 \quad \forall j \in V \setminus R'$$

pour tout α, β_{ij} et γ_j réels.

En remarquant que pour tous les arcs $ij \in A \setminus (A' \cup \{bt\})$, la variable qui lui est associée est nulle dans toutes les solutions proposées, excepté dans la solution y^{ij} à laquelle elle correspond, on en déduit que tous les β_{ij} sont nuls.

Il ne reste donc plus qu'à prouver l'implication

$$\alpha x' + \sum_{j \in V \setminus R'} \gamma_j z^j = 0 \quad \Rightarrow \quad \alpha = 0, \qquad \text{et} \qquad \gamma_j = 0 \quad \forall j \in V \setminus R'$$

Ceci se démontre en notant que, pour tout $j \in V \setminus R'$, la variable qui lui est associée est nulle dans x' et dans toutes les solutions $z^{j'}$ sauf si $j' = j$. Ceci

implique que tous les γ_j sont nuls, et donc que α est nul. $\diamondsuit$

Conclusion : d'après le lemme 5.3.2, on a $\dim(P_k) \leq |A| - 1$, et d'après le lemme 5.3.4, on a $\dim(P_k) \geq |A| - 1$. Donc $\dim(P_k) = |A| - 1$, ce qui prouve le théorème 5.3.1

$\diamondsuit$

Théorème 5.3.5

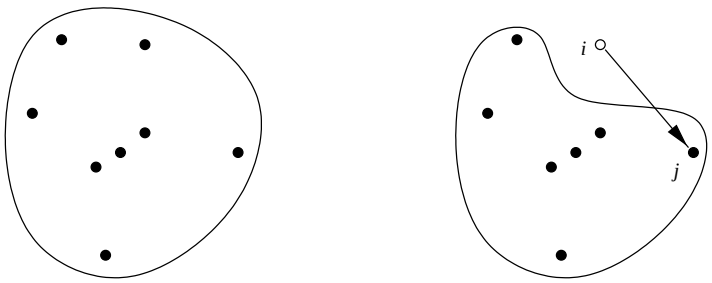
dim$(P_K) = |A|$

Preuve de 5.3.5 : D'après le lemme 5.3.2, nous savons que les contraintes (5.1) sont linéairement indépendantes.

Sachant que le vecteur nul de $\mathbb{R}^{|A|+|V|}$ n'est pas solution de P_K, il ne nous reste plus qu'à exhiber $|A| + 1$ solutions de P_K linéairement indépendantes pour prouver ce théorème.

Pour cela, nous définissons $|A| + 1$ solutions réalisables de la façon suivante :

- x^0 est la solution consistant à produire toutes les combinaisons, c'est-à-dire
  $x^0_{uv} = \begin{cases} 1 & \text{si } u = v \\ 0 & \text{sinon} \end{cases}$
- pour tout $ij \in A$, soit y^{ij} la solution consistant à produire toutes les combinaisons, excepté i qui est remplacée par j. Nous avons donc,
  $y^{ij}_{uv} = \begin{cases} 1 & \text{si } u = v \text{ et } u \neq i \\ 1 & \text{si } uv = ij \\ 0 & \text{sinon} \end{cases}$



Solution x^0 *Solutions* y^{ij}

FIG. 5.5 – *Solutions linéairement indépendantes de* P_K

Par construction, il est clair que ces solutions sont distinctes. Leur indépendance linéaire se démontre en remarquant que, pour chacune d'elles, la composante correspondant à un arc $ij \in A$ vaut 0, excepté pour la solution y^{ij} pour laquelle elle vaut 1. ♢

5.3.2 FACETTES TRIVIALES

Théorème 5.3.6

Pour tout arc $ij \in A$, la contrainte $x_{ij} \geq 0$ est une facette de P_k

Preuve de 5.3.6 : pour démontrer ce théorème, nous reprenons les solutions réalisables utilisées dans la preuve du théorème 5.3.1. Nous supposons que i appartient à l'ensemble R' qui a permis de définir ces solutions, et que l'arc bt est différent de ij.

Nous avons démontré au lemme 5.3.4, que ces $|A|+1$ solutions de P_k étaient linéairement indépendantes.

Il suffit de remarquer que pour toutes ces solutions, excepté y^{ij}, la composante correspondant à ij est nulle. Par conséquent, nous avons $|A|$ solutions linéairement indépendantes de P_k qui serrent la contrainte $x_{ij} \geq 0$. ♢

Théorème 5.3.7

Pour toute combinaison source j, non terminale, la contrainte $x_{jj} \geq 0$ est une facette de P_k

Preuve de 5.3.7 : comme précédemment, nous reprenons les solutions réalisables utilisées dans la preuve du théorème 5.3.1. Nous supposons que l'ensemble R' qui a permis de définir ces solutions ne contient pas la combinaison i, et que le sommet a, qui est utilisé pour définir les solutions $y^{i'j'}$ pour $i', j' \in R'$, est différent de j.

Nous avons démontré au lemme 5.3.4, que ces $|A|+1$ solutions de P_k étaient linéairement indépendantes.

Il suffit de remarquer que dans toutes ces solutions, excepté z^j, la combinaison j n'est pas produite. Par conséquent, nous avons $|A|$ solutions linéairement indépendantes de P_k qui serrent la contrainte $x_{jj} \geq 0$. ♢

Remarque : Il est clair que pour toute combinaison terminale j, la contrainte $x_{jj} \geq 0$ ne peut pas être facette, car dans toute solution réalisable nous avons $x_{jj} = 1$. Si la combinaison j n'est pas une source, c'est-à-dire si l'ensemble

$\{i \in V \mid i \prec j\}$ n'est pas vide, alors la face $x_{jj} \geq 0$ est contenue dans l'intersection des facettes $x_{ij} \geq 0$, car ne pas produire j implique qu'elle ne peut pas remplacer de combinaisons i.

Théorème 5.3.8

Soit $ij \in A$.
Si l'une des propositions suivante est vraie

1. *j est terminale, i est source, et j est la seule combinaison acceptant de remplacer i*
2. *j n'est pas terminale*

Alors la contrainte $x_{ij} \leq x_{jj}$ est une facette de P_k

Preuve de 5.3.8 : A nouveau, nous reprenons les solutions réalisables utilisées dans la preuve du théorème 5.3.1.

cas 1 : j est terminale, i est source, et j est la seule combinaison pouvant remplacer i

Nous supposons ici que i n'appartient pas à l'ensemble R' qui a permis de définir ces solutions, et que le sommet a, qui est utilisé pour définir les solutions $y^{i'j'}$ pour $i', j' \in R'$, est différent de i.

Nous avons démontré au lemme 5.3.4, que ces $|A| + 1$ solutions de P_k étaient linéairement indépendantes.

Il suffit de remarquer que dans toutes ces solutions, excepté z^j, nous avons $x_{ij} = 1 = x_{jj}$.

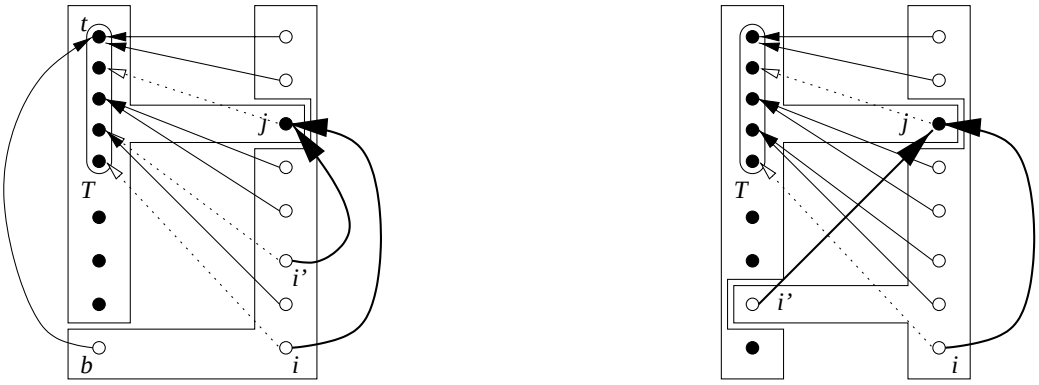
cas 2 : j n'est pas terminale.

Nous supposons que l'ensemble R' qui a permis de définir ces solutions ne contient ni i ni j, et que le sommet a, qui est utilisé pour définir les solutions $y^{i'j'}$ pour $i', j' \in R'$, est différent de j. Cependant, nous redéfinissons les solutions $y^{i'j}$ pour toutes les combinaisons $i' \prec j$ de la façon suivante :

– si $i' \in V \setminus R'$, nous posons

$$\text{pour tout } u \in V \quad y^{i'j}_{uu} = \begin{cases} 1 & \text{si } u = j \\ 0 & \text{si } u = b \\ x'_{uu} & \text{sinon} \end{cases}$$

$$\text{pour tout } uv \in A \quad y^{i'j}_{uv} = \begin{cases} 1 & \text{si } uv \in \{i'j, ij, bt\} \\ 0 & \text{si } uv \in \{i't_{i'}, it_i, jt_j\} \\ x'_{uv} & \text{sinon} \end{cases}$$



Solutions $y^{i'j}$ *pour* $i' \in V \setminus R'$ *Solutions* $y^{i'j}$ *pour* $i' \in R'$

FIG. 5.6 – *Nouvelle définition des solutions* $y^{i'j}$ *pour* $i' \prec j$

– si $i' \in R'$, nous posons

$$\text{pour tout } u \in V \quad y^{i'j}_{uu} = \begin{cases} 1 & \text{si } u = j \\ 0 & \text{si } u = i' \\ x'_{uu} & \text{sinon} \end{cases}$$

$$\text{pour tout } uv \in A \quad y^{i'j}_{uv} = \begin{cases} 1 & \text{si } uv \in \{i'j, ij\} \\ 0 & \text{si } uv \in \{jt_j, it_i\} \\ x'_{uv} & \text{sinon} \end{cases}$$

En utilisant des arguments analogues à ceux qui ont été développés pour prouver les lemmes 5.3.3 et 5.3.4, nous pouvons démontrer que ces solutions de P_k sont distinctes, au nombre de $|A| + 1$, et linéairement indépendantes.
Enfin, il suffit de remarquer que toutes ces solutions, excepté z^j, serrent la contrainte $x_{ij} \leq x_{jj}$.

♢

Remarque : Les hypothèses que nous avons formulées dans ce théorème sont motivées par le fait que si ij ne les vérifient pas alors la contrainte $x_{ij} \leq x_{jj}$ est contenue dans une intersection de facettes. En effet, supposons que j est terminale, ce qui implique $x_{jj} = 1$. Cette contrainte devant être serrée, nous ne nous intéressons qu'aux solutions où $x_{ij} = x_{jj} = 1$. Ces solutions sont donc telles que

– Pour toute combinaison $j' \succ i$, $j' \neq j$, on a $x_{ij'} = 0$

– Pour toute combinaison $\ell \prec i$, on a $x_{\ell i} = 0$

Par conséquent, la face $x_{ij} \leq x_{jj}$ est contenue dans l'intersection des facettes $x_{ij'} \geq 0$ et $x_{\ell i} \geq 0$. D'où, les hypothèses du théorème qui exigeaient qu'il n'existe aucun $j' \succ i$ autre que j, et aucun $\ell \prec i$, si j est une combinaison terminale.

5.3.3 CAS DES CONTRAINTES DE CHEMIN

Nous avons vu au chapitre 3, section 3.5, que les contraintes dites *de chemins* sont valides pour une certaine classe de solutions optimales de $(\mathcal{P}_k)$, et qu'elles sont particulièrement efficaces pour construire de nouveaux critères de fixations de variables. Ces contraintes ont été définies par

Définition 5.3.9

Soient trois combinaisons i, j et ℓ telles que $i \prec j \prec \ell$. Nous appellerons "contrainte de chemin associée à (i, j, ℓ)" la contrainte :

$$x_{i\ell} \leq x_{j\ell}$$

Nous savons qu'elles ne sont peut-être pas valides pour tous les éléments du polyèdre P_k, mais nous avons l'assurance qu'en les ajoutant nous obtiendrions un nouveau polyèdre possédant certains sommets "optimaux" de P_k (c'est-à-dire solutions entières qui sont optimales par rapport à notre fonction économique). A priori, il parait donc avantageux de les ajouter à la relaxation linéaire afin d'augmenter la borne inférieure qu'elle fournira. Cependant, il est important de noter que le nombre de contraintes de chemin est très grand, il y en a autant que de triplets (i, j, ℓ) vérifiant $i \prec j \prec \ell$. L'ajout de telles contraintes n'aura donc un véritable intérêt que si l'augmentation de la borne inférieure est réelle.

Le théorème suivant nous prouve que, du fait de la structure très spéciale de sa fonction économique, ces contraintes n'ont aucune influence sur la valeur de la relaxation linéaire de $(\mathcal{P}_k)$, et donc qu'il est inutile de les ajouter.

Théorème 5.3.10

Il existe au moins une solution optimale de la relaxation linéaire de $(\mathcal{P}_k)$ qui vérifie toutes les contraintes de chemins.

Preuve de 5.3.10 :

Soit $\bar{x}$ une solution optimale de la relaxation linéaire de $(\mathcal{P}_k)$.

Pour prouver ce théorème, nous allons employer la notion de code décimal défini au chapitre 3. Nous rappelons que le code décimal est une fonction injective de l'ensemble des combinaisons V vers $\mathbb{N}$. Ce code nous permet de classer de façon non ambiguë les combinaisons de $V = \{j_1, j_2, ..., j_n\}$ dans l'ordre croissant de leurs coûts, et, en cas d'égalité de ceux-ci, dans l'ordre croissant leurs codes décimaux.

Notons, pour tout $i \in V$, et pour tout $r \in \{1, 2, ..., n\}$, $\Gamma_r^+(i)$ l'ensemble des combinaisons $j \in V$ qui peuvent remplacer i et qui sont classées parmi les r premières combinaisons

$$\Gamma_r^+(i) = \{j_p \in V \mid p \leq r \text{ et } j_p \succeq i\}$$

Remarquer que pour tout $i \in V$ et pour tout $r \in \{1, 2, ..., n\}$, i est le premier élément de $\Gamma_r^+(i)$

Soit $\bar{x}'$, la solution réalisable de la relaxation linéaire de $(\mathcal{P}_k)$ définie par

- Pour tout $j \in V$, $\bar{x}'_{jj} = \bar{x}_{jj}$
- Pour tout $i \in V$, nous répartissons itérativement la partie de sa demande non satisfaite, $1 - \bar{x}'_{ii}$, en privilégiant en premier les substitutions de i par les combinaisons $j \succ i$ les moins chères, et si plusieurs combinaisons j ont le même coût, nous privilégions d'abord celle qui possède le code décimal minimal.

  En d'autres termes, si r_i désigne le plus petit $r \in \{1, 2, ..., n\}$ tel que $\sum_{j \in \Gamma_r^+(i)} \bar{x}'_{jj} \geq 1$, nous notons pour tout $j_r \succeq i$

  1. si $r < r_i$, alors $\bar{x}'_{ij_r} = \bar{x}'_{j_r j_r}$
  2. si $r = r_i$, alors $\bar{x}'_{ij_r} = 1 - \sum_{j \in \Gamma_{r-1}^+(i)} \bar{x}'_{jj}$
  3. si $r > r_i$, alors $\bar{x}'_{ij_r} = 0$

Il est facile de voir que pour tout $i \in V$, r_i est bien défini, car, par réalisabilité de $\bar{x}$, nous avons

$$\sum_{j \succeq i} \bar{x}_{ij} = 1 \leq \sum_{j \succeq i} \bar{x}_{jj} = \sum_{j \succeq i} \bar{x}'_{jj}$$

Il est vient que $\bar{x}'$ est une solution réalisable de la relaxation linéaire de $(\mathcal{P}_k)$.

De plus, grâce à la définition des $\bar{x}'_{ij}$, la valeur de la fonction économique en $\bar{x}'$ est telle que

$$\sum_{j \in V} c_j d_j \bar{x}'_{jj} + \sum_{i \in V} d_i \sum_{j \succ i} c_j \bar{x}'_{ij} \leq \sum_{j \in V} c_j d_j \bar{x}_{jj} + \sum_{i \in V} d_i \sum_{j \succ i} c_j \bar{x}_{ij}$$

Or $\bar{x}$ est optimale, donc $\bar{x}'$ l'est aussi.

Pour prouver que $\bar{x}'$ vérifie toutes les contraintes de chemins, il suffit de remarquer que si $i \prec i'$ alors :

1. $\Gamma_r^+(i') \subset \Gamma_r^+(i)$ pour tout $r \in \{1, 2, ..., n\}$, par transitivité de l'ordre partiel $\succ$
2. $r_i \leq r_{i'}$

ce qui implique que si $j_r \succ i'$, trois cas sont possibles :

- cas 1 : si $r < r_i$, alors

$$\bar{x}'_{ij_r} = \bar{x}'_{j_r j_r} = \bar{x}'_{i'j_r}$$

- cas 2 : si $r = r_i$, alors

$$\bar{x}'_{ij_r} = 1 - \sum_{j \in \Gamma_{r-1}^+(i)} \bar{x}'_{jj} \leq 1 - \sum_{j \in \Gamma_{r-1}^+(i')} \bar{x}'_{jj} \leq \bar{x}'_{i'j_r}$$

- cas 3 : si $r > r_i$, alors

$$\bar{x}'_{ij_r} = 0 \leq \bar{x}'_{i'j_r}$$

♢

Nous allons voir dans la section suivante que nous pouvons exploiter d'autres contraintes valides de P_k afin de couper des solutions fractionnaires.

5.3.4 CONTRAINTES DE CYCLES

Comme nous l'avons vu à la section 1.5, le problème de la diversité, et plus particulièrement la modélisation que nous avons choisie, est un cas particulier de problème de localisation. Nous allons exploiter dans cette section un type de contraintes bien connues du problème de localisation (voir par exemple [CT82], [Gui80], [CJPR83] et [CPR83]).

Avant de rappeler la définition de ces contraintes, dites *de cycles*, nous rappelons celle du polytope P_{loc} du problème de localisation.

Soient I l'ensemble des clients et J l'ensemble des fournisseurs potentiels.

Soit pour tout $i \in I$, et tout $j \in J$,

$$x_{ij} = \begin{cases} 1 & \text{si la demande } i \text{ est satisfaite par } j \\ 0 & \text{sinon} \end{cases}$$

Soit pour tout $j \in J$

$$y_j = \begin{cases} 1 & \text{si } j \text{ est réellement fournisseur} \\ 0 & \text{sinon} \end{cases}$$

Le polytope P_{loc} associé au problème de localisation est l'enveloppe convexe des solutions du système :

$$\begin{cases} \sum_{j \in J} x_{ij} = 1 & \text{pour tout } i \in I \\ x_{ij} \leq y_j & \text{pour tout } i \in I \text{ et pour tout } j \in J \\ x_{ij} \in \{0,1\} & \text{pour tout } i \in I \text{ et pour tout } j \in J \\ y_j \in \{0,1\} & \text{pour tout } j \in J \end{cases}$$

Pour définir une contrainte de cycle, nous considérons une $(p \times p)$-matrice cyclique C^{pt} avec $t \leq p$ chiffres 1 dans chaque ligne, p et t premiers entre eux.

$$C^{pt} = \begin{pmatrix} 1 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \ddots & & & \ddots & \vdots \\ \vdots & \ddots & \ddots & & 0 & \vdots \\ \vdots & & \ddots & 1 & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & & & & & 1 \\ 1 & \ddots & 0 & & & \vdots \\ \vdots & \ddots & & \ddots & & \vdots \\ 1 & \cdots & 1 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

Théorème 5.3.11 : Cornuéjols, Fisher et Nemhauser [CFN77]

La contrainte suivante est appelée contrainte de cycle et est une coupe valide de P_{loc}

$$\sum_{i \in I} \sum_{j \in J} c_{ij}^{pt} x_{ij} - \sum_{j \in J} y_j \leq p - \lceil p/t \rceil$$

avec $\lceil p/t \rceil$ le plus petit entier supérieur à p/t.

Dans les problèmes de localisation traditionnels, la demande de tout client peut être satisfaite par tout fournisseur. Dans le problème de la gestion de la diversité ce n'est plus le cas, en effet la demande d'une combinaison ne peut être satisfaite que par elle même ou par une combinaison de meilleure qualité, c'est-à-dire une combinaison qui lui est supérieure d'après l'ordre partiel.

Dans la section 5.3.4.1 nous expliquons comment réécrire ces contraintes en tenant compte de la structure d'ordre partiel. À la section suivante, nous nous intéressons à un exemple de telles contraintes, et dans la section 5.3.4.3 nous donnons une heuristique efficace permettant de les générer afin de couper la solution fractionnaire courante.

5.3.4.1 DÉFINITION

Dans le théorème 5.3.11, une variable x_{ij} apparaît dans une contrainte de cycle dès que la matrice C^{pt} possède un 1 au croisement de la $i^{\text{ème}}$ ligne et de la $j^{\text{ème}}$ colonne. Cependant, cette variable peut ne pas être définie. Cela se produit lorsque la substitution de la combinaison i par j est interdite, c'est-à-dire lorsque $i \not\preceq j$.

Nous redéfinissons dans cette section ce que nous entendrons par *contrainte de cycle*, et nous en donnons une écriture plus appropriée au problème de la gestion de la diversité.

Nous allons nous intéresser à des sous graphes partiels $G' = (V', A')$ de $G = (V, A)$, avec $V' \subset V$ et $A' \subset A$, très particuliers.

Les sous graphes partiels G' que nous considérons possèdent les propriétés suivantes :

Propriété 1 : Si I désigne l'ensemble des sommets possédant au moins un arc entrant appartenant à A', et J celui des sommets possédant au moins un arc sortant appartenant à A', alors I et J doivent avoir la même cardinalité, notée p. (Noter que I et J ne sont pas forcément disjoints)

Propriété 1 : Il existe un nombre t, premier avec p, et un classement de $I = \{i_1, i_2, ..., i_p\}$ et $J = \{j_1, j_2, ..., j_p\}$ tels que

$$c^{pt}_{nm} = 1 \quad \text{si et seulement si} \quad \begin{cases} \text{ou bien} & i_n j_m \in A' \\ \text{ou bien} & i_n = j_m \in I \cap J \end{cases}$$

D'après les notations que nous avons adoptées, nous devons remplacer les termes y_j par x_{jj}, pour tout $j \in J$. Or, il peut arriver que des combinaisons v appartiennent simultanément à I et à J. Dans ce cas, il existe n et m tels que $v = i_n = j_m$. Si $c^{pt}_{nm} = 1$, alors la variable x_{vv} sera présente non seulement dans le terme $\sum_{i\in I}\sum_{j\in J} c^{pt}_{ij} x_{ij}$ mais aussi dans $\sum_{j\in J} y_j$, et par conséquent elle disparaîtra. Nous allons donc reformuler les contraintes de cycle en tenant compte de cette observation.

Soit T' l'ensemble des "terminaux" du sous graphe partiel G', c'est-à-dire l'ensemble des sommets j appartenant à J et dont la variable x_{jj} n'apparaît pas dans le terme $\sum_{i\in I}\sum_{j\in J} c^{pt}_{ij} x_{ij}$.

Théorème 5.3.12

La contrainte suivante, appelée contrainte de cycle associée au sous graphe partiel G', est valide pour toutes les solutions de P_K, et donc aussi de P_k

$$\sum_{ij \in A'} x_{ij} - \sum_{j \in T'} x_{jj} \leq p - \lceil p/t \rceil$$

Preuve de 5.3.12 : D'après les observations que nous venons de faire, il est évident que ces contraintes sont des contraintes de cycles du problème de localisation. ♢

La figure 5.7 illustre les cinq types de graphes partiels G' de G associés à la matrice cyclique C^{32}. Il est facile de voir qu'il n'existe pas d'autres types de sous graphes partiels de G associés à cette matrice, car le graphe G est un graphe d'ordre partiel, et donc ni lui ni aucun de ses sous graphes partiels ne peut contenir de circuit. Noter qu'en choisissant $t = 2$, le sous graphe partiel G' est un cycle, au sens de la théorie des graphes. Cependant, comme le montre le second schéma, ce cycle peut ne pas être élémentaire : dans cet exemple, le 0 situé au croisement de la colonne b et de la ligne b implique que x_{bb} ne disparaît pas alors qu'il appartient non seulement à I mais aussi à J.

Dans la section suivante nous présentons un exemple où ces contraintes sont liées directement à des cycles élémentaires du graphe $G = (V, A)$.

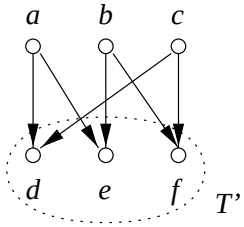
5.3.4.2 EXEMPLES

Considérons un cycle C du graphe $G = (V, A)$. Nous supposons que ce cycle est élémentaire.

Ce cycle C peut être vu comme un sous graphe partiel $G' = (V', A')$ de G où V' et A' sont respectivement les ensembles des sommets et des arcs qui appartiennent à C.

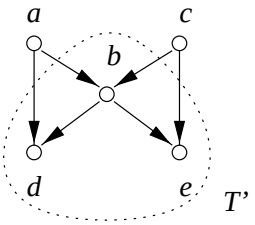
Nous reprenons les notations de la section précédente, à savoir, I désigne l'ensemble des sommets de V' possédant au moins un arc sortant appartenant à A', et J l'ensemble des sommets de V' possédant au moins un arc entrant appartenant à A'.

Nous allons démontrer que les deux propriétés énoncées à la section précédente sont vérifiées par G', ce qui nous permettra d'énoncer la contrainte associée au cycle C.



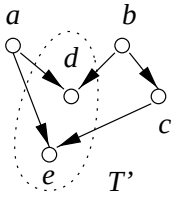
	d	*e*	*f*
a	1	1	0
b	0	1	1
c	1	0	1

$$x_{ad} + x_{ae} + x_{be} + x_{bf} + x_{cd} + x_{cf} - x_{dd} - x_{ee} - x_{ff} \leq 1$$



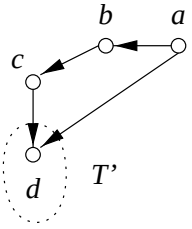
	b	*d*	*e*
a	1	1	0
b	0	1	1
c	1	0	1

$$x_{ab} + x_{ad} + x_{bd} + x_{be} + x_{cb} + x_{ce} - x_{bb} - x_{dd} - x_{ee} \leq 1$$



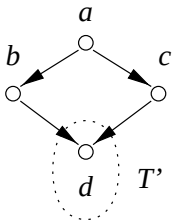
	e	*d*	*b*
a	1	1	0
b	0	1	1
c	1	0	1

$$x_{ae} + x_{ad} + x_{bd} + x_{ce} + x_{cb} - x_{dd} - x_{ee} \leq 1$$



	d	*b*	*c*
a	1	1	0
b	0	1	1
c	1	0	1

$$x_{ad} + x_{ab} + x_{bc} + x_{cd} - x_{dd} \leq 1$$



	c	*b*	*d*
a	1	1	0
b	0	1	1
c	1	0	1

$$x_{ac} + x_{ab} + x_{bd} + x_{cd} - x_{dd} \leq 1$$

FIG. 5.7 – *Graphes partiels G' associés à la matrice C^{32}*

Lemme 5.3.13

I et J ont le même nombre p d'éléments.

Preuve de 5.3.13 : Par construction, les sommets de G' sont incidents à exactement deux arcs de A', car G' est un cycle élémentaire. En conséquence, l'ensemble des sommets v de V' se décompose en trois catégories suivant la nature de ces deux arcs :

Catégorie 1 : les deux arcs sont sortants, ils s'écrivent vw et vw'

Catégorie 2 : les deux arcs sont entrants, ils s'écrivent uv et $u'v$

Catégorie 3 : l'un des arcs est entrant, et l'autre sortant, ils s'écrivent uv et vw

Il est clair que I est composé des sommets de la première et de la troisième catégorie, et que les sommets de J sont ceux de la seconde et de la troisième catégorie.

En conséquence, les sommets de la troisième catégorie appartenant simultanément à I et J, nous allons prouver ce lemme en les enlevant un à un, ce qui préservera la différence de cardinalité entre I et J. La suppression de ces sommets v s'effectue en remplaçant les arcs uv et vw par un seul arc uw.

Le graphe que nous obtenons à la fin de ces opérations est un cycle composé alternativement de sommets des deux premières catégories. De cette alternance, nous en déduisons que les nombres de sommets appartenant à ces deux catégories sont égaux, et donc que $|I| = |J|$. ♢

Dans la suite nous supposerons que la cardinalité commune p de I et J est impaire.

Le lemme suivant nous prouve que la seconde propriété exigée pour pouvoir énoncer une contrainte de cycle est aussi vérifiée.

Lemme 5.3.14

Il existe des classements de $I = \{i_1, i_2, \ldots, i_p\}$ et $J = \{j_1, j_2, \ldots, j_p\}$ tels que $c^{p2}_{nm} = 1$ si et seulement si

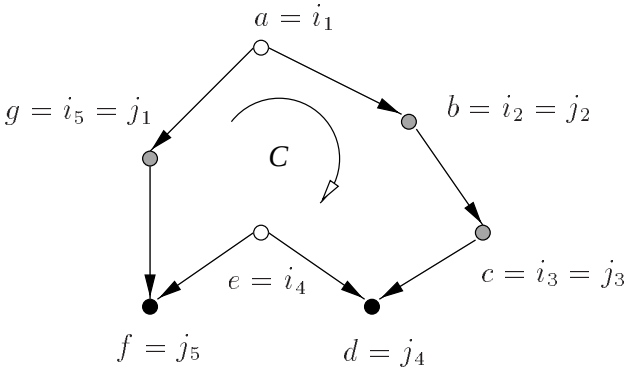
- *ou bien $i_n j_m \in A'$*
- *ou bien $i_n = j_m \in I \cap J$*

Preuve de 5.3.14 : Nous choisissons un sens de parcours du cycle C.

Soit i_1 le premier sommet rencontré qui ne possède pas d'arc entrant appartenant à A'. Ce sommet existe car C est un sous graphe partiel de G, et celui-ci est un graphe sans circuit, étant un graphe d'ordre partiel. Nous définissons i_2, i_3, ...,

i_p comme étant les sommets de I rencontrés après i_1 en parcourant C dans le sens qui a été choisi.

Soit j_1 le sommet qui précède i_1. Par construction, $j_1 \in J$, car i_1 est relié à ses deux voisins par des arcs sortants. Nous définissons j_2, j_3, ..., j_p comme étant les sommets de J rencontrés après j_1 en parcourant C dans le sens qui a été choisi. Voir l'exemple de la figure 5.8.



I \ J	g	b	c	d	f
a	1	1	0	0	0
b	0	1	1	0	0
c	0	0	1	1	0
e	0	0	0	1	1
g	1	0	0	0	1

FIG. 5.8 – *Classement des ensembles I et J*

En raisonnant par récurrence, il est facile de démontrer que pour tout $n \in \{1, 2, ..., p-1\}$, l'arc $i_n j_{n+1}$ appartient à A', et que $i_n = j_n$ ou $i_n j_n \in A'$. Il en est de même pour i_p avec j_p et j_1.

Pour conclure la preuve de ce lemme, il suffit de remarquer que $c^{p2}_{nm} = 1$ si et seulement si $m = n$ ou $m = n + 1$ (modulo p).

♢

Il est facile de voir que l'ensemble T' des "terminaux" de C, défini à la section précédente, correspond aux éléments de V' de la seconde catégorie, c'est-à-dire ceux qui ne possèdent pas d'arc sortant appartenant à A'. Dans la figure 5.8, ces sommets sont représentés par des disques noirs. Les sommets avec un disque blanc correspondent aux éléments de la première catégorie, et ceux avec un disque gris sont associés aux sommets de la troisième catégorie.

Le théorème 5.3.12 s'écrit donc :

Théorème 5.3.15

La contrainte suivante, appelée contrainte associée au cycle élémentaire C, est valide pour toutes les solutions de P_K, et donc aussi de P_k

$$\sum_{ij \in A'} x_{ij} - \sum_{j \in T'} x_{jj} \leq \frac{p-1}{2}$$

Dans l'exemple de la figure 5.8, nous avons $p = 5$, $A' = \{ag, ab, bc, cd, ed, ef, gf\}$ et $T' = \{f, d\}$. La contrainte qui est associée à ce cycle est donc :

$$x_{ag} + x_{ab} + x_{bc} + x_{cd} + x_{ed} + x_{ef} + x_{gf} - x_{dd} - x_{ff} \leq 2$$

La figure 5.9 représente une solution fractionnaire qui est coupée par ce type de contrainte. Le cycle C que nous considérons est le cycle constitué de l'ensemble de sommets $V' = \{13, 25, 15, 37, 17, 32\}$. Nous avons $T' = \{32, 37, 25\}$. La contrainte associée à ce cycle est donc :

$$x_{13,32} + x_{13,25} + x_{15,25} + x_{15,37} + x_{17,37} + x_{17,32} - x_{32,32} - x_{37,37} - x_{25,25} \leq 1$$

Dans cette solution, toutes les variables mentionnées valent 0.5, or nous avons $1.5 > 1$, donc en ajoutant cette contrainte violée, nous n'obtiendrons plus cette solution fractionnaire.

Rechercher des cycles dont les contraintes sont violées n'est pas une tâche facile. L'idée la plus simple serait d'étudier tous les cycles possibles du graphe G, et de tester si la solution courante (supposée fractionnaire) vérifie leur contrainte. Mais cette première méthode de recherche n'est, en pratique, pas envisageable, car le nombre de cycles d'un graphe est exponentiel. À la section suivante, nous allons étudier une autre méthode de recherche plus efficace.

5.3.4.3 HEURISTIQUE DE GÉNÉRATION DE CYCLES

Dans cette section, nous présentons une heuristique permettant de rechercher des cycles de G dont la contrainte est violée par la solution fractionnaire courante.

Cette méthode n'est qu'une heuristique dans le sens qu'elle peut ne pas fournir de cycle dont la contrainte est violée par la solution courante, alors qu'il en existe un.

Nous notons $\bar{x}$ la solution optimale de la relaxation courante. Cette solution est supposée fractionnaire, sinon elle serait optimale pour $(\mathcal{P}_k)$ et nous ne chercherions pas à la couper.

Nous allons nous intéresser uniquement aux cycles C constitués d'arcs ij tels que $0 < \bar{x}_{ij} = \bar{x}_{jj} < 1$. Nous dirons que de tels arcs sont *saturés*. Un cycle C constitué uniquement d'arcs saturés sera dit, lui aussi, *saturé*.

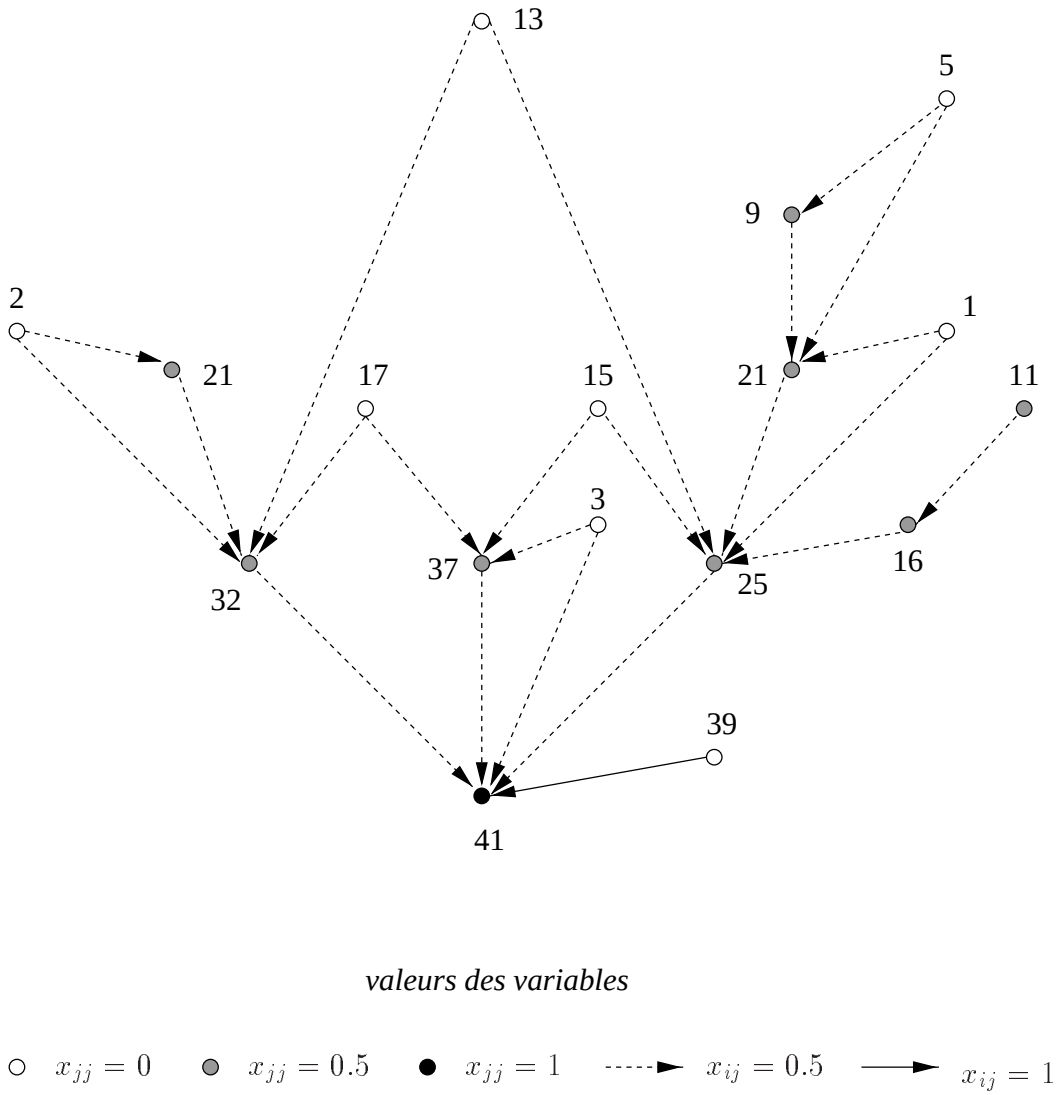


FIG. 5.9 – *Exemple d'une solution fractionnaire*

Cette heuristique est basée sur l'observation suivante :

Propriété 5.3.16

Soit C un cycle saturé de G.
Soit J l'ensemble des sommets de C possédant au moins un arc entrant appartenant à C. Nous supposerons que $p = |J|$ est impair.
La solution $\bar{x}$ viole la contrainte de C si et seulement si :

$$\sum_{j \in J} (\bar{x}_{jj} - \frac{1}{2}) > -\frac{1}{2}$$

Preuve de 5.3.16 : Cette propriété provient d'une réécriture des contraintes de cycle telles qu'elles ont été énoncées à la section 5.3.4.1, en prenant $t = 2$. Il suffit de remarquer que pour tout $j \in J$, il existe exactement deux variables x_{ij} appartenant au premier terme, puis de simplifier l'expression en tenant compte du fait que $\bar{x}_{ij} = \bar{x}_{jj}$ d'après la saturation des arcs, et enfin de faire passer le terme du second membre $p/2$ dans la somme du premier membre de l'inéquation. $\diamondsuit$

Cette remarque nous montre que, pour des cycles saturés, seuls les sommets $j \in V$ possédant au moins un arc entrant saturé ont un rôle réel dans la violation de la contrainte.

Nous construisons donc un graphe non orienté $\bar{G}$ ayant pour sommets les extrémités finales des arcs saturés de G, et dans lequel il existe une arête entre deux sommets j et j' si et seulement si il existe $i \in V$ tel que $i \preceq j$ et $i \preceq j'$ et tels que $\bar{x}_{ij} = \bar{x}_{jj}$ et $\bar{x}_{ij'} = \bar{x}_{j'j'}$. En d'autres termes, il existe une arête entre j et j' si et seulement si

- soit il existe deux arcs saturés ij et ij'
- soit $j \prec j'$ et $\bar{x}_{jj'} = \bar{x}_{j'j'}$

Noter que l'arête jj' peut être due à plusieurs sommets i.

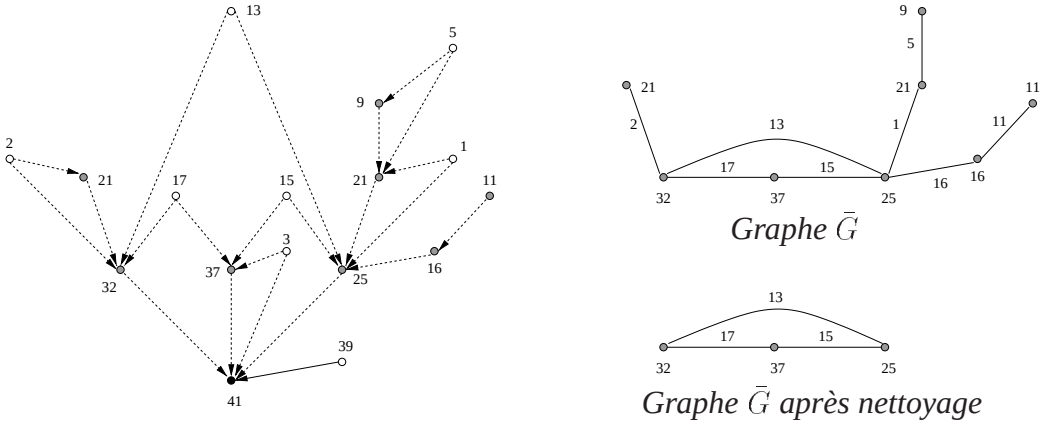
La figure 5.10 illustre le graphe $\bar{G}$ construit à partir de la solution fractionnaire de la figure 5.9. Le numéro sur une arête correspond à l'un des sommets du graphe G à qui elle est associée.

L'intérêt de ce graphe $\bar{G}$ est qu'à tout cycle saturé de V, tel que $|J| = p$, correspond un cycle de p éléments dans $\bar{G}$. La réciproque est vraie et même plus forte car à tout cycle de p éléments dans $\bar{G}$ peut correspondre plusieurs cycles saturés de V, tel que $|J| = p$.

Nous pondérons chaque sommet j de $\bar{G}$ par $\bar{x}_{jj} - 0.5$, et nous appelons *poids d'un cycle* la somme des poids des sommets qui le composent.

L'heuristique se résume donc à chercher un cycle de cardinalité impaire dont le poids est strictement supérieur à -0.5. Cette recherche est d'autant plus rapide que les nombres de sommets et d'arêtes sont réduits par rapport à ceux de G. Pour

accélérer cette heuristique, nous pouvons aussi "nettoyer" $\bar{G}$ en éliminant tous les sommets isolés ou pendants, et en réitérant ce processus jusqu'à ce que tout sommet de $\bar{G}$ appartienne à au moins un cycle.



Solution courante $\bar{x}$

Graphe $\bar{G}$

Graphe $\bar{G}$ après nettoyage

FIG. 5.10 – *Heuristique de génération de contraintes de cycle*

Sur la figure 5.10, nous voyons immédiatement que le graphe résultant du nettoyage de $\bar{G}$ possède un cycle impair constitué des sommets 32, 37 et 25 (d'ailleurs, dans notre exemple, $\bar{G}$ se résume à un seul cycle). Le poids de ce cycle est égal à $0 > -0.5$, donc, dans G, la contrainte associée au cycle constitué des sommets 13, 25, 15, 37, 17 et 32, est violée par la solution fractionnaire courante. Il s'agit de la contrainte donnée à la section précédente.

5.4 GÉNÉRATION DE VARIABLES

La génération des variables est une phase particulièrement cruciale pour le problème de la gestion de la diversité. En effet, le nombre de variables d'arcs peut être de l'ordre de plusieurs centaines de milliers, et il s'avère qu'en pratique, dès qu'une variable x_{ij} est générée, nous devons aussi générer la contrainte $x_{ij} \leq x_{jj}$, sinon celle-ci sera violée par la solution optimale de la prochaine relaxation linéaire. Il est donc important de bien gérer l'ajout des variables.

A priori, chaque variable dont le coût réduit est négatif est candidate pour entrer dans la relaxation linéaire. Si le nombre de ces variables à coût réduit négatif est très élevé, nous devons en désigner seulement un sous ensemble. Une première

idée est de ne choisir que les variables dont les coûts réduits sont les plus négatifs. Cependant, il arrive fréquemment que ces “meilleures” candidates correspondent à des arcs possédant la même origine, c’est-à-dire à des variables x_{ij} associées aux substitutions d’une même combinaison i. Il est clair que la plupart d’entre elles seront nulles dans la solution optimale de la prochaine relaxation linéaire, car leur somme doit être inférieure ou égale à 1 (égale si toutes les substitutions ij sont considérées).

Nous avons donc décidé de restreindre, pour chaque combinaison i, le nombre de variables x_{ij} que nous ajoutons. Ce nombre maximal est arbitraire, les tests nous ont amené à choisir la valeur de 3. Ainsi, nous parcourons toutes les combinaisons i, et nous ne retenons que les trois meilleures variables x_{ij}. Malgré cette stratégie, le nombre de variables retenues peut théoriquement être proche du triple du nombre de combinaisons, ce qui peut représenter encore plusieurs milliers de variables. Nous classons donc toutes ces variables d’arcs retenues suivant leur coût réduit dans l’ordre croissant, quelle que soit leur origine, puis nous ne gardons que celles en tête de ce classement, afin d’atteindre un nombre de variables générées de l’ordre de quelques centaines.

De plus, la génération d’une contrainte x_{ij} n’a d’intérêt que si la variable x_{jj} appartient au programme. En effet, si x_{jj} n’apparaît pas dans la relaxation linéaire, alors la contrainte $x_{ij} \leq x_{jj}$ aura pour conséquence d’attribuer la valeur 0 à la variable x_{ij} dans toute solution de cette relaxation linéaire. Ainsi, à chaque génération de variables x_{ij}, nous générons systématiquement la variable x_{jj}, si elle est actuellement absente du programme.

Enfin, pour préserver une taille raisonnable à la relaxation linéaire, nous éliminons périodiquement des variables hors base dont le coût réduit est fortement positif. Cette méthode, quoi que très efficace, peut ralentir l’algorithme en regénérant des variables précédemment éliminées. Pour éviter ce problème, nous n’excluons du programme une variable que si son coût réduit reste très positif pendant un certain nombre d’itérations. Ce nombre ne doit pas être trop grand, car sinon nous perdrions le bénéfice que nous cherchions à obtenir par cette stratégie d’élimination. Les tests nous ont montré que ce nombre pouvait être fixé arbitrairement à 5.

5.5 RÈGLES DE BRANCHEMENT

Nous sommes dans le cas où la solution de la relaxation linéaire courante est fractionnaire, et nous n’arrivons pas à l’éliminer en générant de nouvelles contraintes.

Nous devons alors *brancher*, c'est-à-dire générer de nouveaux nœuds "fils" de l'arbre d'énumération du Branch-and-Cut. Pour que cet algorithme soit efficace, il faut que toute solution réalisable associée au nœud courant appartienne à au moins l'un des nœuds fils.

Pour résoudre un problème en variables binaires, la règle de branchement la plus courante est de considérer une variable qui vaut 0 ou 1 dans toutes les solutions réalisables du problème initial mais qui est fractionnaire dans la solution de la dernière relaxation linéaire. Cette règle consiste à générer deux nœuds fils : dans le premier nous exigeons que la variable soit égale à 1, et dans le second à 0. L'idée étant de partitionner suivant ce simple critère l'ensemble des solutions réalisables du nœud courant en deux sous ensembles.

Le principal inconvénient d'une telle règle est de déséquilibrer la partition obtenue, et par conséquent, l'arbre d'énumération. En effet, pour des problèmes tels que celui de la diversité, les solutions réalisables possèdent beaucoup moins de variables à 1 que de variables à 0. L'ensemble des solutions associées au premier nœud fils, c'est-à-dire celui où la variable de branchement vaut 1, sera donc bien plus restreint que celui associé au second nœud fils. Cette observation reste vraie même si nous tenons compte, comme nous l'avons vu à la section 1.5, du fait que seules les variables x_{jj} doivent être binaires, car nous ne nous intéressons uniquement qu'aux petites valeurs de k.

La nature du problème traité permet parfois de surmonter cette difficulté en branchant sur une contrainte (voir par exemple [NW88c]). Si $\bar{x}$ désigne la solution de la relaxation linéaire courante, nous partitionnons l'ensemble S des solutions réalisables appartenant au nœud courant, en deux ensembles S^1 et S^2 définis par $S^1 = S \cap \{x \in I\!R^n_+ \mid fx \leq f_0\}$ et $S^2 = S \cap \{x \in I\!R^n_+ \mid fx \geq f_0 + 1\}$, avec $(f, f_0) \in Z^{n+1}$, et $f_0 < f\bar{x} < f_0 + 1$. Ainsi, nous pouvons générer deux nœuds fils, où le premier est associé à S_1, et le second à S_2.

D. Naddef et J.-M. Clochard ([CN93]) ont montré l'efficacité de ce type de règle pour le problème du voyageur de commerce. Les contraintes utilisées pour définir la règle de branchement sont les contraintes de sous-tours, c'est-à-dire celles qui traduisent le fait que le nombre d'arêtes appartenant au cocycle $\delta(V')$, associé à n'importe quel sous-ensemble de sommets V' non vide et différent de l'ensemble des sommets tout entier, doit être pair. Pour résoudre le problème du voyageur de commerce, on défini des variables x_{ij} qui valent 1 si l'arc ij appartient au cycle hamiltonien, et 0 sinon. En considérant la solution $\bar{x}$ de la relaxation linéaire courante, la règle de branchement consiste à trouver un ensemble V' tel que $\bar{x}(\delta(V'))$ est proche d'un nombre impair $2p + 1$, puis à générer deux nœuds fils en imposant, pour le premier, que $x(\delta(V')) \leq 2p$, et pour le second $x(\delta(V')) \geq 2p + 2$.

Pour le problème de la gestion de la diversité, nous exploitons cette idée en utilisant les contraintes $\sum_{j \geq i} x_{ij} = 1$. Ces contraintes traduisent que chaque com-

binaison i doit, ou bien être produite, ou bien être remplacée par une autre. Dans le cas où la solution $\bar{x}$ de la dernière relaxation linéaire est fractionnaire, il existe au moins une combinaison i pour laquelle cette traduction n'est plus aussi évidente. En effet, il existe alors plusieurs combinaisons $j \succeq i$ pour lesquelles la variable x_{ij} est fractionnaire.

La règle de branchement que nous définissons revient à choisir une combinaison j pour laquelle la variable x_{jj} est fractionnaire, et à générer deux nœuds fils où, dans le premier, x_{jj} sera à 1, et, dans le second, à 0. Cette règle est a priori une règle de branchement traditionnelle sur les variables, mais elle s'apparente aussi aux branchements sur des contraintes dans le sens que nous allons, pour choisir j, exploiter toutes les conséquences qu'entraînent x_{jj} à 0.

Comme nous l'avons vu au chapitre 3, exiger qu'une variable x_{jj} soit à 0, c'est-à-dire s'interdire de produire la combinaison j, n'a que peu de conséquences directes mais peut avoir, par effet "boule de neige", beaucoup de conséquences indirectes. Par exemple, si une combinaison i est fixée à 0 et si elle ne peut plus être remplacée que par j et par une autre combinaison j', alors ne pas produire j entraîne la production obligatoire de j'. Le meilleur outil pour analyser ce type de conséquences indirectes est l'hypergraphe $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ que nous avons décrit à la section 3.3.4. Cet hypergraphe a été défini par

- $\mathcal{V} = V \setminus (V^1 \cup V^0)$
- $\mathcal{E} = \{\ E_i \ \mid\ E_i \cap V^1 = \emptyset,\ \ i \in V \ \text{ et } E_i \not\subset E_j\ \forall j \in V\}$

$$\begin{array}{lll} \text{avec} & E_i = \{\ j \succ i \ \mid\ ij \notin A^0\ \} & \text{si } i \in V^0 \\ \text{et} & E_i = \{\ j \succ i \ \mid\ ij \notin A^0\ \} \cup \{i\} & \text{si } i \notin V^0 \end{array}$$

L'intérêt de cet hypergraphe est que chaque arête correspond à l'ensemble des combinaisons qui peuvent remplacer une combinaison donnée. Ainsi, nous avons l'assurance que, dans toutes les solutions réalisables que nous cherchons, au moins une combinaison sera produite dans chaque arête. Comme nous l'avons signalé plus haut, s'il existe une arête ne possédant que deux combinaisons, alors s'interdire de produire l'une revient à s'obliger à produire l'autre. Nous allons donc choisir la variable de branchement comme étant l'une des variables dont la valeur est fractionnaire dans la solution de la dernière relaxation linéaire, et dont la combinaison à laquelle elle est associée appartient à une arête de cardinalité minimale. Si plusieurs combinaisons sont candidates, nous choisissons celle qui appartient au plus grand nombre d'arêtes, car alors ne pas produire cette combinaison aura plus de conséquences que les autres. Si un choix multiple subsiste, nous choisissons la variable dont la valeur dans la solution de la dernière relaxation linéaire est la plus proche de 0.5.

La figure 5.11 illustre cette règle sur un exemple. Les combinaisons, dont la variable est fractionnaire dans la solution de la dernière relaxation linéaire, sont

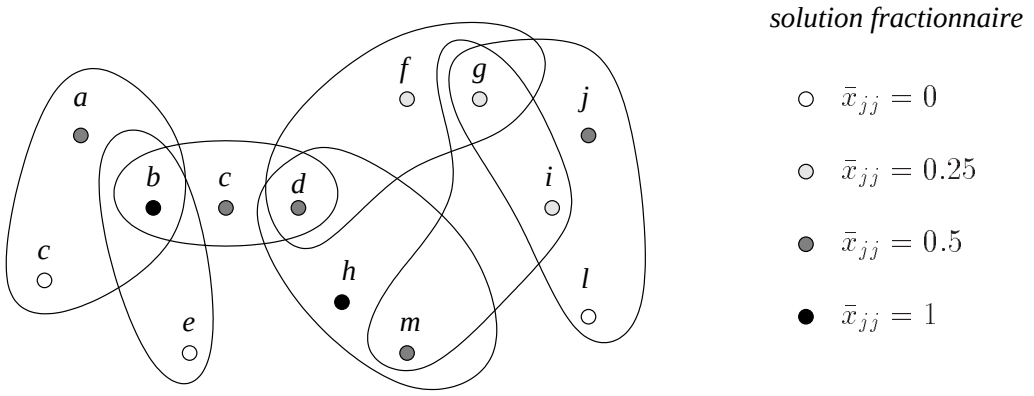


FIG. 5.11 – *Règle de branchement*

a, c, d, f, g, i, j et m. Excepté j, elles appartiennent toutes à des arêtes de cardinalité minimale (ici 3). Les combinaisons d et g sont celles qui appartiennent au plus grand nombre d'arêtes, elles sont à l'intersection chacune de trois arêtes. La variable sur laquelle nous allons brancher est donc x_{dd} car $|\bar{x}_{dd} - 0.5| = 0 < |\bar{x}_{gg} - 0.5| = 0.25$.

Dans chacun des deux nœuds fils, avant même de résoudre le premier programme linéaire, nous appelons les procédures de fixation par implication logique décrites à la section 3.3. Contrairement à l'algorithme lagrangien, les fixations ne sont ici que temporaires, elles ne sont valables que pour ce nœud, et les nœuds appartenant l'arborescence d'énumération dont il est racine. Cet appel prend toute son importance lorsque le nombre de fixations est déjà grand : le nombre de variables et de contraintes des relaxations linéaires à résoudre est alors très restreint.

Il peut arriver qu'à partir d'un nœud il n'existe plus de solution réalisable. Pour le confirmer, nous pourrions générer toutes les variables de la relaxation linéaire, et tenter de résoudre ce programme, l'algorithme du simplexe détectant ce type de problème lors de sa première phase. Cependant, si le nombre de variables est trop important, il est préférable de déceler ce phénomène à l'aide d'arguments plus simples. Ces arguments s'appuient sur la structure du nouvel hypergraphe que nous obtenons après ces fixations. Le premier, le plus trivial, consiste à vérifier qu'aucune arête n'est vide, sinon, cela signifierait qu'il existe une combinaison fixée à 0 qui ne peut plus être substituée. Le second revient à exhiber le plus grand nombre possible d'arêtes deux à deux disjointes. Ceci se fait à l'aide de l'algorithme 3 décrit à la section 3.3.4. Si ce nombre est supérieur au nombre de combinaisons non fixées à 1 restant à choisir, alors il est clair qu'aucune solution réalisable ne pourra être trouvée. Cet algorithme n'étant qu'une heuristique, il

peut arriver que l'inexistence de solutions réalisables ne soit pas prouvée. Nous n'avons alors pas d'autre recours que la résolution du programme linéaire. Noter que cette situation ne se produit que si beaucoup de variables ont été fixées, ce qui implique que la taille du problème est déjà très réduite.

5.6 CONCLUSION

Le premier but que nous nous étions fixé était de résoudre jusqu'à l'optimalité le problème de la gestion de la diversité. Ce but était très souvent atteint, en pratique, par l'algorithme des relaxations lagrangiennes, mais l'apport de l'algorithme *Branch and Cut* que nous avons décrit dans ce chapitre est la preuve de cette optimalité.

Pour utiliser ce type d'algorithme, nous avons étudié le polytope des solutions de ce problème. Nous avons notamment donné sa dimension, quelques-unes de ses facettes, et une classe de contraintes valides permettant de couper certaines solutions fractionnaires.

Malheureusement, nous n'avons pas réussi à endiguer complètement l'inconvénient majeur de cet algorithme, inconvénient qui nous avait poussés, à l'origine, à réduire la taille du problème grâce aux relaxations lagrangiennes, à savoir : la génération excessive de variables et de contraintes.

L'étude approfondie de ce polytope est un problème ouvert. Cette étude est particulièrement difficile à mener, mais elle est d'autant plus intéressante qu'elle pourra aisément se généraliser au polytope associé au problème de localisation k-médian, qui demeure, lui aussi, très peu étudié jusqu'à ce jour.

6

RÉSULTATS NUMÉRIQUES

Sommaire

Ce dernier chapitre est consacré aux tests numériques que nous avons effectués sur des instances réelles.

Dans un premier temps, nous décrivons le contexte informatique dans lequel a été effectué ces tests, puis nous détaillons les instances que nous utiliserons dans ce chapitre.

Nous montrons ensuite l'efficacité de la méthode des fixations de variables, et nous présentons plusieurs résultats illustrant la très bonne qualité des solutions réalisables fournies par les heuristiques décrites au chapitre 4. Puis, nous consacrons une section à l'apport des nouveaux critères de fixation, que nous avons décrits dans ce document.

Enfin, nous abordons les limites de notre algorithme, et nous discutons de l'utilité ou de l'inutilité de l'algorithme *Branch and Cut* pour résoudre ce type de problème.

Mais avant de présenter ces résultats, nous précisons le contexte purement informatique dans lequel ils ont été effectués.

6.1 CONTEXTE INFORMATIQUE

Pour réaliser les tests qui nous permettrons de valider notre travail, nous avons écrit un programme modulaire écrit en C++.

Ce programme se décompose en deux parties principales, qui peuvent être utilisées indépendamment l'une de l'autre : la première est consacrée à l'algorithme lagrangien que nous avons mis au point, et le second à l'algorithme *Branch and Cut*.

La grande modularité de chacune de ces parties rend ce programme très facilement adaptable au traitement d'autres problèmes. De plus, nous avons écrit pour chaque type de variables (sommets ou arcs) des fonctions de fixation à 0 ou 1 distinctes rendant aisés leurs appels récursifs et leur lisibilité. Ce choix de programmation a été motivé par la volonté d'ajouter à tout moment de nouveaux critères de fixation. Pour les mêmes raisons, à chaque heuristique, `2-OPT` y compris, est associée une procédure propre, ce qui permet à l'utilisateur d'implanter de nouvelles heuristiques sans trop de changement majeur.

La partie dédiée à l'algorithme *Branch and Cut* est basée sur une utilisation du progiciel Abacus, version 2.2, développé à l'université de Cologne par S. Thienel et M. Jünger ([Thi95], [JT96], [JT97]). Cette utilisation consiste en la redéfinition de classes dédiées à chacune des composantes définissant un algorithme *Branch and Cut* : les variables et les contraintes du problème étudié, la définition du programme maître du nœud racine, méthodes de génération de colonnes et

de contraintes, règles de branchement, etc... Pour la résolution des programmes linéaires, nous utilisons le logiciel CPLEX, version 6.0.

Les efforts d'implantation que nous avons fournis ont constitué une partie très importante de ce travail de recherche.

Les résultats présentés dans les sections suivantes ont été obtenus sur une machine biprocesseur, UltraSparc-II à 295 Mhz, 512 Moctets de mémoire vive, 4 Moctets de mémoire cache.

6.2 DESCRIPTION DES INSTANCES TESTÉES

Nous consacrons cette première section à la description des instances sur lesquelles nous avons effectué nos tests, et au rappel de quelques notations.

Nous reprenons les notations du chapitre 3, et nous en ajoutons deux nouvelles concernant l'agglomération des sommets et des arcs (cf section 3.7) :

V	: ensemble des sommets
V^1	: ensemble des sommets fixés à 1
V^0	: ensemble des sommets fixés à 0
V^*	: ensemble des sommets agglomérés
A	: ensemble des arcs
A^1	: ensemble des arcs fixés à 1
A^0	: ensemble des arcs fixés à 0
A^*	: ensemble des arcs agglomérés
NF	: nombre de variables restantes (après fixations et agglomérations)
T	: ensemble des terminaux

Les trois principales instances, auxquelles nous nous intéressons dans ce chapitre, possèdent les caractéristiques suivantes :

Instance	$\|V\|$	$\|A\|$	$\|T\|$
Inst535	535	21 003	2
Inst1284	1 284	88 542	2
Inst5535	5 535	666 639	2

D'après la taille de la première instance, il serait a priori envisageable de résoudre directement, pour chaque valeur de k, le programme linéaire en nombres entiers à l'aide du logiciel CPLEX, ce programme contenant 21 534 variables, dont 533 entières, et un peu plus de 21 000 contraintes. Nous l'indiquons comme étant une instance témoin. En revanche, ce n'est plus le cas pour les deux autres

instances : le programme associé à l'instance Inst1284 contient près de 90 000 variables, dont 1 282 entières, et autant de contraintes, et celui de l'instance Inst5535 possèdent près de 700 000 variables, dont 5 533 entières, et autant de contraintes.

Il est important de noter que les instances des problèmes de localisation traitées dans la littérature ne possèdent au plus que quelques centaines de sommets correspondant à des sites où des usines peuvent être exploitées lorsque le but est de prouver l'optimalité des solutions. J.E. Beasley [Bea93a] a effectué des tests avec succès sur des instances possédant de 16 à 500 sites d'exploitation possibles, et entre 50 et 1000 clients, ce qui correspond à des graphes possédant quelques centaines de sommets, et un nombre d'arcs de l'ordre de 16 000. Sur 40 instances et/ou valeurs de k testées pour le problème k-médian, il trouve 38 solutions optimales, dont 20 pour lesquelles cette optimalité est prouvée. Dans tous les cas, la garantie obtenue est inférieure à 0.1%, et le temps moyen d'exécution est de l'ordre de 14 secondes sur une machine Cray X-MP. Pour le problème des grappes, J.M. Mulvey et H.P. Crowder [MC79] ont testé leur algorithme sur des instances contenant entre 24 et 201 points, ce qui correspond à un nombre de variables compris entre 576 et 40 401. Les nombres de grappes testés sont comprises entre 4 et 8, et l'algorithme se termine dès que l'on est sûr d'être à moins de 1% de l'optimal, ou lorsque le nombre d'itérations de l'algorithme lagrangien est supérieur à 120. La garantie de 1% est très souvent atteinte pour les instances de petite ou de moyenne taille, seule l'instance possédant 40 401 variables n'a pu être résolue qu'avec une garantie de 3 à 8%, quelle que soit le nombre de grappes demandé. Citons enfin, l'article de F. Barahona et F.A. Chudak [BC99] dans lequel les auteurs traitent des instances aléatoires du problème de localisation avec coût fixe possédant entre 500 et 3000 sommets. Ils utilisent une heuristique originale, appelé algorithme du volume. L'algorithme lagrangien s'arrête dès qu'une garantie de 1% est assurée. Les valeurs des coûts fixes sont de $\sqrt{n}/10$, $\sqrt{n}/100$ et $\sqrt{n}/1000$. Les temps d'exécution obtenues sont de l'ordre de quelques minutes pour les instances de petites tailles, et de une ou deux heures pour les instances de grandes tailles.

Ainsi, nous pouvons considérer que les tests que nous présentons ont été effectués sur des instances de grandes, voire de très grandes tailles.

Les sections suivantes synthétisent les résultats, d'abord sur le plan des fixations de variables, puis sur celui de la qualité des solutions fournies par notre algorithme lagrangien.

6.3 FIXATIONS DE VARIABLES

Nous consacrons cette section aux fixations de variables dans l'algorithme lagrangien.

Les tableaux suivants regroupent les résultats obtenus sur les instances Inst535, Inst1284 et Inst5535 du problème de la gestion de la diversité, pour des valeurs de k allant de 5 à 20.

Nous rappelons que le nombre de sommets restants est égal à $|V|-|V^0|+|V^1|$, et celui des arcs restants est de $|A| - |A^0| + |A^1| + |A^*|$. En effet, lors de son agglomération, un sommet ne peut être que fixé à 0, ce qui implique $V^* \subset V^0$. En revanche, un arc ne peut être aggloméré que s'il n'est pas fixé.

k	$\|V^1\|$	$\|V^0\|$	$\|V^*\|$	$\|A^1\|$	$\|A^0\|$	$\|A^*\|$	NF	% fixées	Temps
5	5	530	301	229	19070	1704	0	100	2.9
6	6	529	273	256	19465	1282	0	100	3.4
7	7	528	123	405	20105	493	0	100	3.5
8	8	527	224	303	19579	1121	0	100	5.2
9	9	526	351	175	19366	1462	0	100	5.1
10	10	525	368	157	18606	2240	0	100	6.4
11	11	524	192	332	19612	1059	0	100	5.4
12	12	523	211	312	18419	2272	0	100	5.8
13	13	522	269	253	18841	1909	0	100	11.8
14	14	521	312	209	19010	1784	0	100	10.3
15	7	487	294	104	17924	2445	571	97.35	24.2
16	4	457	256	63	17264	2328	1422	93.40	39.7
17	10	495	327	89	17989	2513	442	97.95	26.2
18	18	517	350	167	18509	2327	0	100	27.1
19	19	516	334	182	18187	2634	0	100	17.0
20	20	515	322	193	18653	2157	0	100	25.6

TAB. 6.1 – *Fixations de variables pour l'instance Inst535*

Pour ces trois instances, le pourcentage de variables fixées est toujours supérieur à 90%, et pour de nombreuses valeurs de k, il est même de 100%, c'est-à-dire que toutes les variables ont été fixées, et nous avons la preuve que la solution réalisable de $(\mathcal{P}_k)$ fournie par nos heuristiques est optimale.

k	$\|V^1\|$	$\|V^0\|$	$\|V^*\|$	$\|A^1\|$	$\|A^0\|$	$\|A^*\|$	NF	% fixées	Temps
5	5	1279	466	813	76304	11425	0	100	28.7
6	6	1278	757	521	73843	14178	0	100	51.0
7	7	1277	860	417	72186	15939	0	100	150.9
8	8	1276	1008	268	75054	13220	0	100	30.7
9	9	1275	944	331	78540	9671	0	100	39.2
10	10	1274	965	309	74066	14167	0	100	77.6
11	2	1189	508	57	69273	14748	4557	94.93	306.4
12	3	1191	529	88	69424	14939	4181	95.35	301.2
13	3	1203	583	94	69836	15325	3365	96.25	168.8
14	14	1270	942	328	70886	17328	0	100	83.8
15	15	1269	919	350	72540	15652	0	100	113.7
16	16	1268	993	275	72691	15576	0	100	95.7
17	17	1267	903	364	74147	14031	0	100	75.8
18	18	1266	909	357	73963	14222	0	100	87.6
19	19	1265	953	312	75626	12604	0	100	92.6
20	20	1264	901	363	76761	11418	0	100	75.4

TAB. 6.2 – *Fixations de variables pour l'instance Inst1284*

k	$\|V^1\|$	$\|V^0\|$	$\|V^*\|$	$\|A^1\|$	$\|A^0\|$	$\|A^*\|$	NF	% fixées	Temps
5	5	5530	3091	2439	637682	26518	0	100	421.78
6	6	5529	3391	2138	623033	41468	0	100	538.72
7	7	5528	3001	2527	625661	38451	0	100	499.47
8	8	5527	3575	1952	626183	38504	0	100	872.42
9	9	5526	3362	2164	615863	48612	0	100	1536
10	10	5525	3339	2186	621784	42669	0	100	1226.65
11	11	5524	3775	1749	623091	41799	0	100	686.75
12	12	5523	4002	1521	622220	42898	0	100	626.02
13	13	5522	3790	1732	619999	44908	0	100	1093.22
14	4	5433	2436	1269	634619	18556	12293	98.17	3987.19
15	4	5440	2569	1374	636017	19067	10272	98.47	3491.07
16	2	5314	1876	299	594041	40632	31886	95.26	3357.37
17	3	5430	2694	695	629106	24331	12609	98.12	3401.71
18	18	5517	2994	2523	645667	18449	0	100	4203.84
19	2	5369	2284	600	623099	21773	21331	96.83	8214.17
20	2	5270	1889	419	605934	24751	35798	94.67	6608.05

TAB. 6.3 – *Fixations de variables pour l'instance Inst5535*

Dans les quelques cas, où l'optimalité de la solution n'a pu être prouvée par l'algorithme lagrangien, le nombre de variables à l'entrée de l'algorithme *Branch and Cut* est très réduit, sauf peut-être pour l'instance Inst5535, pour laquelle ce nombre est de l'ordre de plusieurs dizaines de milliers.

Noter aussi le nombre non négligeable d'arcs et de sommets agglomérés. Ces arcs auraient peut-être été fixés à des itérations ultérieures de l'algorithme, mais le fait de les agglomérer nous a permis immédiatement de ne plus les considérer.

Pour des instances de tailles moyennes, telles que Inst535 ou Inst1284, les temps d'exécution sont relativement faibles (ces temps sont donnés en secondes CPU). Ils sont de l'ordre de quelques minutes, voire seulement quelques secondes. Mais pour des instances de grande taille telles que Inst5535, ils se comptent en plusieurs dizaines de minutes, voire en plusieurs heures. Néanmoins, grâce à cette réduction très importante de la taille du problème, la résolue jusqu'à l'optimalité est désormais accessible.

6.4 QUALITÉ DES SOLUTIONS HEURISTIQUES

Nous nous intéressons dans cette section à la qualité des solutions réalisables fournies par les différentes heuristiques décrites au chapitre 4.

Pour évaluer cette qualité, nous reprenons la première définition du problème de la gestion de la diversité, à savoir, le calcul du sur-coût total, et non du coût total de la production.

En effet, il est facile de voir que pour toute solution réalisable x de $(\mathcal{P}_k)$, le sur-coût total est égal au coût total de la production, auquel nous retranchons ce que nous appelons le coût de la diversité maximale, c'est-à-dire la valeur $CoutDivMax = \sum_{j \in V} c_j d_j$ qui représente le coût de la solution (non réalisable) qui consiste à produire toutes les combinaisons. Ce coût représente une part très importante de la valeur de GUB, de l'ordre de 90%. Si nous ne le retranchions pas, la garantie serait faussée.

La garantie que nous calculons est donc donnée par :

$$garantie = \frac{\texttt{GUB} - \texttt{GLB}}{\texttt{GUB} - CoutDivMax}$$

Les tableaux suivants regroupent, pour les mêmes instances qu'à la section précédente, les valeurs de GLB, GUB et de la garantie. Lorsque cette garantie est nulle (cas où GLB = GUB), la meilleure solution réalisable trouvée est optimale. La colonne suivante indique si la solution associée à GUB est optimale. Ceci s'adresse

principalement aux valeurs de k pour lesquelles l'optimalité de la solution n'a pu être prouvée par l'algorithme lagrangien.

Enfin, nous rappelons les temps d'exécution de l'algorithme lagrangien, temps CPU donnés en secondes.

k	GLB	GUB	Garantie	Optimale	Temps
5	1423493724.62	1423493724.62	optimale	oui	2.85
6	1386675757.02	1386675757.02	optimale	oui	3.36
7	1357240049.56	1357240049.56	optimale	oui	3.49
8	1338640056.80	1338640056.80	optimale	oui	5.19
9	1321037545.88	1321037545.88	optimale	oui	5.14
10	1307942064.84	1307942064.84	optimale	oui	6.45
11	1296526319.88	1296526319.88	optimale	oui	5.38
12	1289226602.38	1289226602.38	optimale	oui	5.82
13	1282932656.62	1282932656.62	optimale	oui	11.85
14	1276985639.40	1276985639.40	optimale	oui	10.28
15	1272111370.89	1272326712.58	0.260%	oui	24.17
16	1267701123.10	1268133527.14	0.551%	oui	39.66
17	1263759627.12	1263939432.68	0.242%	oui	26.16
18	1260169258.64	1260169258.64	optimale	oui	27.12
19	1256440008.62	1256440008.62	optimale	oui	17.02
20	1253052190.74	1253052190.74	optimale	oui	25.64

TAB. 6.4 – *Qualité des solutions pour l'instance Inst535*

Pour ces trois instances, et quelle que soit la valeur de k, les garanties sont toujours excellentes : toujours inférieures à 1%. Cela signifie que l'algorithme lagrangien nous apporte la preuve que la valeur GUB est au plus 1% plus élevée que celle des solutions optimales.

De plus, si nous regardons la cinquième colonne de ces tableaux (celle qui est dénommée "Optimale"), nous nous apercevons que même dans les cas où la garantie n'est pas nulle, la solution qui a permis de définir GUB est toujours une solution optimale du problème. Seule exception : la solution donnée pour l'instance Inst5535, et k égal à 16.

Nous ne présentons ici que des tests dans lesquels l'algorithme lagrangien ne s'arrête qu'après avoir prouvé l'optimalité de la solution, ou après la convergence des faisceaux. En effet, nous cherchons à fixer le maximum de variables, et s'arrêter prématurément irait à l'encontre de cet objectif. Cependant, noter que si nous choisissions de stopper notre algorithme dès que la garantie est inférieure à 1%,

k	GLB	GUB	Garantie	Optimale	Temps
5	21240886156.00	21240886156.00	optimale	oui	28.72
6	20912566164.00	20912566164.00	optimale	oui	50.95
7	20599061276.00	20599061276.00	optimale	oui	150.9
8	20310012152.00	20310012152.00	optimale	oui	30.67
9	20139244060.00	20139244060.00	optimale	oui	39.21
10	20004842380.00	20004842380.00	optimale	oui	77.56
11	19876768201.64	19887426004.00	0.856%	oui	306.44
12	19754666803.14	19765848440.00	0.995%	oui	301.16
13	19648661601.84	19658409188.00	0.959%	oui	168.85
14	19564469268.00	19564469268.00	optimale	oui	83.8
15	19496154396.00	19496154396.00	optimale	oui	113.69
16	19438004644.00	19438004644.00	optimale	oui	95.73
17	19383625284.00	19383625284.00	optimale	oui	75.84
18	19342556624.00	19342556624.00	optimale	oui	87.61
19	19304575940.00	19304575940.00	optimale	oui	92.6
20	19267099356.00	19267099356.00	optimale	oui	75.42

TAB. 6.5 – *Qualité des solutions pour l'instance Inst1284*

k	GLB	GUB	Garantie	Optimale	Temps
5	40084805550	40084805550	optimale	oui	421.78
6	39066780061	39066780061	optimale	oui	538.72
7	38466895096	38466895096	optimale	oui	499.47
8	38199881692	38199881692	optimale	oui	872.42
9	37947771433	37947771433	optimale	oui	1536
10	37730784507	37730784507	optimale	oui	1226.65
11	37556511531	37556511531	optimale	oui	686.75
12	37407092309	37407092309	optimale	oui	626.02
13	37292701431	37292701431	optimale	oui	1093.22
14	37189351618	37195994287	0.295%	oui	3987.19
15	37096879582	37103458047	0.305%	oui	3491.07
16	37008437235	37020655837	0.588%	non	3357.37
17	36927840966	36933656040	0.292%	oui	3401.71
18	36851838482	36851838482	optimale	oui	4203.84
19	36770415375	36778225399	0.426%	oui	8214.17
20	36701582074	36711885136	0.583%	oui	6608.05

TAB. 6.6 – *Qualité des solutions pour l'instance Inst5535*

ce qui est souvent le cas dans la littérature, les temps de calculs seraient réduits considérablement. Nous avons testé cette optique sur quelques instances, et les temps étaient souvent divisés par deux ou trois.

6.5 APPORT DES NOUVEAUX CRITÈRES DE FIXATIONS

Dans cette section nous comparons les résultats obtenus grâce à notre algorithme à ceux que nous aurions obtenus si nous n'avions utilisé que les critères de fixations disponibles dans la littérature.

À notre connaissance, les seuls critères décrits dans la littérature sont ceux issus du théorème 3.4.1 concernant les fixations simples par coût réduit des sommets.

Nous dressons donc, pour les trois instances étudiées dans ce chapitre, un tableau dont la première partie correspond à l'usage de tous les critères de fixation que nous avons exposés dans ce document, et la seconde à l'exploitation uniquement des critères du théorème 3.4.1. Dans cette seconde partie, nous ne faisons donc pas appel aux fixations avancées et fortes, ni aux agglomérations de variables. En revanche, dans les deux cas, nous appliquons toutes les fixations par implication logique que nous avons exposées au chapitre 3.

Dans chacune de ces parties, nous donnons les nombres de sommets et d'arcs fixés, le nombre de variables non éliminées, et la garantie sur la qualité de la borne supérieure et inférieure.

Quelle que soit l'instance, nous pouvons noter que de pour très nombreuses valeurs de k, la preuve de l'optimalité de nos solutions était due aux nouveaux critères de fixations et d'agglomération que nous avons exposés dans ce document. Ceci est d'autant plus important que, pour l'instance Inst5535 avec $k = 18$ par exemple, le nombre de variables restantes peut être de l'ordre de plusieurs dizaines de milliers. Dans de tels cas, le programme linéaires que nous aurons à résoudre aura un trop grand nombre de variables et de contraintes pour pouvoir être résolu efficacement, même avec des méthodes de génération colonnes et de contraintes, car dans notre problème générer une variable x_{ij} implique la génération d'une contrainte $x_{ij} \leq x_{jj}$.

Le second point intéressant que nous pouvons observer, est l'influence des fixations sur la qualité de la garantie. Très fréquemment, ne pas utiliser tous les critères de fixations détériore assez sensiblement la garantie, elle double même

k	Fixations complètes				Fixations traditionnelles			
	somm.	arcs	NF	Garantie	somm.	arcs	NF	Garantie
5	535	21003	0	optimale	535	21003	0	optimale
6	535	21003	0	optimale	535	21003	0	optimale
7	535	21003	0	optimale	535	21003	0	optimale
8	535	21003	0	optimale	527	20827	184	0.003%
9	535	21003	0	optimale	520	20283	735	0.367%
10	535	21003	0	optimale	512	20198	828	0.407%
11	535	21003	0	optimale	535	21003	0	optimale
12	535	21003	0	optimale	535	21003	0	optimale
13	535	21003	0	optimale	491	19097	1950	0.291%
14	535	21003	0	optimale	503	19448	1587	0.100%
15	494	20473	571	0.260%	461	17458	3619	0.386%
16	461	19655	1422	0.551%	438	15760	5340	0.554%
17	505	20591	442	0.242%	468	17595	3475	0.255%
18	535	21003	0	optimale	472	17767	3299	0.261%
19	535	21003	0	optimale	489	18236	2813	0.137%
20	535	21003	0	optimale	499	18844	2195	0.106%

TAB. 6.7 – *Efficacité des nouveaux critères de fixation : Inst535*

k	Fixations complètes				Fixations traditionnelles			
	somm.	arcs	NF	Garantie	somm.	arcs	NF	Garantie
5	1284	88542	0	optimale	1284	88542	0	optimale
6	1284	88542	0	optimale	1263	84662	3901	0.033%
7	1284	88542	0	optimale	1225	80289	8312	0.764%
8	1284	88542	0	optimale	1284	88542	0	optimale
9	1284	88542	0	optimale	1284	88542	0	optimale
10	1284	88542	0	optimale	1253	82431	6142	0.075%
11	1191	84078	4557	0.856%	1185	75901	12740	0.856%
12	1194	84451	4181	0.995%	1182	75251	13393	0.995%
13	1206	85255	3365	0.959%	1192	76076	12558	0.959%
14	1284	88542	0	optimale	1226	80630	7970	0.441%
15	1284	88542	0	optimale	1284	88542	0	optimale
16	1284	88542	0	optimale	1237	82561	6028	0.331%
17	1284	88542	0	optimale	1249	83926	4651	0.122%
18	1284	88542	0	optimale	1241	82799	5786	0.122%
19	1284	88542	0	optimale	1248	83318	5260	0.053%
20	1284	88542	0	optimale	1284	88542	0	optimale

TAB. 6.8 – *Efficacité des nouveaux critères de fixation : Inst1284*

k	Fixations complètes				Fixations traditionnelles			
	somm.	arcs	NF	Garantie	somm.	arcs	NF	Garantie
5	5535	666639	0	optimale	5535	666639	0	optimale
6	5535	666639	0	optimale	5535	666639	0	optimale
7	5535	666639	0	optimale	5535	666639	0	optimale
8	5535	666639	0	optimale	5507	647120	19547	0.164%
9	5535	666639	0	optimale	5504	646623	20047	0.199%
10	5535	666639	0	optimale	5494	643265	23415	0.206%
11	5535	666639	0	optimale	5535	666639	0	optimale
12	5535	666639	0	optimale	5535	666639	0	optimale
13	5535	666639	0	optimale	5492	647515	19167	0.078%
14	5437	654444	12293	0.295%	5383	611369	43129	0.389%
15	5444	656458	10272	0.305%	5373	601966	54563	0.406%
16	5316	634972	31886	0.588%	5265	570193	64830	0.588%
17	5433	654132	12609	0.292%	5276	574551	79738	0.613%
18	5535	666639	0	optimale	5393	611183	55598	0.338%
19	5371	645472	21331	0.426%	5334	592122	53387	0.481%
20	5272	631104	35798	0.583%	5251	570276	60849	0.583%

TAB. 6.9 – *Efficacité des nouveaux critères de fixation : Inst5535*

parfois (exemple : Inst5535, $k = 17$, la garantie passe de 0.292% à 0.613%). Ce phénomène s'explique par le fait que fixer une variable revient à réduire l'ensemble des solutions lagrangiennes réalisables. La valeur de la fonction objectif à l'optimum ne peut donc qu'augmenter. Elle augmente réellement lorsque la solution lagrangienne optimale n'est plus réalisable après les fixations. Cette explication est d'autant plus recevable que les valeurs GUB étaient, dans nos tests, identiques, que l'on utilise tous les critères de fixations ou non (pour ne pas alourdir les tableaux ci-dessus, nous n'avons pas rappelé ces valeurs).

6.6 LIMITES

Nous exposons dans cette section les limites de nos algorithmes lagrangiens et *Branch and Cut*.

Nous donnons d'abord un exemple d'instance pour laquelle l'algorithme de relaxations lagrangiennes ne nous a pas permis de fixer suffisamment de variables pour pouvoir la résoudre jusqu'à l'optimalité.

Puis, nous comparons les résultats de notre algorithme *Branch and Cut* avec

ceux obtenus en utilisant directement le logiciel CPLEX, version 6.0. Pour cela nous reprenons les trois instances précédemment citées, et pour chacune d'elles, les valeurs de k pour lesquelles l'optimalité de la solution n'avait pas été prouvée par les relaxations lagrangiennes.

6.6.1 INSTANCES NON RÉSOLUES

L'instance que nous présentons dans cette section possède 3773 sommets, 349524 arcs et un seul terminal.

Le graphique suivant montre, pour des valeurs de k comprises entre 5 et 14, le pourcentage de variables fixées à la fin de la boucle du dernier faisceau (histogramme gris, axe de gauche), et la garantie calculée (histogramme noir, axe de droite).

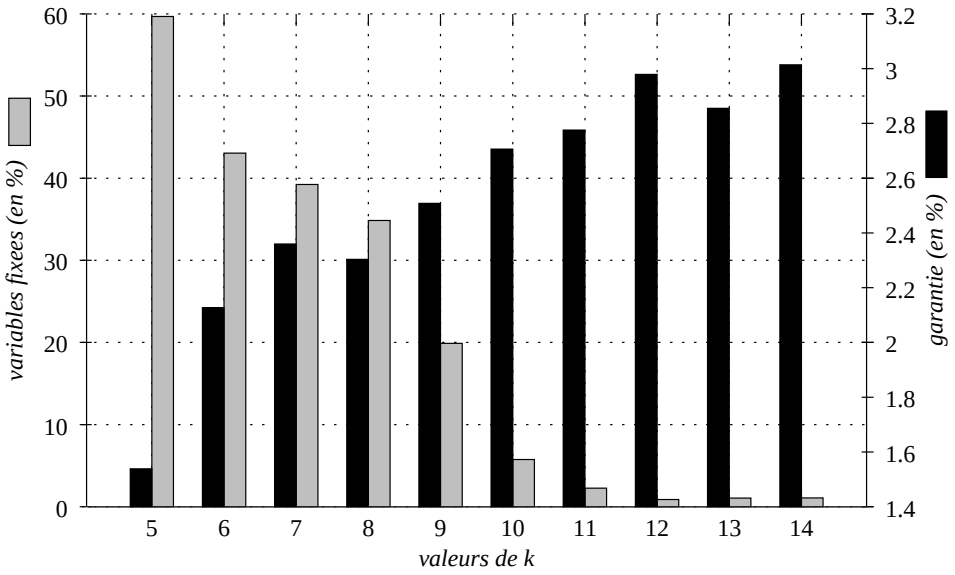


FIG. 6.1 – *Exemple d'une instance non résolue : Inst3773*

Excepté pour $k = 5$, nous n'avons jamais réussi à fixer plus de la moitié des variables de cette instance. À partir de $k = 12$, le pourcentage de variables fixées est de l'ordre de 1%, c'est-à-dire négligeable. Nous supposons que ces mauvais résultats sont dûs à la présence de plusieurs solutions optimales. En fait, les coûts unitaires de production sont ici très peu diversifiés : il existe de très nombreuses combinaisons (incomparables entre elles, mais pouvant remplacer les mêmes com-

binaisons) possédant des coûts identiques. Or, il est clair que notre méthode de fixation ne donne de bons résultats que si ces coûts sont réellement distincts.

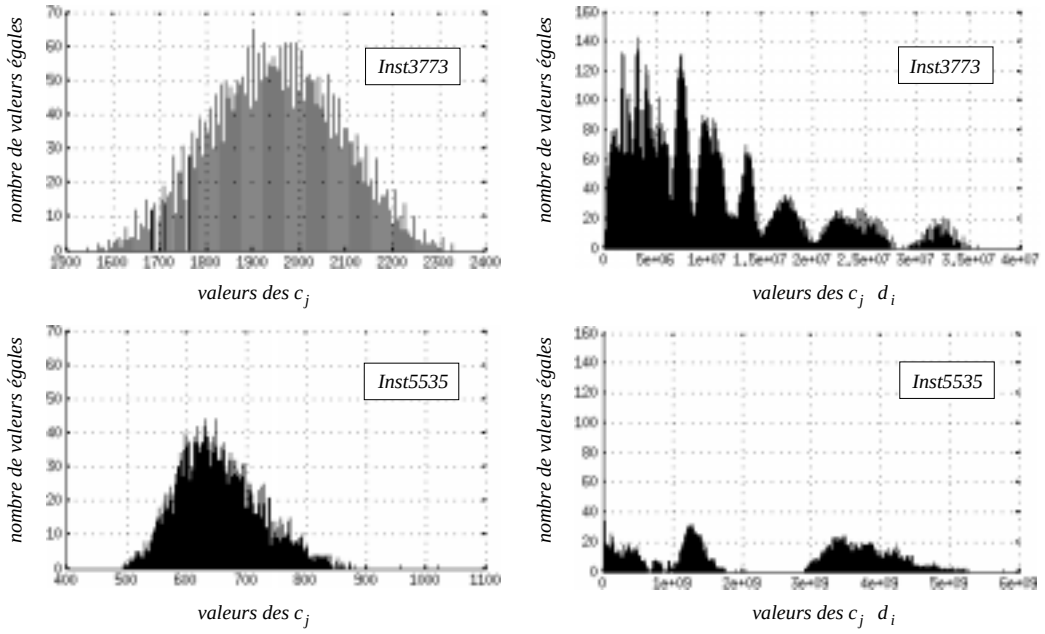


FIG. 6.2 – *Répartition des coûts de Inst3773 et Inst5535*

Les graphiques de la figure 6.2 représentent la répartitions de coûts unitaires de production c_j, et les coûts des substitutions $c_j d_i$ pour les instances Inst3773 et Inst5535. Nous n'avons pas donné la répartition des demandes, car celles-ci sont relativement distinctes les unes des autres, elles peuvent varier d'une centaine à plusieurs millions. Ces graphiques représentent pour chaque valeur possible de ces coûts, le nombre de combinaisons et de substitutions possédant ce coût. Nous pouvons remarquer que, comme nous l'avions supposé, les coûts de l'instance Inst3773 sont très peu diversifiés. Ceci est d'autant plus vrai que, malgré le fait que Inst5535 possède beaucoup plus de combinaisons et de substituions que Inst3773, le nombre maximum de combinaisons de Inst5535 possédant le même coût de production est de 44 alors qu'il est de 65 pour Inst3773. Cette observation est encore plus flagrante pour les coûts des substitutions : le nombre de substitutions maximum possédant le même coût est de 57 pour Inst5535, et de 143 pour Inst3773. À titre de comparaison, le tableau suivant synthétise la répartition des coûts pour toutes les instances citées dans ce chapitre. Nb est le nombre de coûts distincts des combinaisons (resp. des substitutions), et $NbMax$ représente le nombre maximum de combinaisons (resp. des substitutions) possédant le même

coût.

Instance	Combinaisons			Substitutions		
	$\|V\|$	Nb	$NbMax$	$\|A\|$	Nb	$NbMax$
Inst535	535	106	15	21 003	6 643	29
Inst1284	1 284	153	28	88 542	31 238	57
Inst3773	**3 773**	**137**	**65**	**349 524**	**30 027**	**143**
Inst5535	5 535	355	44	666 639	208 482	57

TAB. 6.10 – *Répartition des coûts*

Malgré tout, il faut noter que la garantie obtenue sur cette instance Inst3773 demeure très honorable, quelle que soit la valeur de k : elle est toujours inférieure à 3%.

6.6.2 COMPARAISONS AVEC UNE UTILISATION DIRECTE DE CPLEX

Le tableau suivant compare les performances de notre algorithme *Branch and Cut* et du logiciel CPLEX lorsque nous les appelons après l'algorithme des relaxations lagrangiennes. Ces tests ont été effectués sur les instances Inst535, Inst1284 et Inst5535, pour les valeurs de k pour lesquelles, bien sûr, l'optimalité de la solution n'a pas pu être prouvée par l'algorithme lagrangien.

Nous indiquons les nombres de nœuds explorés, le nombre de contraintes de cycles générées dans notre algorithme, ainsi que les temps CPU en secondes.

Au vu de tels résultats, il est légitime de s'interroger sur l'utilité d'un algorithme *Branch and Cut* pour résoudre jusqu'à l'optimalité le problème de la gestion de la diversité.

Cependant, les très bons résultats de CPLEX sont dûs en grande partie à l'efficacité de notre algorithme lagrangien. Des instances de grande taille, telle que Inst3773, qui n'ont pas pu être réduits par les relaxations lagrangiennes, ne peuvent pas être résolues avec ce logiciel, pour des raisons simples de consommation excessive de mémoire informatique.

Instance	k	Branch and Cut			CPLEX	
		# nœuds	# cycles	Temps	# nœuds	Temps
Inst535	15	5	14	3.87	19	0.33
	16	21	73	23.46	16	5.49
	17	5	65	2.55	6	0.16
Inst1284	11	77	295	1341.99	24	29.06
	12	35	104	422.00	10	17.79
	13	21	27	182.82	14	13.10
Inst5535	14	5	675	906.24	7	63.38
	15	5	726	777.60	3	44.65
	16	11	1158	18452	4	383.61
	17	5	932	1039.10	3	76.85
	19	9	4904	9288.76	9	326.96
	20	?	?	?	12	1066.48

TAB. 6.11 – *Comparaison de l'algorithme Branch and Cut avec CPLEX*

6.7 CONCLUSION

Nous avons vu dans ce chapitre les limites des deux algorithmes, lagrangiens et de type *Branch and Cut*, que nous avons développés.

Cependant, nous avons montré que les relaxations lagrangiennes, et surtout la méthode des fixations et les heuristiques que nous avons mis au point, permettent d'obtenir des solutions heuristiques d'excellente qualité, voire même presque toujours optimales, et de réduire la taille du problème, pour la majorité des instances testées, suffisamment pour prouver cette optimalité.

7

CONCLUSIONS ET PERSPECTIVES

Le sujet de cette thèse consistait à résoudre par un algorithme exact le problème de la gestion de la diversité. Ce problème, bien que possédant de nombreuses applications dans l'industrie, n'avait été que très peu été étudié jusqu'à aujourd'hui.

Dans un premier temps, nous avons mené une étude théorique sur ce problème, afin notamment de connaître sa complexité, et de déterminer la meilleure modélisation.

Nous avons modélisé ce problème à l'aide d'un programme linéaire en nombres entiers proche de ceux des problèmes de localisation k-médians.

Parallèlement à cette étude théorique, nous avons effectué un nombre conséquent de tests numériques sur des instances réelles. Ces tests nous ont conduit à exploiter la théorie des relaxations lagrangiennes afin de réduire la taille du problème.

Nous avons mis au point un algorithme lagrangien original dans le sens qu'il est optimisé en combinant avec succès deux méthodes connues : celle dite *du sous-gradient*, et celle *des faisceaux*.

Nous avons mené une étude approfondie des différentes voies nous permettant de fixer des variables. À notre connaissance, jamais une telle étude de réduction n'avait été poussée aussi loin.

Nous avons aussi élaboré des heuristiques exploitant les différents renseignements offerts par les relaxations lagrangiennes afin de construire des solutions réa-

lisables de notre problème. Les idées sur lesquelles sont basées certaines d'entre elles ne sont pas dédiées à la gestion de la diversité. Elles peuvent être reprises pour résoudre de nombreux autres problèmes.

L'algorithme lagrangien que nous avons développé a démontré son efficacité en fournissant presque toujours des solutions optimales. Dans le cas contraire, la garantie sur la qualité de ces solutions est toujours excellente.

Très souvent, nous avons prouvé, grâce à la théorie même des relaxations lagrangiennes, l'optimalité de ces solutions. Lorsque ce n'était pas le cas, nous avons fait appel à un algorithme d'énumération implicite des solutions, basé sur la génération de variables et de contraintes. Nous avons étudié le polyèdre associé à l'enveloppe convexe des solutions du programme linéaire en nombres entiers issus de la modélisation, et mis au point une heuristique pour générer des coupes.

Tous ces points ont été développés dans les différents chapitres de ce documents. Ils ont été implantés, et forment un algorithme original de type *Branch-and-Cut* initialisé par une procédure lagrangienne de réduction.

Les perspectives auxquelles nous nous intéressons sont au nombre de quatre.

La première est l'intégration de l'algorithme que nous avons développé dans un logiciel industriel, intégration que nous espérons mener à terme.

La seconde consiste à généraliser les méthodes de fixation de variables et à les tester sur des problèmes de localisation tels que les problèmes k-médians et les problèmes de localisation sans contrainte de capacité. Les premiers résultats que nous avons obtenus sont prometteurs.

La troisième est l'exploitation de nouveaux algorithmes, que nous nommons *Lagrange and Cut*, combinant la théorie des relaxations lagrangiennes et la génération de coupes. Ce type d'algorithmes consiste à résoudre les relaxations lagrangiennes jusqu'à ce que les faisceaux aient convergé. Ces derniers nous fournissant une solution optimale de la relaxation linéaire, nous générons des coupes afin d'éliminer cette solution si elle est fractionnaire. Nous relâchons ensuite ces nouvelles contraintes en leur affectant de nouveaux multiplicateurs lagrangiens. Nous réitérons ce processus jusqu'à ce que la solution soit entière, ou que plus aucune coupe ne puisse être générée.

Enfin, la quatrième, qui est suscitée principalement par la précédente, est une étude plus approfondie du polyèdre associé aux solutions réalisables du problème. Nous avons déjà exposé quelques résultats dans ce document. Cette étude est particulièrement délicate, mais elle mérite encore d'être approfondie afin de générer des coupes plus profondes permettant d'être intégrées très efficacement dans un algorithme *Lagrange and Cut*.

Bibliographie

[BC99] Barahona F. et Chudak F.A. – *Appoximation and complexity in numerical optimization : continuous and discrete problems*, chap. Solving large scale uncapacitated facility location problems. – Kluwer Academic Publishers, 1999. édité par Pardalos (P.M.).

[Bea93a] Beasley J.E. – Lagrangean heuristics for location problems. *European Journal of Operational Research*, vol. 65, 1993, pp. 383–399.

[Bea93b] Beasley J.E. – *Modern Heuristic Techniques for Combinatorial Problems*, chap. 6, Lagrangean Relaxation, pp. 243–303. – Blackwell Scientific Publications, 1993. édité par Reeves (C.R.).

[Ber70] Berge Claude. – *Graphes et Hypergraphes*. – Paris, Dunod, 1970.

[BJN+94] Barnhart C., Johnson E.L., Nemhauser G.L., Savelsberg M.W.P. et Vance P.H. – *Branch-and-Price : Column Generation for solving Huge Integer Programs*. – Rapport technique, Computational Optimization Center COC-94-03, Georgia Institute of Technology, Atlanta, 1994.

[BMM97] Balakrishnan A., Magnanti T.L. et Mirchandani P. – *Annotated bibliographies in combinatorial optimization*, chap. 18, Network Design, pp. 311–334. – John Wiley & Sons, 1997. édité par Dell'Amico (M.), Maffioli (F.) et Martello (S.).

[Bri95] Briant Olivier. – Les algorithmes branch and price. – 1995. Mémoire de D.E.A. de Recherche Opérationnelle, Grenoble.

[Cap91] Captivo M.E. – Fast primal and dual heuristics for the p-median location problem. *European Journal of Operational Research*, vol. 52, 1991, pp. 65–74.

[CB82] Christofides N. et Beasley J. – A tree search algorithm for the p-median problem. *European Journal of Operational Research*, vol. 10, 1982, pp. 196–204.

[CFN77] Cornuéjols G., Fisher M.L. et Nemhauser G.L. – On the uncapacitated location problem. *Annals of Discrete Mathematics*, 1977, pp. 163–177.

[CJPR83] Cho D.C., Johnson E.L., Padberg M. et Rao M.R. – On the uncapacitated plant location problem. I : valid inequalities and facets. *Mathematics of Operations Research*, vol. 8, n4, 1983, pp. 579–589.

[CN93] Clochard Jean Maurice et Naddef Denis. – Using path inequalities in a branch-and-cut code for the symmetric traveling salesman problem. *In : Proceedings on the Third IPCO Conference*, éd. par Wolsey Lawrence et Rinaldi Giovanni, pp. 291–311.

[CNW90] Cornuéjols G., Nemhauser G.L. et Wolsey L.A. – *Discrete location theory*, chap. 3 : the uncapacitated facility location problem, pp. 119–172. – John Wiley & Sons, 1990. édité par Mirchandani (P.B.) et Francis (R.L.).

[CPR83] Cho D.C., Padberg M.W. et Rao M.R. – On the uncapacitated plant location problem. II : facets and lifting theorems. *Mathematics of Operations Research*, vol. 8, n4, 1983, pp. 590–589.

[CT82] Cornuejols G. et Thizy J.-M. – Some facets of the simple plant location polytope. *Mathematical Programming*, vol. 23, 1982, pp. 50–74.

[DDS92] Desrochers M., Desrosiers J. et Solomon M. – A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, vol. 40, n2, 1992.

[Dor95] Dorhout B. – Solution of a tinned purchasing problem by lagrangean relaxation. *European Journal of Operational Research*, vol. 81, 1995, pp. 597–604.

[DSD84] Desrosiers J., Soumis F. et Desrochers M. – Routing with time windows by column generation. *Networks*, vol. 14, 1984, pp. 545–565.

[Erl78] Erlenkotter D. – A dual-based procedure for uncapacitated facility location. *Operations Research*, vol. 26, n6, 1978, pp. 992–1009.

[Geo74] Geoffrion A. M. – Lagrangean relaxation for integer programming. *Mathematical Programming Study*, vol. 2, 1974, pp. 82–114.

[GG61] Gilmore P.C. et Gomory R.E. – A linear programming approach to the cutting stock problem. *Operations Research*, vol. 9, 1961, pp. 849–859.

[GJ79] Garey M.R. et Johnson D.S. – *Computers and Intractibility, A Guide to the Theory of* NP*-Completeness*. – New York, W. H. Freeman and Company, 1979.

[GJR84] Grötschel M., Jünger M. et Reinelt G. – A cutting plane algorithm for the linear ordering problem. *Operations Research*, vol. 32, 1984, pp. 1195–1220.

[GLS81] Grötschel M., Lovász L. et Schrijver A. – The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, vol. 1, n 2, 1981, pp. 169–197.

[Gui80] Guignard M. – Fractional vertices, cuts and facets of the simple plant location problem. *Mathematical Programming*, vol. 12, 1980, pp. 150–162.

[HT91] Hassin R. et Tamir A. – Improved complexity bounds for location problems on the real line. *Operations Research Letters*, vol. 10, 1991, pp. 395–402.

[HUL93] Hiriart-Urruty J.B. et Lemaréchal C. – *Convex Analysis and Minimization Algorithms II*. – Berlin, Heidelberg, Springer, 1993, *Grundlehren der mathematischen Wissenschaften*, volume 306.

[JLM+95] Jones P.C., Lowe T.J., Muller G., Xu N., Ye Y. et Zydiak J.L. – Specially structured uncapacitated facility location problems. *Operations Research*, vol. 43, n4, 1995, pp. 661–669.

[JT96] Jünger Michael et Thienel Stefan. – *Basic Design Ideas for the Branch-and-Cut System ABACUS*. – Rapport technique, Universität zu Köln, 1996.

[JT97] Jünger Michael et Thienel Stefan. – *The Design of the Branch-and-Cut System ABACUS*. – Rapport technique, Universität zu Köln, 1997.

[KP83] Krarup J. et Pruzan P.M. – The simple plant location problem: Survey and synthesis. *European Journal of Operational Research*, vol. 12, n 1, 1983, pp. 36–81.

[LL97] Labbé M. et Louveaux F. V. – *Annotated bibliographies in combinatorial optimization*, chap. 16, Location problems, pp. 261–281. – John Wiley & Sons, 1997. édité par Dell'Amico (M.), Maffioli (F.) et Martello (S.).

[LS97] Lemaréchal C. et Sagastizábal C. – Variable metric bundle methods: from conceptual to implementable forms. *Mathematical Programming*, vol. 76, n3, 1997.

[MC79] Mulvey J.M. et Crowder H.P. – Cluster analysis: an application of lagrangean relaxation. *Management Science*, vol. 25, n4, April 1979, pp. 329–340.

[Mir90] Mirchandani P.B. – *Discrete location theory*, chap. 2 : The p-median problem and generalizations, pp. 55–118. – John Wiley & Sons, 1990. édité par Mirchandani (P.B.) et Francis (R.L.).

[NW88a] Nemhauser G.L. et Wolsey L.A. – *Integer and Combinatorial Optimization*, chap. II.6 : Applications of Special-Purpose Algorithms. – New York, NY, USA, John Wiley & Sons, 1988.

[NW88b] Nemhauser G.L. et Wolsey L.A. – *Integer and Combinatorial Optimization*, chap. I.4 : Polyhedral Theory. – New York, NY, USA, John Wiley & Sons, 1988.

[NW88c] Nemhauser G.L. et Wolsey L.A. – *Integer and Combinatorial Optimization*, chap. II.4 : General Algorithms. – New York, NY, USA, John Wiley & Sons, 1988.

[PR82] Padberg M. et Rao M. R. – Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, vol. 7, 1982, pp. 67–80.

[PR87] Padberg Manfred W. et Rinaldi Giovanni. – Optimization of a 532 city symmetric traveling salesman problem by branch-and-cut. *Operations Research Letters*, 1987, pp. 1–7.

[PR91] Padberg Manfred W. et Rinaldi Giovanni. – A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, vol. 33, 1991, pp. 60–100.

[PVW85] Padberg M.W., Van Roy T.J. et Wolsey L.A. – Valid linear inequalities for fixed charge problems. *Operations Research*, vol. 33, 1985, pp. 842–861.

[Sav93] Savelsberg M. – *A Branch-and-Price algorithm for the generalized assignement problem*. – Rapport technique, Computational Optimization Center COC-93-02, Georgia Institute of Technology, Atlanta, 1993.

[Sch86] Schrijver A. – *Theory of linear and integer programming*. – John Wiley & Sons, 1986.

[Thi95] Thienel Stefan. – *ABACUS - A Branch-And-CUt System*. – Thèse de Doctorat, Universität zu Köln, 1995.

[VBJN92] Vance P.H., Barnhart C., Johnson E.L. et Nemhauser G.L. – *Solving binary cutting stock problems by column generation and Branch-and-Bound*. – Rapport technique, Computational Optimization Center COC-92-09, Georgia Institute of Technology, Atlanta, 1992.

[VW85] Van Roy T.J. et Wolsey L.A. – Valid inequalities and separation for uncapacitated fixed charge networks. *Operations Research Letters*, vol. 4, n3, 1985, pp. 105–112.

[VW94] Vanderbeck F. et Wolsey L.A. – *An exact algorithm for IP column generation*. – Rapport technique, Core Discussion Paper, Center for Operations Research & Econometrics UCL, Avril 1994.

[Wol89] Wolsey L.A. – Submodularity and valid inequalities in capacitated fixed charge networks. *Operations Research Letters*, vol. 8, 1989, pp. 119–124.

Résumé :

Le problème de la gestion de la diversité est défini sur un ensemble partiellement ordonné d'élements possédant des demandes et des coûts unitaires de production. L'objectif est de produire un sous-ensemble de k éléments références, k étant un nombre donné, minimisant les coûts. Chaque élément non produit doit être remplacé par une référence qui lui est supérieure, ce qui implique un sur-coût.

Après une étude théorique de complexité, nous modélisons ce problème grâce à un programme linéaire en nombres entiers, proche de ceux des problèmes de localisation k-médians. Pour résoudre ce programme, nous présentons un algorithme lagrangien, ainsi que de nombreux critères de fixation de variables permettant de réduire la taille du problème. Nous exploitons ensuite cet algorithme pour construire des solutions de bonne qualité.

Nous développons enfin un algorithme exact de *Séparation et Coupe*. Nous étudions un certain type de coupes ainsi qu'une heuristique permettant de les générer. Nous concluons par des tests numériques effectués sur des instances réelles.

Mots clés : problème de localisation, relaxation lagrangienne, fixation de variable, algorithme de *Séparation et Coupe* , gestion de la diversité

Title : Theoretical and computational studies of the diversity management problem

Abstract :

The diversity management problem is defined on a partially ordered set. The objective is to produce a subset of k reference elements, k being a given number, while minimizing the cost. Demands and unit costs of production are known. Each non produced element must be replaced by a reference which is higher in the partial order, and this implies an overcost.

After a theoretical study on the complexity of this problem, we describe and justify the model we chose. This model is an integer linear program close to that commonly used for the k-median problem. To solve this program, we present a Lagrangean approach, and various criteria of variable fixing. We exploit also this approach to build feasible solutions of good quality.

Finally, we design a *Branch and Cut* type algorithm to solve our problem to optimality. To do so, we first do a polyhedral study of the convex hull of the solutions. We present a particular type of cuts that eliminate fractional solutions, as well as a heuristic to generate them. We conclude by numerical tests carried out on real world instances.

Keywords : plant location problem, lagrangean relaxation, variable fixing, *Branch and Cut* algorithm, diversity management problem