Problèmes de multicoupes minimales et de multiflots maximaux en nombres entiers.

Lucas Létocart

19 décembre 2002

A Sandrine.

Remerciements

Je voudrais, en tout premier lieu, remercier Marie-Christine Costa, ma directrice de thèse. Elle m'a aidé, soutenu et encouragé durant ces trois années. Sa disponibilité et ses conseils m'ont permis de découvrir la Recherche et elle a su me guider dans ce parcours initiatique. Je tiens également à remercier Frédéric Roupin, qui a co-encadré cette thèse, pour son aide et ses conseils. Marie-Christine, Frédéric, notre collaboration a été pour moi très fructueuse, je garderai le souvenir de réunions animées et constructives, dans lesquelles vous avez su me guider et m'orienter. Puissiez-vous trouver dans ces quelques lignes toute ma reconnaissance.

Monsieur Michel Minoux, Professeur à l'université Paris 6, et Monsieur Gérard Plateau, Professeur à l'université Paris 13, m'ont fait l'honneur et le plaisir d'accepter d'être rapporteurs de cette thèse. Pour leurs conseils et leurs remarques qui m'ont été précieux et pour l'intérêt qu'ils ont porté à mon travail de recherche, je tiens à les remercier.

Je remercie vivement Monsieur Alain Billionnet, Professeur à l'Institut d'Informatique d'Entreprise (CNAM), Madame Virginie Gabrel, Maître de Conférences à l'université Paris 9 et Monsieur Abdel Lisser, Professeur à l'université Paris 11, d'avoir accepté de faire partie de ce jury et de l'attention qu'ils ont porté à cette thèse.

Je souhaite également remercier les membres du laboratoire CEDRIC du CNAM. Ils ont facilité mon intégration par leur accueil et m'ont permis de travailler dans une ambiance cordiale. Je remercie plus particulièrement les membres de l'équipe Optimisation Combinatoire du CEDRIC pour leur soutien et leurs conseils.

Durant ces trois années, parsemées de moments inoubliables et de moments de doute, une personne était toujours à mes côtés et c'est cette personne que je souhaite remercier le plus. C'est toi, Sandrine, ma compagne de tous les instants, je te remercie pour m'avoir soutenu et aimé comme tu l'as fait.

Je remercie également ma mère pour son aide et son soutien, ainsi que ceux qui m'ont permis de m'éloigner de ce travail et qui, parfois, l'ont même retardé. Merci donc à Yves,

Béatrice, Madeleine, Delphine, Guillaume et Sacha pour leur soutien et leur amitié sans faille.
Julie, Capella, je ne vous oublie pas.

Table des matières

1 Introduction **15**

1.1 Formulation des problèmes de multiflot et de multicoupe 17

1.1.1 Formulation du problème de multiflot 17

1.1.2 Formulation du problème de multicoupe 19

1.2 Les problèmes connexes 20

1.2.1 Les problèmes de coupe et de flot multiterminaux 21

1.2.2 Le problème de la K-coupe 22

1.2.3 Maximisation des flots inséparables 23

1.2.4 Maximisation des multichemins 24

1.2.5 Maximisation du nombre de chemins disjoints par les arêtes 25

1.2.6 Autres problèmes connexes 26

1.2.6.1 Problèmes avec demandes 26

1.3 Applications 27

1.4 Organisation de la thèse 28

2 État de l'art **31**

2.1 Deux problèmes continus liés par dualité 31

2.2 Les méthodes de résolution du multiflot continu 33

2.2.1 Méthodes de décomposition 33

2.2.1.1 Décomposition par les prix 34

2.2.1.1.1 Relaxation Lagrangienne 34

2.2.1.1.2 Génération de colonnes 34

2.2.1.2 Décomposition par les ressources 34

2.2.2 Méthodes de partitionnement 35

2.2.2.1 Partitionnement primal 35

2.2.2.2 Partitionnement dual 35
2.3 Complexité et résolution des problèmes entiers dans des graphes quelconques . 36
2.3.1 Résolution exacte de $IMFP$ 36
2.3.2 Complexité de $IMFP$ et $IMCP$ 39
2.3.3 Approximation de $IMFP$ et $IMCP$ 39
2.3.4 Complexité et approximation des problèmes connexes 40
2.3.4.1 Les problèmes de coupe multiterminaux 40
2.3.4.2 Les problèmes de chemins disjoints par les arêtes et de flots inséparables 42
2.4 Graphes particuliers 42
2.4.1 Graphes orientés sans circuit 42
2.4.2 Arbres 43
2.4.2.1 Arbres non orientés 43
2.4.2.2 Arbres bi-orientés 45
2.4.3 Graphes bipartis 45
2.4.4 Graphes planaires 46
2.4.5 Anneaux 47

3 Résolution des problèmes dans les arbres orientés 49
3.1 Complexité de $IMFP$ et $IMCP$ dans les arbres orientés 49
3.2 Un algorithme polynomial pour $IMFP$ et $IMCP$ dans les arborescences 52
3.2.1 Recherche d'un multiflot entier 52
3.2.2 Recherche d'une multicoupe 53
3.2.3 Exemple 54
3.2.4 Optimalité de l'algorithme 56
3.2.5 Complexité de l'algorithme 57

4 Résolution de $IMFP$ et $IMCP$ dans les arbres non orientés 63
4.1 Orientation d'un arbre 63
4.2 Résolution de $IMCP$ 64
4.2.1 Notations, définitions 65
4.2.2 Construction du programme SDP associé à ($IMCP$) 65
4.2.3 Comparaison des bornes de la programmation linéaire et de la SDP . . . 67
4.2.4 Utilisation de la SDP dans un branch-and-bound 69
4.2.4.1 Branchement et bornes 69

4.2.4.2 Résultats numériques 70
4.2.4.3 Améliorations 72
4.3 Résolution de $IMFP$ 73
4.3.1 Programmation linéaire en nombres entiers 73
4.3.2 Programmation semi-définie 74

5 Résolution des problèmes dans les anneaux 75
5.1 Simplifications 75
5.2 Polynomialité des problèmes de coupe et de flot multiterminaux 77
5.3 $IMCP$ dans les anneaux 78
5.3.1 Introduction 78
5.3.2 Calcul d'une multicoupe C_j^* 79
5.3.3 Multicoupe minimale sur l'anneau 80
5.3.4 Exemple 80
5.3.5 Optimalité de l'algorithme 82
5.3.6 Complexité de l'algorithme 82
5.4 $IMFP$ dans les anneaux 83

6 Conclusion 85

Table des figures

1.1 Un exemple avec 3 flots. 19
1.2 Solutions de $(IMFP)$ et $(IMCP)$ pour l'exemple avec 3 flots. 20
1.3 Un exemple avec un nombre de terminaux égal à 4. 21
1.4 Solutions de FLOT MULTITERM et COUPE MULTITERM pour l'exemple avec 4 terminaux. 22
1.5 Un exemple avec 6 sommets dont une solution optimale de K-COUPE avec $K = 5$ a pour valeur 17. 23
1.6 Solution de FLOTINSEP MAX pour l'exemple avec 3 flots. 24
1.7 Solution de CAPCHEM MAX pour l'exemple avec 3 flots. 25
1.8 Exemple pour CHEMDISJARE MAX. 26
1.9 Transformation d'un multiflot avec demandes en un multiflot maximal. 27

2.1 Un exemple de biflot. 37
2.2 Transformation d'un biflot en 2 flots simples. 37
2.3 Un exemple pour $IMFP$ dans un arbre. 43
2.4 Un exemple d'instance pour $IMFP$ dans un graphe biparti augmenté. 46

3.1 Un exemple pour $IMFP$ dans un arbre orienté et la matrice des contraintes associée. 50
3.2 Application de l'algorithme 55
3.3 Preuve du Lemme 3.2 : pas i avec $p_k \bigcap p_i \neq \emptyset$ 58
3.4 $K = O(n^2)$: structure de données associée à la figure 3.2 59
3.5 Solution pour $IMFP$ dans un arbre orienté 62

4.1 Instances faciles et difficiles dans des arbres non orientés. 64
4.2 $IMCP$ avec $K = 3$ dans une étoile. 68

5.1 Orientation d'un anneau. 76
5.2 Simplification dans un anneau. 77
5.3 Instance pour COUPE MULTITERM et FLOT MULTITERM sur un anneau orienté. 78
5.4 Solution pour COUPE MULTITERM et FLOT MULTITERM sur un anneau orienté. 79
5.5 Décomposition d'un anneau en 2 chaînes. 81
5.6 Résolution de $IMFP$ et $IMCP$ sur les 2 chaînes. 81
5.7 Instance pour laquelle $IMCP$ se résout en $O(n^3)$. 83
5.8 Instance pour $IMFP$ et $IMCP$ dans un anneau orienté. 84

Liste des tableaux

4.1 Contraintes d'un programme linéaire et contraintes du programme SDP correspondant 66
4.2 Récapitulatif des principaux résultats pour $IMCP$ dans les arbres non orientés 70
4.3 $IMCP$: Résultats numériques 1. 71
4.4 $IMCP$: Résultats numériques 2. 72
4.5 $IMFP$: Résultats numériques. 74

6.1 Principaux résultats de complexité et d'approximation. 88

Chapitre 1

Introduction

L'objet de cette thèse est l'étude et la résolution de certains problèmes d'optimisation combinatoire dans certains types de graphes : les problèmes de multiflots entiers maximaux et de multicoupes minimales, ainsi que plusieurs problèmes connexes.

Les problèmes de coupe et de flot se sont tout d'abord posés dans la pratique. Les premières études des problèmes de multiflot et de multicoupe datent de la fin des années 1950 et du début des années 1960 (voir [27] et [49]), et font suite à la découverte d'un algorithme polynomial par Ford et Fulkerson [28] pour les problèmes de flot simple, c'est-à-dire les problèmes de flot maximal et de coupe minimale. Compte tenu de la difficulté des problèmes à plusieurs flots, très peu de résultats nouveaux ont été trouvés pour les problèmes entiers au cours des années 1970 à 1980. C'est grâce à l'essor de l'informatique, et aux progrès réalisés en programmation linéaire, que l'on a pu envisager la résolution des problèmes de multiflot continu, pour lesquels de nombreuses méthodes de résolution sont apparues à cette époque. Les chercheurs se sont de nouveau intéressés aux problèmes de multicoupe et de multiflot en nombres entiers au début des années 1990, avec principalement de nombreux résultats de complexité. Dalhaus, Johnson, Papadimitriou, Seymour et Yannakakis [23] d'une part, et Garg, Vazirani et Yannakakis [35] d'autre part, ont montré que les problèmes de multicoupe et de multiflot en nombres entiers sont NP-difficiles et Max SNP-difficiles. Le fait qu'il n'existe très probablement pas d'algorithme de complexité polynomiale permettant de résoudre un problème NP-difficile incite généralement à utiliser pour sa résolution une méthode heuristique ou une méthode énumérative, et il aura presque fallu attendre le 21^{eme} siècle pour voir une tentative de résolution exacte pour le problème du multiflot maximal en nombres entiers (voir [9]). La plupart des résultats connus, en ce qui concerne la complexité, l'approximation ou

encore la résolution des problèmes de multiflot et de multicoupe sont donc très récents.

Nous verrons que la plupart des problèmes présentés dans cette thèse sont des problèmes difficiles à résoudre dans des graphes quelconques. Ce sont également des problèmes pour lesquels il peut être difficile d'obtenir un algorithme approché, avec une performance fixe garantie, en temps polynomial. Les particularités des graphes, ainsi que l'orientation ou non de ces graphes, facilitent parfois la résolution ou l'approximation des problèmes. Mais nous verrons également que les problèmes de multiflot et de multicoupe peuvent rester très difficiles même dans des graphes très simples.

Dans la pratique, de nouveaux problèmes de flot et de coupe, que nous appelons problèmes connexes, sont apparus et leur étude a permis d'obtenir de nombreux nouveaux résultats. C'est en étudiant les modèles associés à ces problèmes que les chercheurs ont tenté de se rapprocher des modèles de multicoupe et de multiflot en nombres entiers.

Dans cette introduction, nous donnons les différentes formulations des problèmes de multicoupe minimale et de multiflot maximal en nombres entiers. Nous présentons également plusieurs problèmes connexes aux problèmes de flots et de coupes. L'un de ceux qui a été le plus étudié est sans doute le problème de coupe multiterminaux qui est un problème pour lequel il existe de nombreux résultats. A ce problème nous associons le problème de flot multiterminaux qui a été au contraire très peu étudié. Nous introduisons aussi d'autres cas particuliers du problème de multiflot : les problèmes de flots inséparables, de multichemins et de chemins disjoints.

Nous proposons, dans le chapitre 2, un état de l'art concernant les problèmes présentés dans l'introduction. Ce tour d'horizon des principaux résultats de complexité va nous permettre de mieux appréhender la difficulté de ces problèmes selon le type de graphe dans lequel nous cherchons à les résoudre.

Dans le chapitre 3, nous nous focaliserons sur les arbres. Nous montrerons que les problèmes de multicoupe minimale et de multiflot maximal en nombres entiers sont polynomiaux si l'arbre est orienté. Nous proposons également un algorithme polynomial pour ces deux problèmes dans le cas encore plus particulier des arborescences.

Ces résultats ne peuvent pas s'appliquer sur les arbres non orientés car les problèmes de multicoupe et de multiflot en nombres entiers sont NP-difficiles dans les arbres non orientés. Nous proposons, dans le chapitre 4, une nouvelle approche de résolution pour le problème difficile de la multicoupe minimale dans les arbres non orientés. On sait que l'efficacité des méthodes énumératives est fortement liée à la qualité des bornes que l'on est capable de déterminer. Nous utilisons la programmation linéaire et la programmation semi-définie dans un

algorithme de séparation et d'évaluation afin de résoudre exactement le problème de multicoupe dans des arbres non orientés pour des instances de grande taille.

Nous présentons enfin, dans le chapitre 5, de nouveaux résultats dans les anneaux. Nous proposons un algorithme polynomial pour les problèmes de coupe et flot multiterminaux dans les anneaux ainsi qu'un algorithme polynomial pour le problème de multicoupe minimale.

1.1 Formulation des problèmes de multiflot et de multicoupe

Nous utilisons la terminologie de Berge [7] pour la définition des graphes. Soit $G = (V, E)$ un graphe non orienté avec une capacité (ou poids) positive et entière u_e sur chaque arête e de E et soit une liste de K paires de sommets $\{s_k, t_k\}$, $k = 1, .., K$. On note $n = |V|$ et $m = |E|$. On associe un flot à chaque paire de sommets $\{s_k, t_k\}$. Le problème du multiflot maximal en nombres entiers, que nous appellerons dans la suite $IMFP$ (Integral Maximum Multicommodity Flow Problem), consiste à maximiser la somme des valeurs des flots entiers F_k routés de s_k à t_k, en respectant les contraintes de capacité et de conservation de flot. Les contraintes de conservation de flot impliquent que pour chaque flot F_k, $k = 1, .., K$, la quantité de flots qui arrive en un sommet est égale à la quantité de flots qui part de ce sommet, sauf en s_k et en t_k, il s'agit de la loi de conservation aux nœuds. Le problème de la $s_k - t_k$ multicoupe minimale, que nous appellerons dans la suite $IMCP$ (Integral Minimum MultiCut Problem), consiste à trouver un ensemble d'arêtes de poids minimal dont le retrait coupe chaque chaîne menant de s_k à t_k, $k = 1, .., K$. Ces deux problèmes sont connus pour être NP-difficiles et Max SNP-difficiles pour $K \geq 3$ dans des graphes quelconques ([23] et [35]).

1.1.1 Formulation du problème de multiflot

Le problème du multiflot entier maximal $IMFP$ peut être formulé de deux manières différentes. La plus couramment utilisée est la formulation arête-flot où l'on associe une variable $f_{i,j}^k$ pour chaque valeur du flot k sur chaque arc (i, j). Le programme mathématique $(IMFP)$ (I) associé à cette formulation est le suivant (dans un graphe orienté) :

$$(IMFP) \quad (I) \quad \left| \begin{array}{lll} \text{Max} & \sum_k F^k & \\ \text{s.c.} & \sum_j f^k_{i,j} - \sum_j f^k_{j,i} = \left\{ \begin{array}{ll} F^k & i = s_k \\ 0 & i \neq s_k, t_k \\ -F^k & i = t_k \end{array} \right. & \forall (i,j) \in E,\ \forall k = 1,..,K \ (1) \\ & \sum_k f^k_{i,j} \leq u_{i,j} & \forall (i,j) \in E \ (2) \\ & f^k_{i,j} \in N & \forall (i,j) \in E,\ \forall k = 1,..,K \end{array} \right.$$

Les contraintes (1) sont les contraintes de conservation de flot. Les contraintes (2) impliquent que la somme des flots routés sur une arête soit inférieure à la capacité de cette arête. Les dernières contraintes impliquent que les variables sont entières.

Une autre formulation possible est la formulation arête-chaîne dans laquelle on considère toutes les chaînes menant de s_k à t_k. Soit P^k l'ensemble de toutes les chaînes élémentaires de s_k à t_k. On note $\Pi = \bigcup_{k=1,..,K} P^k$ et M le cardinal de Π.

On note f_i la valeur du flot circulant sur la i^{eme} chaîne p_i, $i = 1,..,M$. Le problème du multiflot entier maximal $IMFP$ consiste à maximiser la somme des f_i en respectant les contraintes de capacité et peut être formulé de la manière suivante :

$$(IMFP) \quad (II) \quad \left| \begin{array}{lll} \text{Max} & \sum_{i=1}^{M} f_i & \\ \text{s.c.} & \sum_{i\ t.q.\ e \in p_i} f_i \ \leq u_e & \forall e \in E \qquad (3) \\ & f_i \in N & \forall i = 1,..,M \end{array} \right.$$

Les contraintes (3) impliquent que la somme des flots passant sur une arête soit inférieure à la capacité de l'arête et les contraintes suivantes imposent aux variables d'être entières.

Dans un graphe quelconque, cette formulation implique davantage de variables que la première formulation. En effet, dans la formulation (I), le nombre de variables et le nombre de contraintes sont égaux à mK, tandis que dans la formulation $(IMFP)$ (II), le nombre de variables peut être exponentiel, car il correspond au nombre de chaînes menant d'une source au puits correspondant et cela pour tous les flots ; le nombre de contraintes dans la formulation (II) est égal à m. Mais la formulation $(IMFP)$ (II) n'a pas que des inconvénients : d'une part, le problème du multiflot se présente comme un programme linéaire à structure bloc-angulaire, ce qui facilite l'emploi de certaines méthodes de résolution de la relaxation continue, et d'autre part cette formulation permet de mieux appréhender le problème pour la recherche d'algorithmes dans les graphes. Enfin, elle permet une vision simple de la relation de dualité

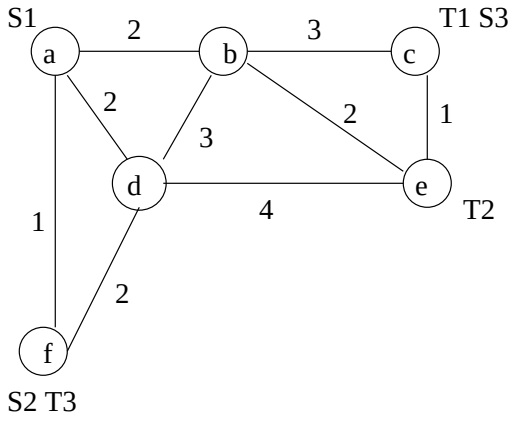


FIG. 1.1 – Un exemple avec 3 flots.

existant entre les problèmes de multicoupe et de multiflot comme nous le verrons dans le section 2.1.

Dans la suite de cette thèse, à chaque fois que nous ferons référence à la formulation de $IMFP$, il s'agira de la formulation $(IMFP)$ (II) (sauf indication contraire), i.e. de la formulation arête-chaîne du problème.

Exemple : Un exemple de $IMFP$ dans un graphe est donné dans la figure 1.1. Une solution optimale de $IMFP$ sur cet exemple est la suivante :

On route 2 unités du flot F_1 sur la chaîne (a,b,c).

On route 1 unité du flot F_1 sur la chaîne (a,d,e,c).

On route 2 unité du flot F_2 sur la chaîne (f,d,e).

On route 1 unité du flot F_2 sur la chaîne (f,a,d,e).

Notons $v(F_i)$ la valeur du flot F_i, $v(F^*)$ la valeur du flot optimal et F^* le vecteur solution, on obtient :

$v(F_1) = 3$, $v(F_2) = 3$, $v(F_3) = 0$, soit $F^* = \{3, 3, 0\}$ et $v(F^*) = 6$.

1.1.2 Formulation du problème de multicoupe

Le problème de la $s_k - t_k$ multicoupe $IMCP$ consiste à couper chaque chaîne reliant s_k et t_k, $k = 1, .., K$, en sélectionnant au moins une arête par chaîne p_k, p_k étant défini comme dans le paragraphe précédent pour la formulation $(IMFP)$ (II), de manière à minimiser la somme des poids des arêtes sélectionnées. Il peut-être formulé de la manière suivante :

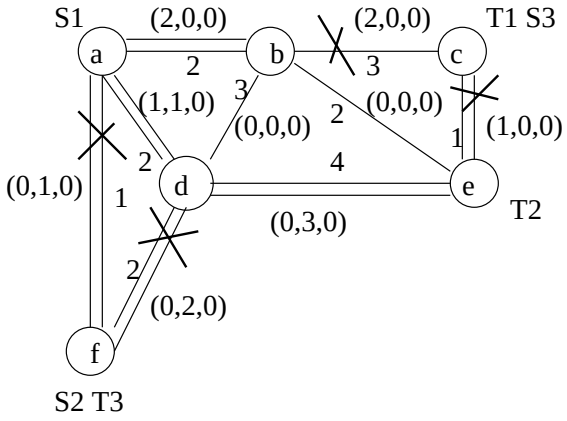


FIG. 1.2 – Solutions de $(IMFP)$ et $(IMCP)$ pour l'exemple avec 3 flots.

$$(IMCP) \quad \left| \begin{array}{lll} \text{Min} & \sum_{e \in E} u_e \, c_e & \\ \text{s.c.} & \sum_{e \in p_i} c_e \geq 1 & \forall i = 1, .., M \\ & c_e \in \{0,1\} & \forall e \in E \end{array} \right. \qquad (4)$$

où $c_e = 1$ si et seulement si l'arête e appartient à la coupe.

Les contraintes (4) impliquent qu'au moins une arête est coupée sur chaque chaîne menant d'une source à un puits. Les contraintes suivantes imposent que les variables sont booléennes, c'est-à-dire que l'arête est coupée entièrement ou n'est pas coupée du tout.

L'exemple proposé dans la section précédente (section 1.1.1) dans la figure 1.1 fournit également une instance pour $IMCP$. Une solution optimale pour $IMCP$ sur cette instance est la suivante :

Notons $v(C^*)$ la valeur de la coupe optimale C^*.

$C^* = \{(a,f),(d,f),(b,c),(c,e)\}$ et $v(C^*) = 7$. Rappelons que $v(F^*) = 6$ (voir paragraphe 1.1.1).

Les solutions du multiflot et de la multicoupe sont présentées dans la figure 1.2.

1.2 Les problèmes connexes

Pour $K = 1$ les deux problèmes précédents sont les problèmes bien connus de la coupe minimale et du flot maximal que l'on peut résoudre en temps polynomial [28].

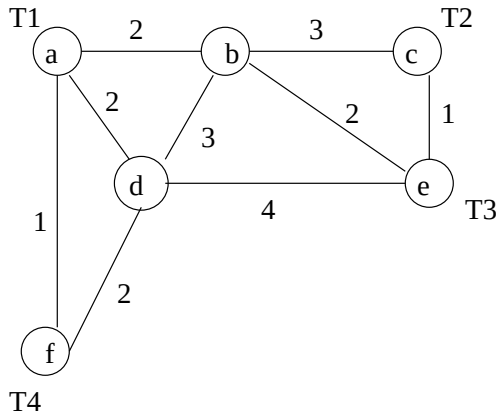


FIG. 1.3 – Un exemple avec un nombre de terminaux égal à 4.

Nous présentons ici les principaux problèmes connexes à $IMFP$ et $IMCP$.

1.2.1 Les problèmes de coupe et de flot multiterminaux

Un cas particulier très connu du problème de multicoupe est le problème de la coupe multiterminaux, que nous notons COUPE MULTITERM, où la coupe doit séparer deux à deux les sommets appartenant à un ensemble donné $X \subseteq V$. Notons $CardX = |X|$, il s'agit d'un problème particulier de multicoupe pour lequel les K paires à séparer dans la multicoupe sont les $\frac{1}{2}|X|(|X|-1)$ paires de sommets de X. Nous considérons également le problème de flot multiterminaux, que nous notons FLOT MULTITERM, qui est le multiflot entier maximal associé au problème de COUPE MULTITERM : l'objectif est de maximiser la somme totale des flots routés entre toutes les paires de sommets de X. Les programmes mathématiques restent ici les mêmes que ($IMCP$) et ($IMFP$), seules les données des problèmes changent : à chaque paire de sommets de X est associé un couple $\{s_k, t_k\}$.

Exemple : Un exemple de COUPE MULTITERM et de FLOT MULTITERM peut être vu dans la figure 1.3.

Une solution optimale pour le FLOT MULTITERM est la suivante :

On route 1 unité de flot entre T_1 et T_4 sur la chaîne (a,f).

On route 2 unités de flot entre T_1 et T_3 sur la chaîne (a,d,e).

On route 2 unités de flot entre T_3 et T_4 sur la chaîne (e,d,f).

On route 2 unités de flot entre T_1 et T_2 sur la chaîne (a,b,c).

On route 1 unité de flot entre T_2 et T_3 sur la chaîne (c,b,e).

On route 1 unité de flot entre T_2 et T_3 sur la chaîne (c,e).

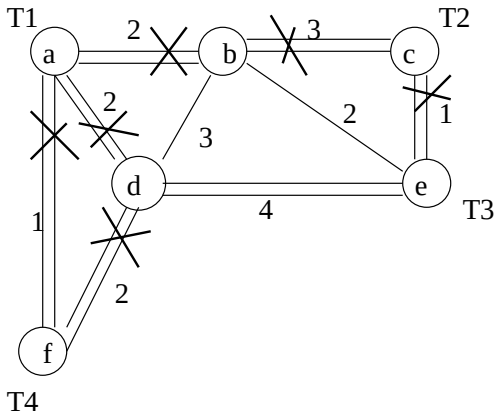


FIG. 1.4 – Solutions de FLOT MULTITERM et COUPE MULTITERM pour l'exemple avec 4 terminaux.

Notons $v(\Phi_{ij})$ la valeur du flot Φ_{ij}, entre le terminal i et le terminal j, $v(\Phi^*)$ la valeur du flot multiterminaux optimal et Φ^* le vecteur solution, on obtient :

$v(\Phi_{12}) = 2$, $v(\Phi_{13}) = 2$, $v(\Phi_{14}) = 1$, $v(\Phi_{23}) = 2$, $v(\Phi_{24}) = 0$, $v(\Phi_{34}) = 2$, soit $\Phi^* = \{2, 2, 1, 2, 0, 2\}$ et $v(\Phi^*) = 9$.

Notons $v(\Psi^*)$ la valeur de la coupe optimale Ψ^*.

$\Psi^* = \{(a, b), (a, d), (a, f), (b, c), (c, e), (d, f)\}$ et $v(\Psi^*) = 11$.

Les solutions du flot multiterminaux et de la coupe multiterminaux sont présentées dans la figure 1.4.

1.2.2 Le problème de la K-coupe

Un cas encore plus particulier du problème de la multicoupe minimale est le problème de la K-COUPE. Rappelons que l'objectif de ce problème est de partitionner les n sommets du graphe en K paquets non vides de manière à minimiser la somme des poids des arêtes permettant ce partionnement des sommets. Pour un K fixé, cela devient un cas particulier du problème de COUPE MULTITERM : si l'on considère tous les sous-ensembles de K sommets, alors nous pouvons résoudre toutes les instances correspondantes de COUPE MULTITERM, pour ne garder que la meilleure coupe trouvée. De cette manière, nous obtenons une solution optimale de K-COUPE car il existe alors K sommets appartenant à des paquets différents.

Exemple : Un exemple de K-COUPE peut-être vu dans la figure 1.5. Pour $K = 5$, une solution optimale de valeur 17 est décrite.

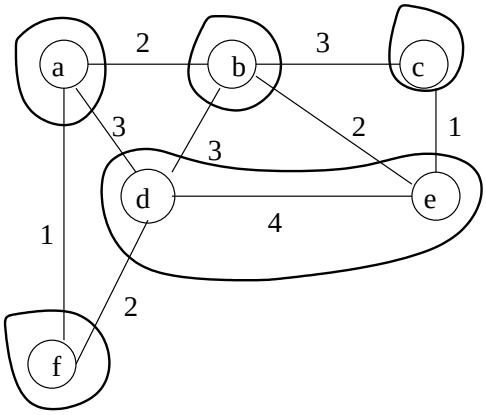


FIG. 1.5 – Un exemple avec 6 sommets dont une solution optimale de K-COUPE avec $K = 5$ a pour valeur 17.

1.2.3 Maximisation des flots inséparables

Généralement un flot F_k est routé sur différentes chaînes de P_k, c'est-à-dire sur différentes chaînes reliant s_k et t_k dans le graphe. Mais le problème requiert parfois que chaque flot F_k soit inséparable : F_k doit alors être routé sur une unique chaîne de P_k. Le problème FLOTINSEP MAX consiste à sélectionner K chaînes, un dans chaque ensemble P_k, et à router K flots vérifiant les contraintes de capacité de manière à maximiser la somme des flots. Pour obtenir le programme mathématique associé nous devons ajouter la contrainte quadratique suivante au programme $(IMFP)$:

$$f_i f_j = 0, \forall p_i \neq p_j \in P_k, \forall k = 1, .., K \quad (5)$$

Si le graphe est un arbre, il n'existe qu'une seule chaîne entre deux sommets et les contraintes (5) deviennent sans objet. Remarquons que le problème FLOTINSEP MAX dans un arbre n'est autre que le problème du multiflot entier maximal $IMFP$.

Si l'on souhaite appliquer le problème de FLOTINSEP MAX sur l'exemple de la section 1.1.1, il ne faut utiliser qu'une seule chaîne pour router chaque flot (voir figure 1.1). Une solution optimale pour FLOTINSEP MAX est la suivante :

On route 2 unités du flot F_1 sur la chaîne (a,b,c).

On route 2 unités du flot F_2 sur la chaîne (f,d,e).

On route 1 unité du flot F_3 sur la chaîne (f,a,d,e,c).

Notons comme précédemment $v(F_i)$ la valeur du flot F_i, $v(F^*)$ la valeur du flot optimal

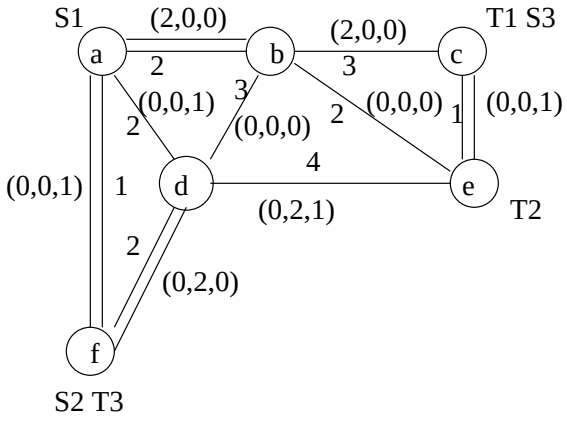


FIG. 1.6 – Solution de FLOTINSEP MAX pour l'exemple avec 3 flots.

et F^* le vecteur solution, on obtient :

$v(F_1) = 2$, $v(F_2) = 2$, $v(F_3) = 1$, soit $F^* = \{2, 2, 1\}$ et $v(F^*) = 5$.

Cette solution optimale de FLOTINSEP MAX est présentée dans la figure 1.6.

1.2.4 Maximisation des multichemins

Le problème CAPCHEM MAX consiste à maximiser le nombre total de chaînes reliant les paires de sommets sachant que le nombre total de chaînes pouvant passer sur chaque arête ne peut excéder la capacité de celle-ci. Pour obtenir le programme en $0-1$ associé, nous devons remplacer les contraintes $f_i \in N$ de $(IMFP)$ par les contraintes booléennes suivantes :

$$f_i \in \{0, 1\} \quad \forall i = 1, .., M$$

L'application de ce problème peut également s'effectuer sur l'exemple de la section 1.1.1, figure 1.1, en utilisant les conditions précédentes. Une solution optimale pour CAPCHEM MAX est la suivante :

On route 1 unité du flot F_1 sur la chaîne (a,b,c).

On route 1 unité du flot F_1 sur la chaîne (a,d,b,c).

On route 1 unité du flot F_1 sur la chaîne (a,d,e,b,c).

On route 1 unité du flot F_2 sur la chaîne (f,d,e).

On route 1 unité du flot F_2 sur la chaîne (f,a,b,e).

On route 1 unité du flot F_3 sur la chaîne (c,e,d,f).

En notant comme précédemment $v(F^*)$ la valeur du flot optimal , on obtient :

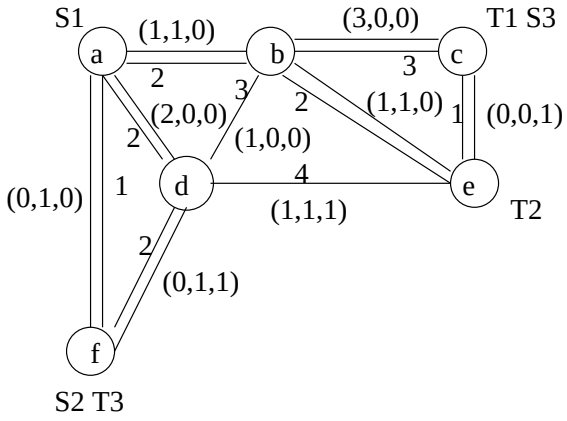


FIG. 1.7 – Solution de CAPCHEM MAX pour l'exemple avec 3 flots.

$v(F^*) = 6$ car nous avons relié les 3 paires de sommets par 6 chaînes différentes.

Cette solution optimale de CAPCHEM MAX est présentée dans la figure 1.7.

1.2.5 Maximisation du nombre de chemins disjoints par les arêtes

Considérons $IMFP$ avec toutes les capacités sur les arêtes égales à 1 : $u_e = 1$, pour tout $e \in E$. Nous obtenons le problème CHEMDISJARE MAX qui consiste à maximiser le nombre total de chaînes reliant les K paires de sommets de manière à ce que chaque arête n'appartienne qu'à une seule chaîne. Tous les flots f_i sont alors égaux à 0 ou 1 et le problème est un cas particulier de $IMFP$.

Exemple : Un exemple avec des capacités unitaires où l'on peut appliquer le problème de CHEMDISJARE MAX est donné dans la figure 1.8.

Une solution optimale pour CHEMDISJARE MAX est la suivante :

On route 1 unité du flot F_1 sur la chaîne (a,b,c).

On route 1 unité du flot F_2 sur la chaîne (f,d,e).

On route 1 unité du flot F_3 sur la chaîne (c,e,b,d,a,f).

Notons comme précédemment $v(F_i)$ la valeur du flot F_i, $v(F^*)$ la valeur du flot optimal et F^* le vecteur solution, on obtient :

$v(F_1) = 1$, $v(F_2) = 1$, $v(F_3) = 1$, soit $F^* = \{1, 1, 1\}$ et $v(F^*) = 3$.

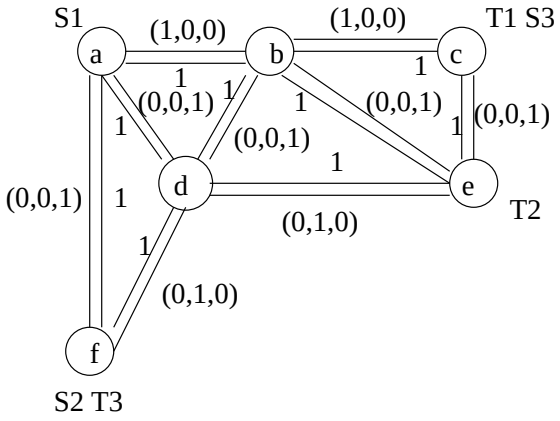


FIG. 1.8 – Exemple pour CHEMDISJARE MAX.

1.2.6 Autres problèmes connexes

De nombreux autres problèmes peuvent être reliés aux problèmes de multiflot et de multicoupe, comme les problèmes de multiflot entier à coût minimal. Nous n'aborderons pas ces problèmes car les méthodes de résolution diffèrent des méthodes utilisées pour résoudre les problèmes de multiflot maximal. En effet, la plupart de ces méthodes utilisent la programmation linéaire. Cependant on peut se référer à [4], [5], [37] ou [71] pour plus de précisions sur ces problèmes de minimisation.

Le sujet de cette thèse étant l'étude des problèmes entiers, nous ne traiterons que très brièvement des problèmes continus, dont des états de l'art très complets peuvent être trouvés dans [1] et [40]. Un aperçu des différentes méthodes de résolution des problèmes continus sera donné dans la section 2.2.

1.2.6.1 Problèmes avec demandes

Pour un problème de multiflot avec demandes, si l'on désire uniquement savoir si toutes les demandes peuvent être satisfaites ou si l'on souhaite maximiser le multiflot borné par ces demandes, alors le problème devient un problème de multiflot maximal classique, comme on peut le voir sur la figure 1.9. En effet, on peut ajouter au graphe K sommets, $v_1, .., v_k$ et K arêtes (ou arcs dans un graphe orienté) (t_k, v_k) dont les capacités sont égales à la demande d_k pour tout $k = 1, .., K$. Ensuite il suffit de déplacer t_k de son sommet initial vers l'autre extrémité de la nouvelle arête, i.e. en v_k. Dans certains cas, comme dans le cas où le graphe est un anneau, le graphe peut ne pas conserver ses propriétés de base après ces modifications.

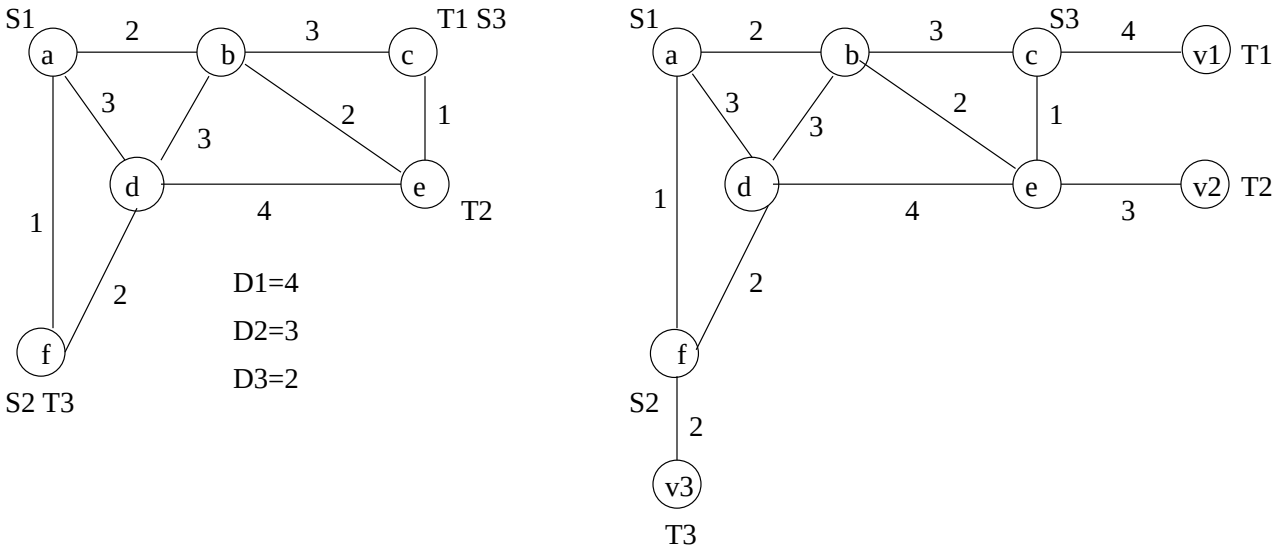


FIG. 1.9 – Transformation d'un multiflot avec demandes en un multiflot maximal.

Inversement, $IMFP$ peut être réduit à un problème de multiflot maximal borné par des demandes : ajoutons une demande d_k égale à la somme des capacités des arêtes (ou des arcs entrant dans un graphe orienté) dont le sommet final est t_k et l'objectif est alors de maximiser le multiflot borné par les demandes d_k.

Il existe également d'autres problèmes avec demandes, comme par exemple les problèmes de flot concurrent ou de coupe "sparsest" ([62] et [88]), que nous n'aborderons pas dans cette thèse. Les problèmes de flot concurrent et de coupe "sparsest" ont été largement étudiés, avec de nombreux résultats récents. Considérons un multiflot dans lequel on désire envoyer d_k unités de la source s_k au puits t_k, $k = 1, .., K$; l'objectif du problème de flot concurrent est de satisfaire toutes les demandes de manière à ce que la proportion des demandes satisfaites soit maximale et identique pour tous les flots. Le problème de coupe "sparsest" consiste quant à lui à obtenir une coupe pour laquelle le rapport, formé par les capacités des arêtes de la coupe sur les demandes qui traversent cette coupe, soit minimal. On peut trouver de nombreux résultats concernant ces problèmes dans [34], [88] et [10].

1.3 Applications

Du fait de leur généralité, les problèmes de multiflot et de multicoupe en nombres entiers, ainsi que leurs problèmes connexes, offrent une large gamme d'applications. Nous citons dans

cette partie les applications mentionnées par différents auteurs. L'un ou l'autre des problèmes définis dans la section précédente fournit directement un modèle reflétant des situations pratiques, dans des domaines variés : réseaux de communication et de télécommunication, réseaux informatiques, problèmes de distribution ou de routage ou encore problèmes de transport ferroviaire. Nous pouvons citer par exemple le problème de congestion téléphonique résolu dans [9], ce problème consiste à trouver un routage optimal des appels téléphoniques dans les périodes où les capacités des liaisons directes ne sont pas suffisantes. Nous pouvons également citer le problème d'optimisation parallèle dans les bases de données résolu dans [44]. De nombreuses applications existent pour les problèmes avec coûts, comme par exemple une application dans les transports aériens dans [5] et une application pour le cablage de centrales nucléaires dans [19]. D'un point de vue plus théorique, la résolution de certains des problèmes de multiflots ou de multicoupes est également un enjeu important puisqu'elle permet de résoudre d'autres problèmes que l'on peut formuler à l'aide de ceux-ci.

1.4 Organisation de la thèse

Dans le chapitre 2 nous dressons l'état de l'art concernant les problèmes de multiflot entier maximal et de multicoupe minimale ainsi que leurs problèmes connexes. Nous présentons en premier lieu la relation de dualité qui lie la relaxation continue de ($IMFP$) et la relaxation continue de ($IMCP$) en utilisant les modèles mathématiques définis dans l'introduction. Nous présentons les principales méthodes de résolution du problème de multiflot continu. Puis nous donnons la complexité des problèmes entiers et nous détaillons les méthodes utilisées pour résoudre ces problèmes. La complexité d'un problème dépend de plusieurs paramètres comme le type de graphe considéré ou le nombre K de paires à relier ou à séparer. Les résultats varient également en fonction de l'orientation ou non du graphe, l'orientation facilite parfois la résolution des problèmes (voir section 2.4.2), et parfois elle la rend plus difficile (voir section 2.3.4). Notons qu'il n'est pas possible de transformer un problème non orienté en un problème orienté en remplaçant chaque arête par deux arcs de directions opposées : la capacité (ou poids) d'un arc serait alors couplée avec celle de l'arc de direction opposée qui lui est associé et les problèmes qui en résulteraient seraient différents des problèmes initiaux que sont $IMFP$ et $IMCP$. Nous étudions les problèmes dans les graphes généraux non orientés et orientés dans la section 2.3 avant de les considérer dans des graphes particuliers comme les arbres ou les graphes planaires dans la section 2.4. Dans chacune de ces deux sections, nous considérons en premier les problèmes généraux, souvent difficiles à résoudre, puis les problèmes connexes

qui peuvent alors devenir polynomiaux. On peut noter que jusqu'à maintenant, très peu de méthodes de résolution pour les problèmes NP-difficiles, présentés dans cette thèse, ont été proposées.

Dans le chapitre 3, nous présentons de nouveaux résultats de complexité pour $IMFP$ et $IMCP$ lorsque le graphe considéré est un arbre orienté. Nous montrons que $IMFP$ et $IMCP$ sont polynomiaux dans les arbres orientés. Nous proposons un algorithme gourmand permettant de résoudre ces problèmes dans les arborescences.

Le fait que $IMFP$ et $IMCP$ sont polynomiaux dans les arbres orientés est à opposer au fait que ces problèmes sont Max SNP-difficiles dans les arbres non orientés. Dans le chapitre 4, nous montrons comment vérifier si l'on peut orienter un arbre non orienté afin d'obtenir une instance facile de $IMFP$ et $IMCP$. Nous présentons ensuite une méthode de résolution pour $IMCP$ dans les arbres non orientés, pour les instances difficiles, fondée sur un branch-and-bound utilisant à la fois la programmation linéaire (PL) et la programmation semi-définie positive (SDP). Nous montrons que la borne inférieure fournie par la SDP est meilleure que celle fournie par la PL. En revanche le coût d'exécution de la SDP est beaucoup plus élevé que celui de la PL.

Dans le chapitre 5, nous nous intéressons aux problèmes de coupe et de flot dans les réseaux en anneau. Nous proposons un algorithme polynomial pour les problèmes de coupe et de flot multiterminaux ainsi qu'un algorithme polynomial pour $IMCP$, algorithme fondé sur l'algorithme gourmand proposé dans le chapitre 3.

Nous concluons par un tableau résumant les principaux résultats de complexité présentés dans cette thèse.

Chapitre 2

État de l'art

Comme nous l'avons précédemment indiqué, le problème du multiflot maximal en nombres entiers a été introduit en 1963 par Hu [49]. Depuis lors, de nombreux résultats ont aidé à mieux cerner ce problème et tous les problèmes connexes définis dans le chapitre précédent. Les résultats de dualité entre $(IMFP)$ et $(IMCP)$ ont notamment permis d'obtenir de nombreux résultats de complexité et plusieurs algorithmes polynomiaux sont fondés sur cette relation de dualité. Nous rappelons dans ce chapitre les principales méthodes utilisées pour la résolution de la relaxation continue de $(IMFP)$. La plupart des résultats importants de complexité ([70, 23, 35]) et d'approximation ([2, 34, 55, 42]) sont très récents. Nous en ferons une synthèse la plus complète possible et nous présenterons également la seule réelle tentative de résolution exacte de $(IMFP)$ ([9]) qui a été réalisée jusqu'à aujourd'hui.

2.1 Deux problèmes continus liés par dualité

Une condition générale de coupe qui doit être vérifiée par tout multiflot respectant les contraintes de capacité a été donnée par Lomonosov dans [66] puis dans [67], on peut la trouver également dans [84]. Elle se formule de la manière suivante :

$$\text{(Condition de Coupe)} \quad \text{pour tout } X \subseteq V, \sum_{e \in \rho_1(X)} u_e \geq \sum_{k \in \rho_2(X)} F_k$$

où $\rho_1(X)$ est l'ensemble des arêtes avec exactement un sommet dans X et $\rho_2(X)$ l'ensemble des $k \in \{1, .., K\}$ pour lesquels soit s_k soit t_k appartient à X, de manière exclusive. Pour une instance donnée, la valeur d'un flot admissible est au plus égale à la valeur d'une multicoupe.

La preuve de cette condition générale de coupe utilise un résultat sur les métriques donné dans [74].

Garg, Vazirani et Yannakakis [34] montrent la dualité des deux problèmes continus relâchés en utilisant une modélisation arête-flot pour le problème de multiflot dans un graphe quelconque [1], mais ils utilisent une modélisation arête-chaîne pour étudier ces problèmes dans les arbres [35]. Comme dans [88], nous proposons ici d'utiliser une modélisation arête-chaîne qui permet une vision plus simple de la dualité.

Théorème 2.1 : *(Garg, Vazirani et Yannakakis* [34]) La relaxation continue du programme de la $s_k - t_k$ multicoupe minimale est le dual linéaire d'un programme de multiflot maximal continu.

Preuve : Considérons la relaxation continue de $(IMFP)$ que nous appellerons $(CMFP)$ et associons une variable duale c_e à chaque contrainte (3), nous obtenons alors deux programmes duaux continus :

$$(CMFP) \quad \left| \begin{array}{lll} \text{Maximiser} & \sum_{i=1}^{M} f_i & \\ \text{Sous les contraintes} & \sum_{i\, s.t.\, e \in p_i} f_i \leq u_e & \forall e \in E \\ & f_i \geq 0 & \forall i = 1, .., M \end{array} \right. \quad (3)$$

$$(CMCP) \quad \left| \begin{array}{lll} \text{Minimiser} & \sum_{e \in E} u_e\, c_e & \\ \text{Sous les contraintes} & \sum_{e \in p_i} c_e \geq 1 & \forall i = 1, .., M \\ & c_e \geq 0 & \forall e \in E \end{array} \right. \quad (4)$$

Si nous remplaçons les contraintes ($c_e \geq 0\ \forall e \in E$) par les contraintes ($c_e \in \{0,1\}\ \forall e \in E$) nous obtenons un programme entier qui est précisément le programme $(IMCP)$. Notons que $(CMCP)$ est en effet la relaxation continue de $(IMCP)$: les contraintes ($c_e \leq 1\ \forall e \in E$) peuvent être omises dans $(CMCP)$ car elles sont vérifiées par toute solution optimale, le problème devenant une minimisation sous les contraintes (4) et les données u_e étant positives.

Les conditions d'optimalité des écarts complémentaires en programmation linéaire sont données par :

$\forall i = 1, .., M \;\; f_i^* > 0 \;\Rightarrow\; \sum_{e \in p_i} c_e^* = 1 \quad (6)$

$\forall e \in E \;\; c_e^* > 0 \;\Rightarrow\; \sum_{i\, s.t.\, e \in p_i} f_i^* = u_e \quad (7)$

avec f_i^* la valeur du flot optimal circulant sur la i^{eme} chaîne p_i, $i = 1, .., M$ et $c_e^* = 1$ si et seulement si l'arête e appartient à la coupe optimale.

Les conditions (6) signifient que dans toute solution optimale soit le flot circulant sur la chaîne p_i est égal à 0 soit la contrainte (4) associée est saturée ; si les variables c_e^* sont entières alors il y a une et une seule arête de p_i dans la multicoupe pour tout i tel que $f_i^* > 0$. Les conditions (7) signifient que dans toute solution optimale si l'arête e n'est pas saturée par le flot alors $c_e^* = 0$; si les variables sont entières alors toutes les arêtes de la coupe sont saturées, i.e. sont des arêtes avec une capacité résiduelle égale à zéro.

Notons que les résultats de dualité présentés ci-dessus sont valables dans tout graphe orienté ou non orienté. Pour le cas du flot simple ($K = 1$) les sommets du polyèdre du primal et du dual sont entiers et le théorème de max-flot min-coupe est une conséquence directe du fait que les solutions obtenues par programmation linéaires sont entières. Mais cette propriété n'est pas vraie en général et cela explique en partie les difficultés de résolution des problèmes [34]. La condition de coupe donnée au début de cette section est également une conséquence directe du théorème 2.1.

2.2 Les méthodes de résolution du multiflot continu

Nous avons vu dans la section 1.1.1 que le problème du multiflot maximal peut se formuler comme un programme linéaire, on peut donc résoudre sa relaxation continue par toute méthode de programmation linéaire comme par exemple la méthode du simplexe ou la méthode des points intérieurs. Mais la dimension du problème est généralement trop grande pour que celui-ci soit résolu directement de manière efficace par ces méthodes. En effet, pour une instance comportant 20 flots, 100 sommets et 1 000 arêtes, le problème de multiflot (utilisant la formulation $(IMFP)$ (I)) associé comprend 20 000 variables et plus de 20 000 contraintes et pour une instance comportant 100 flots, 1 000 sommets et 100 000 arêtes, le problème de multiflot (utilisant la formulation $(IMFP)$ (I)) associé comprend 10 000 000 de variables et plus de 10 000 000 de contraintes. Il faut donc envisager d'autres techniques de résolution, comme les méthodes de décomposition ou les méthodes de partitionnement. Nous renvoyons à [1] et à [40] pour une étude complète du problème du multiflot maximal continu. Nous ne présentons ici qu'un aperçu des méthodes de résolution connues pour ce problème.

2.2.1 Méthodes de décomposition

Les méthodes de décomposition, comme leurs noms l'indiquent, procèdent par décomposition pour ramener la résolution du problème initial à une séquence de sous-problèmes de flots simples ou de plus courts chemins, pour lesquels il existe des algorithmes de résolution

efficaces. Deux classes de méthodes de décomposition sont utilisées habituellement, la décomposition par les prix, qui contient la relaxation lagrangienne et la génération de colonnes, et la décomposition par les ressources.

2.2.1.1 Décomposition par les prix

Elle consiste à résoudre un ensemble de flots simples et à mettre à jour les variables prix qui sont les variables duales associées aux contraintes de capacité. Il faut alors trouver des prix appropriés de telle sorte que les solutions optimales de ces sous-problèmes permettent conjointement de résoudre le problème initial.

2.2.1.1.1 Relaxation Lagrangienne La méthode associe des multiplicateurs de Lagrange positifs aux contraintes de capacité, de la formulation $(IMFP)$ (I), puis relaxe ces contraintes de manière à les introduire dans la fonction objectif pour obtenir un problème qui se décompose en K sous-problèmes de flots simples. Pour mettre à jour les multiplicateurs de Lagrange, plusieurs techniques heuristiques peuvent être utilisées comme par exemple la méthode des ajustements, la méthode du sous-gradient ou bien encore la méthode des faisceaux.

2.2.1.1.2 Génération de colonnes Cette méthode utilise la formulation $(IMFP)$ (II) et met en œuvre l'idée que toutes les colonnes ne peuvent pas être complètement explicitées, elles sont générées au fur et à mesure de leur utilité. La méthode révisée du simplexe apparaît assez adaptée pour réaliser ce type de stratégie, en mettant à jour les variables duales de façon à ce que le coût réduit de chaque variable en base (base maintenue par le simplexe révisé à chaque itération) soit nul. A chaque étape, nous devons calculer pour chaque flot k la valeur du plus court chemin entre s_k et t_k où le coût associé à chaque arc est égal au coût réduit. Ainsi nous pouvons juger à chaque étape de la qualité de la solution courante et interrompre le processus quand la solution courante est satisfaisante.

2.2.1.2 Décomposition par les ressources

Au lieu d'utiliser des prix pour décomposer le problème de multiflot maximal continu en un ensemble de sous-problèmes de flots simples ou de plus courts chemins comme dans les approches précédentes, l'approche de décomposition par les ressources consiste à déterminer une affectation de la capacité sur chaque arc pour chaque flot simple. L'optimum du problème global est atteint par la résolution de K sous-problèmes de flots simples. Pour chaque arc, la somme des capacités allouées à chaque flot doit être inférieure ou égale à la capacité de cet arc.

Il s'ensuit que les flots simples de chaque sous-problème constituent une solution réalisable pour le problème initial. Les techniques de décomposition par les ressources diffèrent dans la méthode de résolution du problème d'affectation sous-jacent. Plusieurs approches ont été proposées, comme la méthode d'approximation tangentielle et la méthode de sous-gradient.

2.2.2 Méthodes de partitionnement

Comme nous avons pu le constater, le problème de multiflot continu peut être modélisé (par la formulation $(IMFP)$ (I)) comme un programme linéaire qui combine deux types de contraintes : d'une part, des contraintes de conservation de flots indépendantes et d'autre part des contraintes couplantes relatives au partage des capacités sur les arcs par tous les flots. Cette structure sous-jacente de flots dans les réseaux est à la base des méthodes de partitionnement. En effet, ces méthodes sont des spécialisations de la méthode du simplexe, où la base courante est partitionnée de façon à tirer profit des propriétés particulières des flots dans le graphe. Nous pouvons ainsi distinguer deux types de partitionnement, le partitionnement primal et le partitionnement dual.

2.2.2.1 Partitionnement primal

Dans la formulation $(IMFP)$ (II), formulation arêtes-chaînes, le problème du multiflot se présente comme un programme linéaire à structure bloc-angulaire. La spécification du simplexe dans l'approche du partitionnement primal tente de tirer profit de cette structure particulière. Les simplifications apportées par cette approche réalisent des gains de temps substantiels en ce qui concerne les calculs matriciels nécessaires dans la méthode du simplexe.

2.2.2.2 Partitionnement dual

Cette approche se base sur le fait qu'en pratique, seul un sous-ensemble réduit des contraintes est saturé à l'optimum. Ainsi la plupart des contraintes peuvent être relaxées sans perte d'optimalité. Si les contraintes relaxées sont satisfaites par la solution courante, on est à l'optimum, sinon le partitionnement en variable de base et hors base est mis à jour.

2.3 Complexité et résolution des problèmes entiers dans des graphes quelconques

Le problème du multiflot en nombres entiers a été étudié en 1963 par Hu [49], qui s'est appuyé sur les travaux de Ford et Fulkerson [28] pour résoudre le problème du biflot maximal en nombres entiers dans un cas particulier.

Des travaux ultérieurs ont montré la NP-difficulté du problème général et sa polynomialité dans certains cas particuliers.

Dans cette section, nous donnons les principaux résultats connus de complexité pour $IMFP$ et $IMCP$ et pour les problèmes connexes dans des graphes quelconques, nous présentons également quelques algorithmes d'approximation et les différentes méthodes existantes permettant de résoudre ces problèmes.

2.3.1 Résolution exacte de $IMFP$

Hu [49] a prouvé que $IMFP$ dans un graphe non orienté avec $K = 2$ et des capacités paires sur les arêtes est polynomial ; Seymour a donné une nouvelle preuve du théorème de Hu dans [87]. L'algorithme proposé calcule un biflot par combinaison de deux flots simples. Ces résultats peuvent être obtenus car les solutions continues de ($IMCP$) et ($IMFP$) quand $K = 2$ sont semi-entières, c'est-à-dire que les variables sont multiples de $\frac{1}{2}$ ([49], [50] et [86]). Rothschild et Winston [81] ont étendu le résultat de Hu [49] pour les biflots au cas des graphes Eulériens.

Détaillons sur un exemple la méthode utilisée par Hu pour montrer que si toutes les capacités sont paires, alors le problème du biflot maximal en nombres entiers dans un graphe quelconque non orienté est polynomial :

La figure 2.1 donne un exemple d'une instance du biflot sur un graphe dans lequel nous avons choisi une orientation arbitraire.

Ajoutons au graphe deux nouveaux sommets, sur lesquels nous plaçons une source S et un puits T que nous relions aux sources et aux puits existants par des arcs (de capacité M très grande) de deux manières différentes, comme le montre la figure 2.2. Pour chaque nouveau graphe, nous calculons un flot simple (h_1 et h_2) de S à T.

La solution optimale du biflot dans le graphe d'origine est alors donnée par :

$f_1 = -\frac{1}{2}(h_1 + h_2)$ et $f_2 = -\frac{1}{2}(h_1 - h_2)$ et $f* = f_1 + f_2$.

Ainsi, si toutes les capacités sont paires, comme dans notre exemple, alors les deux flots simples h_1 et h_2 sont également pairs et finalement les flots f_1 et f_2 sont entiers et le problème

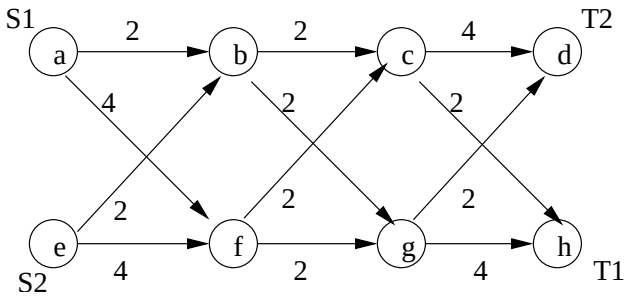


FIG. 2.1 – Un exemple de biflot.

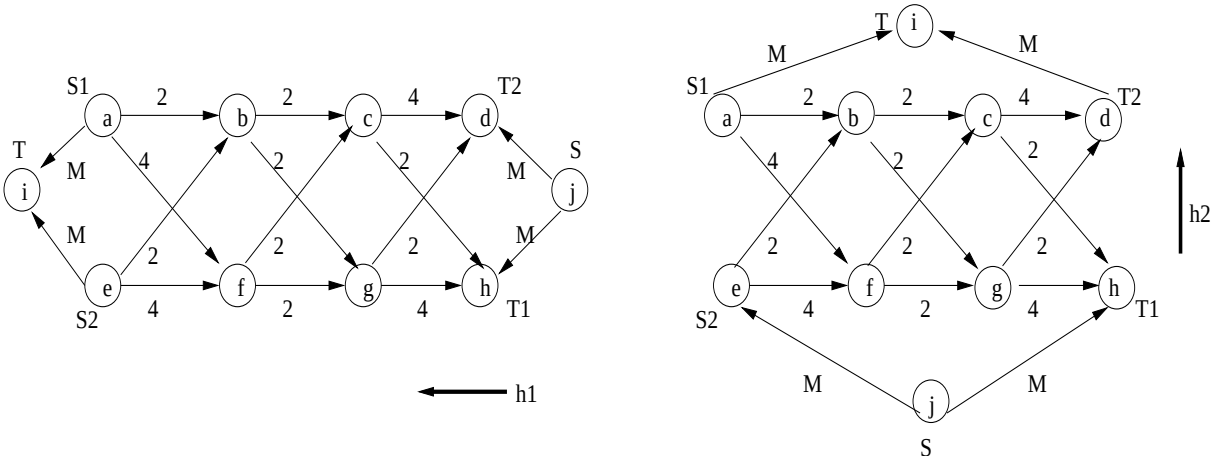


FIG. 2.2 – Transformation d'un biflot en 2 flots simples.

du biflot est polynomial.

Pour notre exemple, nous obtenons $h_1 = -8$ et $h_2 = 6 - 2 = 4$, d'où $f_1 = \frac{(-8+4)}{-2} = 2$ et $f_2 = \frac{(-8-4)}{-2} = 6$, ce que l'on peut en effet obtenir de la manière suivante :

On route 2 unités de flot F_1 sur la chaîne (a, b, g, h).

On route 2 unités de flot F_2 sur la chaîne (e, b, c, d).

On route 2 unités de flot F_2 sur la chaîne (e, f, c, d).

On route 2 unités de flot F_2 sur la chaîne (e, f, g, d).

Plus tard Rajagopalan [79] a amélioré le résultat de Hu : $IMFP$ avec $K = 2$ est polynomial si la somme des capacités de toutes les arêtes incidentes à chaque sommet est paire.

Comme cela a déjà été dit, contrairement au problème du multiflot entier à coût minimal pour lequel plusieurs résultats pratiques ont été publiés récemment ([5, 32, 65, 71]), il existe très peu de tentatives de résolution de $IMFP$ et $IMCP$. Néanmoins, Brunetta, Conforti et Fischetti ([9]) ont proposé un algorithme de Branch and Cut basé sur une approche polyédrale. Ils décrivent tout d'abord plusieurs classes d'inégalités valides puis plusieurs procédures de lifting afin d'étendre une inégalité valide donnée pour un problème contenant K flots à router, à un problème contenant un très grand nombre de flots à router. Pour cela, ils introduisent une nouvelle classe de contraintes valides, contraintes appelées "multi-handle comb inequalities". Ils prouvent également que le polytope de ($IMFP$), défini dans le cas où les capacités sur les arêtes sont unitaires (i.e. le polytope de ChemDisjAre Max), est de pleine dimension, cela signifie que le polytope a pour dimension le nombre de contraintes du problème, et ils montrent que certaines des contraintes valides précédentes définissent dans ce cas des facettes. Ils montrent ensuite que les procédures de lifting permettent de préserver ces facettes sous certaines conditions. A partir de ces résultats, ils construisent un algorithme de Branch and Cut, contenant des procédures de séparation pour les principales classes d'inégalités étudiées. Ils résolvent tout d'abord des instances de $IMFP$ avec des capacités unitaires sur les arêtes, i.e. le problème de ChemDisjAre Max, les instances comptent jusqu'à 100 sommets, 495 arêtes et 5 paires de sommets à relier, pour des temps de calcul compris entre moins d'une seconde et une heure selon la taille de l'instance. Ils ont également appliqué leur algorithme à un problème réel de congestion téléphonique dont les instances contiennent 8 sommets et 28 arêtes, des capacités entières sur les arêtes et jusqu'à 13 flots, toutes les instances ont été résolues en moins d'une seconde. En fait, on peut espérer résoudre de manière exacte uniquement des petites instances dans des graphes quelconques. La raison principale peut être trouvée dans l'étude de la complexité de ces problèmes.

2.3.2 Complexité de $IMFP$ et $IMCP$

En premier lieu, rappelons que $IMFP$ et $IMCP$ sont deux problèmes Max SNP-difficiles ([76], [23] et [35]), même dans plusieurs cas particuliers. Ces résultats impliquent qu'il n'existe pas de schéma d'approximation polynomial pour ces problèmes si P≠NP. Il est également connu que $IMFP$ est fortement NP-difficile [54] même quand le nombre de flots est égal à 2 et que toutes les arêtes ont des capacités égales à 1 dans un graphe orienté (cela nous ramène donc à résoudre une instance orientée de MAX CHEMDISJARE avec $K = 2$). Even, Itaï et Shamir [26] ont ensuite montré que le problème de décision associé au problème du biflot entier était NP-complet dans le cas non orienté, en effectuant une réduction à partir du problème Sat. Résultats repris par Itaï [52] en 1978, avec une étude plus complète de la complexité de l'ensemble des problèmes de flots.

En fait, comme nous allons le voir dans la section suivante, $IMFP$ n'est pas seulement fortement NP-difficile mais trouver une solution approchée avec un ratio de performance fixe est encore un problème NP-difficile. Pour $IMCP$, les résultats de complexité sont également négatifs alors que certains problèmes connexes peuvent être bien approximés (voir la sous-section 2.3.4 pour plus de détails).

2.3.3 Approximation de $IMFP$ et $IMCP$

Comme $IMFP$ et $IMCP$ sont NP-difficiles, les seuls algorithmes polynomiaux que l'on peut espérer obtenir sont des algorithmes donnant des solutions approchées. D'un point de vue positif, dans des graphes non orientés, Garg, Vazirani et Yannakakis ont proposé dans [34] un algorithme $O(log(K))$-approché où K est le nombre de flots. Leur algorithme fournit une solution à la fois à $IMFP$ et à $IMCP$. Pour obtenir ce résultat remarquable, les auteurs utilisent une relaxation du programme linéaire ($IMFP$) qui est équivalente à la formulation ($IMFP$) (I) présentée dans l'introduction de cette thèse (voir section 1.1.1). En résolvant le dual de ce programme linéaire, ils définissent un graphe avec des distances sur les arêtes. Ensuite, en utilisant les conditions des écarts complémentaires et en commençant par chaque sommet terminal (source ou puits) ils construisent plusieurs coupes (de capacités suffisamment petites) séparant le sommet initial de son puits (resp. source). Finalement, ils obtiennent une multicoupe admissible en considérant l'union de toutes ces coupes. De plus, ils prouvent que l'analyse du pire cas est fine (un exemple atteignant la borne est donné).

Les deux problèmes semblent plus difficiles dans les graphes orientés. Pour $IMCP$, Cheriyan, Karloff et Rabani [12] ont prouvé qu'il existe un algorithme polynomial pour trouver une

multicoupe dont la valeur C satisfait la relation suivante : $C \leq 108F^{*3}$, avec F^* la valeur d'un multiflot maximal. Ils ont également prouvé que l'on peut trouver en un temps polynomial une multicoupe dont la valeur satisfait la relation suivante : $C \leq 39ln(K+1)F^{*2}$. Ces résultats doivent être comparés avec ceux obtenus dans les graphes non orientés : $C = O\left(F^* log\left(F^*\right)\right)$.

Le meilleur résultat négatif connu pour l'approximation de $IMFP$ dans les graphes orientés dit qu'à moins que P=NP, aucun algorithme polynomial ne peut fournir un meilleur ratio de performance que $m^{\frac{1}{2}-\varepsilon}$ pour tout $\varepsilon > 0$ [42]. Dans le même article, les auteurs proposent un algorithme gourmand $O\left(\sqrt{md_{max}} log^2 m\right)$-approché pour le problème du multiflot maximal où le but est de maximiser le nombre de demandes satisfaites (ici d_{max} est la valeur de la demande maximale).

Roupin a montré en 2002 que dans un graphe non orienté, si P≠NP, il n'existe pas d'algorithme approché polynomial avec une performance fixe garantie pour le problème de multiflot maximal en nombres entiers dont l'objectif est de satisfaire le plus grand nombre de demandes (ce résultat apparaît dans [18]). Pour cela il utilise une réduction polynomiale à partir du problème de l'Ensemble Stable de Cardinal Maximal qui est connu pour être NP-difficile à approximer par un ratio constant.

2.3.4 Complexité et approximation des problèmes connexes

Dans cette section, nous donnons plusieurs résultats de complexité et d'approximation des cas particuliers de $IMFP$ et de $IMCP$ dans des graphes quelconques. Nous présentons principalement des résultats concernant les problèmes de coupe multiterminaux, de chemins disjoints par les arêtes et de flots inséparables.

2.3.4.1 Les problèmes de coupe multiterminaux

Comme énoncé dans la section 1.2.1, le problème de Coupe Multiterm est un cas particulier de $IMCP$. Dalhaus, Johnson, Papadimitriou, Seymour et Yannakakis ont prouvé dans [23] que le problème de la Coupe Multiterm dans les graphes quelconques est NP-difficile pour $K \geq 3$, et ce même dans les graphes planaires. Ils ont également prouvé que Coupe Multiterm est Max SNP-difficile pour $K \geq 3$ dans les graphes quelconques. Ce dernier résultat a été utilisé dans [33] pour montrer que le problème de Coupe Multiterm est Max SNP-difficile dans un graphe orienté, même pour $K = 2$. Cela implique qu'il n'existe pas de schéma d'approximation polynomial si P≠NP [2]. Cependant, les auteurs de [23] donnent un résultat positif sur l'approximation de ce problème dans les graphes quelconques non orien-

tés : il existe un algorithme $(2 - \frac{2}{K})$-approché polynomial pour le problème de la COUPE MULTITERM minimale dans les graphes quelconques. L'idée principale de cet algorithme est de construire une solution admissible en cherchant K coupes "isolantes" pour chaque sommet terminal [23]. Ce dernier résultat a été amélioré par Calinescu, Karloff et Rabani [10] ; ils ont utilisé une nouvelle relaxation géométrique et ont obtenu un algorithme $\left(\frac{3}{2} - \frac{1}{K}\right)$-approché. Leur relaxation utilise le K-simplexe S_K qui a K sommets ; le i^{eme} sommet est un point de x dans S_K avec $x_i = 1$ (et toutes les autres coordonnées sont égales à 0). La relaxation se déroule comme suit : placer les sommets du graphe aux points de S_K tel que le terminal i soit placé au i^{eme} sommet de S_K. Chaque arête est placée sur la droite reliant deux terminaux. Une deuxième amélioration a été obtenue par Karger, Klein, Thorup, Stein et Young (voir [53]). Ces auteurs ont amélioré le ratio de l'algorithme approché en étudiant la précédente relaxation géométrique et ils ont obtenu un algorithme 1.3438-approché pour tout K, et un algorithme $\frac{12}{11}$-approché pour $K = 3$.

Dans les graphes quelconques orientés le problème de COUPE MULTITERM peut être approximé avec un ratio égal à $2log\,K$ [33]. L'algorithme s'effectue en $log\,K$ étapes et à chacune d'elle, on retire les arcs dont la capacité est au plus égale à deux fois la valeur du multiflot maximal (sur les arcs considérés).

Rappelons qu'un cas encore plus particulier de $IMCP$ est le problème de la K-COUPE minimale (voir section 1.2.2 pour la définition). Le problème dans un graphe non orienté est polynomial pour K fixé, mais NP-difficile pour un K non fixé : en fait Goldschmidt et Hochbaum ont démontré dans [39] que le problème de la CLIQUE maximale peut se réduire polynomialement au problème de la K-COUPE. Levine [64] a récemment amélioré les résultats de Goldschmidt et Hochbaum [39] en proposant un algorithme polynomial en $O\left(m\,n^{K-2}log^3 n\right)$ quand $K \leq 6$.

Dans les graphes orientés, tous les problèmes de multicoupe semblent plus difficiles à approximer. Néanmoins, Naor et Zosin ont amélioré l'algorithme $(O\,(log(K))$-approché de Garg, Vazirani et Yannakakis [34] en proposant dans [72] un algorithme 2-approché pour le problème orienté de la multicoupe symétrique. Cet algorithme approché utilise un programme linéaire particulier où le saut d'intégrité est au plus de 2. Une multicoupe symétrique est l'ensemble des arcs dont le retrait sépare soit s_k de t_k, soit t_k de s_k, $\forall k = 1, .., K$. Notons qu'il n'existe malheureusement auncune relation directe entre ce problème et $IMCP$ dans les graphes orientés qui nous permettrait d'obtenir des résultats similaires pour $IMCP$.

2.3.4.2 Les problèmes de chemins disjoints par les arêtes et de flots inséparables

Certains problèmes connexes à $IMFP$ sont moins difficiles à résoudre. En effet, les problèmes de FlotInSep Max et de ChemDisjAre Max admettent un algorithme $O(\sqrt{m})$-approché ([6], [55] et [58]). Notons qu'il s'agit là de la meilleure garantie que l'on puisse obtenir par un algorithme polynomial pour ces deux problèmes dans des graphes quelconques orientés, car ces problèmes sont NP-difficiles à approximer par un facteur $m^{\frac{1}{2}-\varepsilon}$ pour tout $\varepsilon > 0$ dans les graphes orientés [42]. De plus, le problème de ChemDisjAre Max est NP-difficile pour $K = 2$ dans les graphes orientés [29]. Ceci contraste avec le cas non orienté, où Robertson et Seymour [80] ont montré que pour tout K fixé, le problème de ChemDisjAre Max peut être résolu en temps polynomial. Dans des graphes orientés et non orientés, le problème de ChemDisjAre Max est NP-difficile si l'on ne fixe pas K ([29] et [70]).

2.4 Graphes particuliers

Cette section présente une étude des résultats connus pour les différents problèmes dans des graphes particuliers : les graphes orientés sans circuit, les arbres, les graphes bipartis, les graphes planaires et les anneaux.

2.4.1 Graphes orientés sans circuit

Nous avons vu, dans la section 2.3.4.1, que le problème de Coupe Multiterm est NP-difficile dans les graphes orientés. Néanmoins, Costa a montré dans [17] que les problèmes de Flot Multiterm et de Coupe Multiterm sont polynomiaux dans les graphes orientés sans circuit. En effet, l'auteur a montré que les problèmes de Flot Multiterm et de Coupe Multiterm dans les graphes orientés sans circuit peuvent être résolus polynomialement en utilisant un algorithme de max flot-min coupe.

Nous avons vu dans la section 2.3.4.2 que le problème de ChemDisjAre Max est NP-difficile pour $K = 2$ dans les graphes orientés [29] et que pour tout K fixé, le problème de ChemDisjAre Max peut être résolu en temps polynomial dans un graphe non orienté [80]. Les auteurs de [29] ont également montré que si le graphe est orienté et sans circuit, alors pour tout K fixé, le problème de ChemDisjAre Max peut être résolu en temps polynomial.

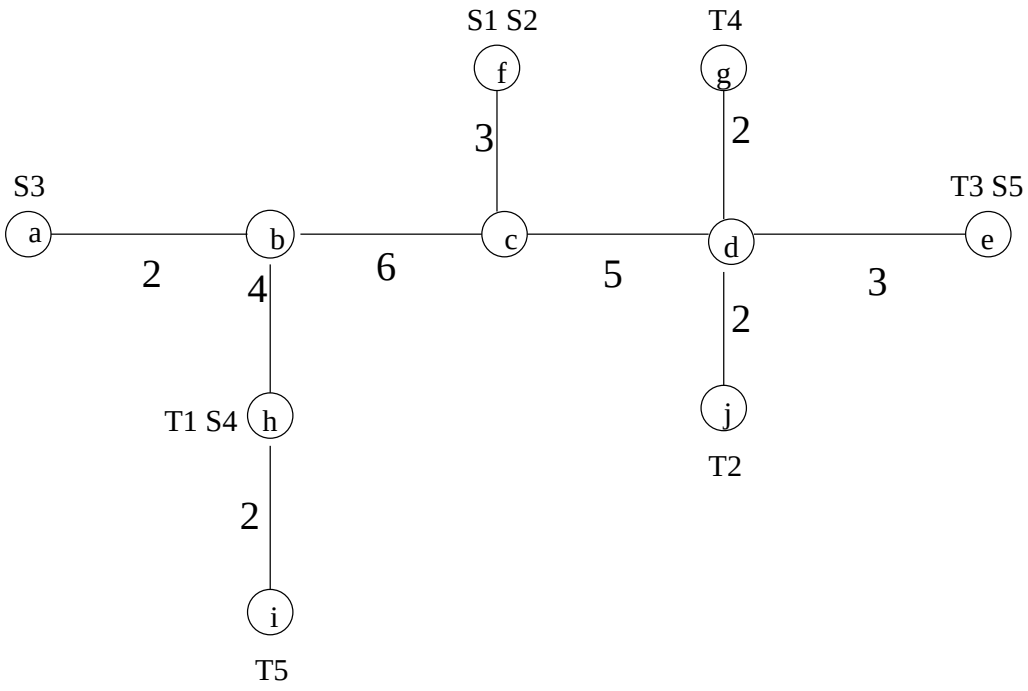


FIG. 2.3 – Un exemple pour $IMFP$ dans un arbre.

2.4.2 Arbres

Dans un arbre, il n'existe qu'une seule chaîne entre la source et le puits d'un flot (voir figure 2.3). Cette propriété nous sera utile pour résoudre polynomialement plusieurs problèmes dans ce type de graphe, mais elle n'implique pas la polynomialité de tous les problèmes dans les arbres non orientés.

2.4.2.1 Arbres non orientés

Garg, Vazirani et Yannakakis [35] ont montré que $IMFP$ et $IMCP$ étaient Max SNP-difficiles sur un arbre non orienté. Ils ont utilisé une réduction linéaire à partir du problème de 3-couplage pour $IMFP$ et à partir du problème de couverture de sommet pour $IMCP$. Les réductions linéaires précédentes utilisant les capacités des arêtes, Srivastav et Stangier [89] ont étendu les résultats de la manière suivante : $IMFP$ et $IMCP$ sont Max SNP-difficiles sur un arbre non orienté avec de grandes capacités.

Les auteurs de [35] ont également présenté un algorithme approché efficace pour résoudre $IMCP$ et $IMFP$: leur algorithme calcule une multicoupe et un multiflot tel que le poids de

la multicoupe est égal au plus à deux fois la valeur du multiflot, c'est-à-dire qu'ils proposent un algorithme 2-approché pour $IMCP$ et un algorithme $\frac{1}{2}$-approché pour $IMFP$ dans les arbres.

Leur algorithme suit une approche primale-duale. Cette approche est guidée par les conditions des écarts complémentaires (voir section 2.1). Les auteurs commencent par transformer l'arbre en une arborescence à partir d'un sommet arbitraire puis l'algorithme effectue deux passages sur l'arbre. Dans le premier, l'algorithme parcourt l'arbre des feuilles vers la racine en routant les flots au fur et à mesure et en coupant certaines arêtes saturées. Au second passage, de la racine vers les feuilles, les arêtes redondantes sont enlevées de l'ensemble obtenu précédemment. Les auteurs ont prouvé que l'ensemble d'arêtes obtenu à la fin de l'algorithme est une multicoupe et que cette multicoupe inclut au plus deux arêtes par chaîne (une chaîne correspondant ici à un flot). Ainsi la capacité de la multicoupe est au plus deux fois supérieure à la valeur du multiflot.

Les auteurs de [35] ont montré que la réduction linéaire proposée à partir du problème de couverture de sommet pour $IMCP$ restait valable dans des arbres de hauteur 1, i.e. des étoiles, avec capacités unitaires sur les arêtes, et donc que $IMCP$ dans ce cas particulier était encore Max SNP-difficile. Néanmoins, ils montrent que $IMFP$ sur des arbres de hauteur 1 peut être résolu en temps polynomial car ce problème se réduit en un problème de b-couplage qui peut être résolu en temps polynomial [2]. $IMFP$ dans un arbre avec des capacités unitaires sur les arêtes devient un problème de ChemDisjAre Max ; Garg, Vazirani et Yannakakis [35] ont également proposé un algorithme polynomial pour le résoudre. Cet algorithme effectue deux passages sur l'arbre après l'avoir transformé en une arborescence. Dans le premier passage, l'algorithme parcourt l'arbre des feuilles vers la racine en considérant les sommets niveau par niveau. Au sommet v, ils résolvent $IMFP$ sur un arbre de hauteur 1, en utilisant le fait que dans ce cas, $IMFP$ se réduit en un problème de couplage qui peut être résolu en temps polynomial, ils déterminent ensuite pour chaque flot, dont la source (resp. le puits) appartient à la sous-arborescence ayant comme racine v et dont le puits (resp. la source) est en dehors de cette sous-arborescence, si celui-ci peut encore être routé. Dans le second passage, de la racine vers les feuilles, ils routent ces flots non encore routés.

Néanmoins, contrairement à $IMCP$ et $IMFP$, les problèmes de Coupe Multiterm et de Flot Multiterm sont polynomiaux si le graphe est un arbre non orienté. Erdos et Szekely [24] ont proposé un algorithme en $O(n^2)$ pour résoudre le problème de Coupe Multiterm dans les arbres : en fait, leur algorithme résout un problème plus général qui consiste à séparer r sous-ensembles disjoints de sommets. Hasan et Motwani ([44]) ont proposé un algorithme

qu'ils ont nommé DLC pour résoudre le problème de Coupe Multiterm. Costa ([17]) a montré que cet algorithme DLC, pour le problème de Coupe Multiterm dans les arbres non orientés, peut être implémenté en $O(n)$. L'auteur propose également un algorithme en $O\left(n^2\right)$ pour le problème de Flot Multiterm dans les arbres non orientés : cet algorithme résout le problème sur un sous-arbre de hauteur 1, i.e. une étoile reliée à l'arbre par une seule arête, saturant certaines arêtes de l'étoile , arêtes que l'on peut désormais contracter, l'algorithme résout à nouveau le problème sur l'arbre réduit tant qu'il reste des flots que l'on peut router.

2.4.2.2 Arbres bi-orientés

Un arbre bi-orienté est le graphe orienté obtenu à partir d'un arbre non orienté en remplaçant chaque arête par deux arcs indépendants de directions opposées. Erlebach et Jansen [25] ont prouvé que le problème de ChemDisjAre Max dans les arbres bi-orientés de degré quelconque est un problème Max SNP-difficile. Ils ont effectué une réduction linéaire à partir d'une variante bornée du problème de 3-couplage. Ils ont également proposé un algorithme $\left(\frac{5}{3}+\varepsilon\right)$-approché pour le problème de ChemDisjAre Max dans les arbres bi-orientés. Néanmoins, ce problème peut être résolu optimalement en temps polynomial si l'on se restreint à certains cas :

a) Si le degré maximal de l'arbre est borné par une constante, alors la solution optimale peut être obtenue en utilisant la programmation dynamique.

b) Si l'arbre bi-orienté est une étoile, i.e. qu'il ne contient qu'un seul sommet avec un demi-degré extérieur supérieur à 1, alors le problème de ChemDisjAre Max peut être transformé en un problème de couplage maximum dans un graphe biparti que l'on sait résoudre exactement en temps polynomial. Ce dernier résultat s'applique également aux graphes en "toile d'araignée" : un graphe en "toile d'araignée" est un arbre bi-orienté dans lequel au plus un sommet (le centre) a un demi-degré extérieur strictement supérieur à 2.

2.4.3 Graphes bipartis

Considérons maintenant les graphes bipartis augmentés, i.e. les graphes bipartis auxquels on ajoute K sources et K puits (voir fig 2.4). Costa a montré en 2002 que $IMFP$ dans un graphe biparti orienté augmenté est NP-difficile si $K \geq 3$ (ce résultat apparaît dans [18]). La preuve utilise le problème de tomographie discrète "couverture de structure polyatomique de rayons X discrets" [14], i.e. la reconstruction de $(a,b)-$ matrices colorées à partir des

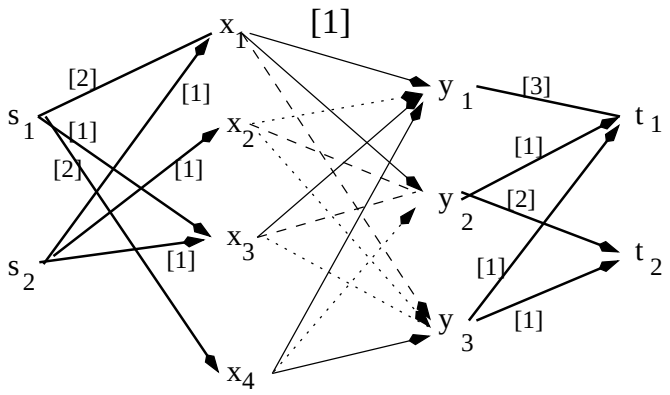


FIG. 2.4 – Un exemple d'instance pour $IMFP$ dans un graphe biparti augmenté.

projections des couleurs (voir par exemple [77]).

Si $K = 2$ la complexité de $IMFP$ dans un graphe biparti orienté augmenté reste un problème ouvert à ce jour. Néanmoins, rappelons que, si $K = 2$, le problème est polynomial dans un graphe non orienté avec des capacités paires (voir section 2.3.2).

Le fait que les problèmes de FLOT MULTITERM et de COUPE MULTITERM sont polynomiaux dans les graphes orientés sans circuit (voir section 2.4.1) implique que ces problèmes sont également polynomiaux sur les graphes bipartis orientés, qu'ils soient augmentés ou non, dans le cas où les arcs suivent tous la même direction (voir figure 2.4).

2.4.4 Graphes planaires

Dans un graphe planaire G, le problème de CHEMDISJARE MAX est NP-difficile (comme on peut le voir dans [70] et [68]), donc $IMFP$ est également NP-difficile. Certains auteurs considèrent les graphes planaires augmentés, obtenus en ajoutant à G toutes les arêtes $\{s_k, t_k\}$, $k = 1, .., K$. Si le graphe augmenté est planaire, Sebo [85] a prouvé que pour K fixé, le problème de CHEMDISJARE MAX est polynomial. Une conséquence de ce résultat est la polynomialité du problème du biflot entier maximal dans les graphes augmentés planaires. Sakarovitch avait déjà prouvé ce résultat dans [83] et avait également proposé un algorithme pour le résoudre. Korach et Penn [60] ont donné un algorithme en $O\left(n\sqrt{\log n}\right)$ pour ce problème. Leur algorithme utilise des résultats connus pour transformer le graphe en son dual et pour calculer des plus courts chemins, ils résolvent ensuite un système d'équations et d'inéquations linéaires pour trouver un biflot entier maximal.

$IMCP$ et le problème de la COUPE MULTITERM dans des graphes planaires ont été prouvés

NP-difficiles par Dalhaus, Johnson, Papadimitriou, Seymour et Yannakakis [23] : la réduction a été effectuée à partir de 3-SAT planaire. Le ratio des valeurs de la multicoupe minimale et du multiflot maximal dans les graphes planaires est au plus en $O(1)$, et il existe un algorithme approché avec un facteur constant pour $IMCP$. Ces derniers résultats ont été montrés par Tardos et Vazirani [90]. Ils utilisent un algorithme de décomposition et résolvent ensuite le dual du programme linéaire correspondant au problème de multiflot en trouvant des plus courts chemins.

Le problème de ChemDisjAre Max est NP-difficile car le problème de décision associé est NP-complet ([70] et [68]). Notons que la condition nécessaire de coupe donnée au début de ce chapitre a été modifiée par Frank [31] si le graphe est une grille : cette condition devient en effet suffisante pour l'existence de K chaînes disjointes, si on l'applique à toutes les lignes et les colonnes de la grille. Le précédent résultat sur le problème de ChemDisjAre Max implique la NP-difficulté de CapChem Max et FlotInSep Max dans les graphes planaires. Kleinberg et Tardos ([56], [57]) ont donné un algorithme $O(\log n)$-approché pour le problème de ChemDisjAre Max dans les graphes planaires presque eulériens et de diamètre uniforme.

2.4.5 Anneaux

Tous les résultats donnés pour les graphes planaires restent valables pour les anneaux. Cette section présente des résultats de complexité pour trois variantes de $IMFP$ dans les anneaux non orientés (ou bidirectionnels). Kubat, Shulman, Vachani et Ward [61] ont proposé un algorithme en $O(n^3)$ pour trouver un multiflot entier avec demandes, ce problème ne peut pas être transformé en un problème de multiflot maximal dans un anneau, comme cela est indiqué dans la section 1.2.6.1, car le graphe ne conserve pas ses caractéristiques d'anneau si l'on ajoute des arêtes après chaque puits. Leur réseau en anneau est caractérisé par le fait que toutes les capacités sont égales et que chaque demande peut être routée en partie dans le sens des aiguilles d'une montre et en partie dans le sens inverse des aiguilles d'une montre. De plus, il s'agit d'un problème avec demandes particulier puisqu'ils cherchent à minimiser les capacités afin de satisfaire les demandes. La seconde variante est le problème de FlotInSep avec des demandes dans un réseau en anneau, où chaque demande doit être routée entièrement dans un sens ou dans l'autre ; ce problème a été montré NP-difficile par Cosares et Saniee dans [15]. Ici encore, ce résultat ne peut pas être adapté à la recherche du FlotInSep Max dans un anneau pour les mêmes raisons. Le problème de ChemDisjAre Max est polynomial dans un anneau, ce résultat a été montré par Frank [30], la preuve utilise le théorème de Okamura et Seymour [73] pour les problèmes de multiflot dans les graphes planaires.

Chapitre 3

Résolution des problèmes dans les arbres orientés

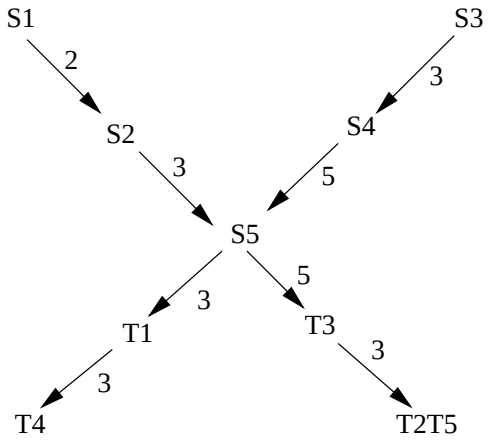
$IMFP$ et $IMCP$ sont Max SNP-difficiles dans les arbres non orientés ([35]), cette difficulté s'étend-t-elle aux arbres orientés ? Ce chapitre donne la réponse à cette question. En effet, nous montrons, par différentes méthodes, que $IMFP$ et $IMCP$ sont polynomiaux dans des arbres orientés. Nous proposons ensuite un algorithme polynomial permettant de résoudre ces deux problèmes dans des arborescences.

3.1 Complexité de $IMFP$ et $IMCP$ dans les arbres orientés

Dans cette section, nous montrons la polynomialité de $IMFP$ et $IMCP$ dans les arbres orientés, nous montrons que cela implique également la polynomialité de FlotInSep Max, CapChem Max et ChemDisjAre Max dans les arbres orientés.

Revenons tout d'abord sur la notion de matrice totalement unimodulaire. Une matrice totalement unimodulaire est une matrice dont toutes les sous-matrices carrées ont leurs déterminants qui valent -1, 0 ou 1. Plus formellement, une matrice A est totalement unimodulaire si et seulement si les points extrêmes du polyèdre $\{x \mid Ax \leq b,\ x \geq 0\}$ sont entiers pour tout vecteur entier b [48]. L'importance de ces matrices réside dans le fait qu'elles correspondent exactement aux matrices de contraintes des programmes entiers pour lesquels on peut relaxer les contraintes d'intégrité sans altérer la solution optimale. Cette notion va nous permettre d'énoncer le résultat suivant sur les arbres orientés.

Lemme 3.1 : $IMFP$ et $IMCP$ sont polynomiaux dans des arbres orientés.



	F1	F2	F3	F4	F5
(S1,S2)	1	0	0	0	0
(S2,S5)	1	1	0	0	0
(S3,S4)	0	0	1	0	0
(S4,S5)	0	0	1	1	0
(S5,T1)	1	0	0	1	0
(T1,T4)	0	0	0	1	0
(S5,T3)	0	1	1	0	1
(T3,T2)	0	1	0	0	1

FIG. 3.1 – Un exemple pour $IMFP$ dans un arbre orienté et la matrice des contraintes associée.

Preuve : La matrice de contraintes du problème de multiflot dans un arbre orienté est une sous-matrice de chaînes si le problème est formulé avec le modèle arc-chemin. Une matrice de chaînes est une matrice dont les colonnes sont les vecteurs de tous les chemins orientés dans un graphe. Dans un arbre, il n'existe qu'une seule chaîne associée à chaque flot. Camion [11] a montré qu'une matrice de chaînes définie sur un arbre orienté est totalement unimodulaire. Donc la matrice de contraintes du problème de multiflot, qui est une sous-matrice d'une matrice totalement unimodulaire, est totalement unimodulaire (voir figure 3.1 pour un exemple).

Le programme mathématique de la multicoupe est le dual de celui du multiflot, donc la transposée de la matrice du multiflot est la matrice de contraintes de la multicoupe qui est ainsi également totalement unimodulaire. Les deux problèmes entiers peuvent donc être résolus par la programmation linéaire : les problèmes de multiflot entier maximal et de multicoupe minimale sont polynomiaux dans des arbres orientés.

La polynomialité de ces problèmes dans les arbres orientés peut être prouvée de deux autres manières :

De la même manière que précédemment nous pouvons montrer que la matrice des contraintes de $(IMFP)$, avec la modélisation arc-chemin, est une matrice de réseau. Soit un arbre orienté $T = (V, A_1)$ et un graphe orienté $D = (V, A_2)$, avec $|V| = m_1 + 1$, $|A_1| = m_1$ et $|A_2| = m_2$, la matrice $m_1 \times m_2$ d'incidence arc-chemin, notée $M(T, D)$, correspondant aux chemins de T dont les extrémités sont définies par les arcs de A_2, est appelée une matrice de réseau.

Il suffit de prendre comme arbre orienté notre arbre et comme graphe orienté le graphe suivant :

Les sommets sont les sommets de l'arbre et on associe un arc (x, y) pour chaque flot k avec x le sommet où se trouve la source s_k et y le sommet où se trouve le puits t_k. Si ensuite on associe la matrice de réseau alors on retrouve la matrice de contraintes de $(IMFP)$. Ainsi, comme dans le paragraphe précédent, comme la matrice des contraintes de $(IMCP)$ est la transposée de celle de $(IMFP)$, on en déduit que les deux matrices sont totalement unimodulaires et donc que les deux problèmes sont polynomiaux dans des arbres orientés.

La dernière méthode que l'on peut appliquer afin de montrer que $IMFP$ est polynomial dans un arbre orienté est de montrer que $IMFP$ se réduit à un problème de circulation de coût minimal dans un autre graphe orienté avec un vecteur de coûts, associés aux arcs, composé de 0 et -1. Rappelons que le problème de circulation de coût minimal consiste à trouver une circulation, i.e. un flot qui respecte la loi de conservation aux nœuds pour tous les sommets du graphe (même en s_k et en t_k), de coût minimal dans un graphe.

Afin de montrer cette réduction, nous affectons un coût de 0 aux arcs originaux de l'arbre orienté, puis nous ajoutons les arcs (t_k,s_k), $k = 1, .., K$, à l'arbre orienté d'origine, en donnant à chacun de ces arcs ajoutés un coût de -1 par unité de flot, c'est-à-dire que pour chaque unité de flot routé sur l'arc (t_k,s_k), le coût diminuera de -1. Notons que dans le nouveau graphe orienté, on a toujours un chemin unique s_k–t_k pour chaque paire $\{s_k, t_k\}$. Le problème de circulation de coût minimal dans un graphe orienté peut être résolu en temps polynomial [38]. On peut ensuite aisément décomposer la solution en circuits correspondants aux flots entiers de l'arbre d'origine (car à chaque arc (t_k,s_k) correspond un circuit unique). Le meilleur algorithme existant pour résoudre ce problème est celui de Orlin (voir [75]), cet algorithme est en $O((m + n\,log\,n)m\,log\,n)$. Si nous l'appliquons pour résoudre $IMFP$ dans les arbres orientés, le nombre d'arcs du graphe orienté obtenu après l'ajout des arcs (t_k,s_k) est $n+K-1$, ainsi la complexité de l'algorithme d'Orlin est en $O((n + K - 1) + n\,log\,n)\,(n + K - 1)\,log\,n)$, nous obtenons donc une complexité en $O\,(max\,(K^2 log\,n,\, n^2 log^2 n))$, soit $O\,(n^2 log^2 n)$ quand $K = O(n)$ et $O\,(n^4 log\,n)$ quand $K = O\,(n^2)$.

Comme cela est dit dans la section 1.2.3, $IMFP$ et FlotInSep Max sont équivalents dans un arbre car il n'existe qu'un seul chemin entre une source et un puits, on peut donc en déduire que FlotInSep Max est également polynomial dans les arbres orientés. La matrice des contraintes de CapChem Max est la même que celle de $(IMFP)$, car entre les deux problèmes, seules les valeurs des variables changent, on en déduit donc que la matrice des contraintes de CapChem Max est totalement unimodulaire et que par conséquent, CapChem Max est polynomial dans les arbres orientés. ChemDisjAre Max est un cas particulier de $IMFP$ avec des capacités unitaires sur les arcs et comme $IMFP$ est polynomial sur les arbres

orientés, alors ChemDisjAre Max est polynomial sur les arbres orientés.

3.2 Un algorithme polynomial pour $IMFP$ et $IMCP$ dans les arborescences

On vient de voir que $IMFP$, FlotInSep Max, CapChem Max, ChemDisjAre Max, et $IMCP$ sont polynomiaux dans les arbres orientés, ce résultat reste valable dans les arborescences, car une arborescence est un arbre orienté particulier. Cet arbre orienté admet un sommet, la racine, à partir duquel il existe un chemin menant à tout autre sommet de l'arbre.

Rappelons que l'on note f_i la valeur du flot F_i routé sur le chemin p_i de s_i à t_i.

Nous supposons dans cette section que le graphe G est une arborescence et nous proposons un algorithme polynomial gourmand pour résoudre le problème du multiflot maximal en nombres entiers et le problème de la multicoupe minimale. Les formulations $(IMFP)$ et $(IMCP)$ présentées dans l'introduction restent les mêmes et on a en plus l'égalité suivante : $K = M$, car dans un arbre, il existe un seul chemin $p_i \in \Pi$ pour chaque flot de s_i à t_i. L'idée de base de l'algorithme est de trouver des solutions entières vérifiant les conditions des écarts complémentaires de la programmation linéaire données dans la section 2.1.

D'après ce qui précède, les conditions d'optimalité des écarts complémentaires appliquées à $(IMFP)$ et $(IMCP)$ dans les arborescences sont données par :

$\forall i = 1, .., K \;\; f_i^* > 0 \;\Rightarrow\; \sum_{e \in p_i} c_e^* = 1 \quad (6)$

$\forall e \in E \;\; c_e^* > 0 \;\Rightarrow\; \sum_{i \; s.t. \; e \in p_i} f_i^* = u_e \quad (7)$

Les conditions (6) signifient que dans toute solution optimale soit le flot sur le chemin p_i est égal à 0 soit la contrainte $\sum_{e \in p_i} c_e \geq 1$ associée est saturée, $\forall i = 1, .., K$. Si les variables c_e^* sont entières alors il y a un et un seul arc de p_i dans la multicoupe pour tout i tel que $f_i^* > 0$. Les conditions (7) signifient que dans toute solution optimale si l'arc e n'est pas saturé par le flot alors $c_e^* = 0$; cela signifie que tous les arcs de la coupe sont saturés.

3.2.1 Recherche d'un multiflot entier

Nous ordonnons les sommets de l'arborescence selon un ordre lexicographique, la racine est le sommet a, puis nous affectons une lettre, selon l'ordre lexicographique, à chaque sommet en parcourant l'arbre en largeur et en descendant niveau par niveau jusqu'à atteindre les feuilles. L'ordre sur les sommets est obtenu en parcourant tous les sommets de l'arborescences, i.e. en $O(n)$. La procédure commence à partir des feuilles de l'arborescence et couvre ensuite les

sommets niveau par niveau jusqu'à la racine. A chaque fois que l'on rencontre une source s_k, on route une quantité maximale de flot sur p_k, saturant au moins un nouvel arc si $f_k > 0$. S'il y a plusieurs sources à un sommet, tout ordre peut être considéré pour ces sources. Voir la section 3.2.3 pour un exemple.

L'ensemble d'arcs $C_0 \subseteq E$ contiendra tous les arcs saturés tout au long de la procédure.

Algorithm 1 $IMFP$ dans une arborescence.

procédure $Maxmultiflot$;
entrée : $T = (V, E)$, soit $a, b, c...$ un ordre lexicographique, ordonné selon la largeur de l'arbre, sur les sommets de V
$(s_k, t_k) \in V^2 \ \forall k \in \{1, .., K\}$.
sortie : $(F_1^*, ..., F_k^*, ..., F_K^*)$ *un multiflot maximal.* C_0 l'ensemble des arcs saturés par le multiflot.
1. On numérote les sources (ainsi que les puits et les flots) de la racine vers les feuilles de 1 à K ;
2. $C_0 \leftarrow \emptyset$;
3. **pour** $k = K$ à 1 **faire**
//on explore les sources du bas vers le haut
on route le flot maximal F_k^* de s_k à t_k, en respectant les capacités résiduelles ;
$E_k \leftarrow \{$ nouveaux arcs saturés par $F_k^* \}$;
$C_0 \leftarrow C_0 \bigcup E_k$;
//il existe au moins un arc de plus dans C_0 *si* $f_k^* > 0$ *;*
fin faire ;
fin $Maxmultiflot$;

Par hypothèse, les capacités sont entières, donc tous les flots F_k routés sont entiers. A la fin de la procédure la valeur du multiflot entier $\widehat{F}$ est égale à $\Phi = \sum_{k=1}^{K} f_k^*$ et il existe au moins un arc saturé (dans C_0) sur chaque chemin p_k.

3.2.2 Recherche d'une multicoupe

La multicoupe est construite à partir de la solution du multiflot obtenue précédemment. La procédure considère les sources dans l'ordre donné par la numérotation effectuée dans $Maxmultiflot$, de la racine vers les feuilles. Pour satisfaire les conditions (7) des écarts complémentaires, tous les arcs de la coupe sont pris dans C_0 et sont donc des arcs saturés. De plus, afin de satisfaire les conditions (6), si $f_k^* > 0$ alors nous ne gardons qu'un arc e_k sur le chemin p_k. A l'étape k de l'algorithme, on note e_k l'arc qui est soit le seul arc restant sur p_k, soit l'arc saturé le plus haut restant sur p_k, l'arc saturé le plus haut étant le premier arc rencontré sur le chemin de la source s_k au puits t_k. Nous enlevons alors définitivement tous les autres arcs du chemin p_k qui sont dans la coupe C_{k-1}, obtenue à l'étape précédente. En faisant ainsi, nous

obtenons un ensemble $\widehat{C} \subseteq E$ qui contient au plus un arc sur chaque chemin p_k tel que $f_k^* > 0$. Il reste à vérifier qu'il reste dans $\widehat{C}$ au moins un arc saturé par chemin p_k, $\forall k \in \{1,..,K\}$, et donc que $\widehat{C}$ est une multicoupe. Soit Γ la valeur de l'ensemble $\widehat{C}$, $\Gamma = \sum_{e \in \widehat{C}} u_e$. Dans la section qui suit, nous allons prouver que $\widehat{C}$ est une multicoupe et que $\Gamma = \Phi$.

Algorithm 2 *IMCP* dans une arborescence.

procédure *Minmulticoupe*
entrée : $T = (V, E)$, $(s_k, t_k) \in V^2$ $\forall k \in \{1,..,K\}$, F^*, C_0=*{arcs saturés par le multiflot}*
sortie : une multicoupe minimale $\widehat{C}$.
Pour $k = 1,..,K$ **faire**
  //On considère la numérotation donnée par Maxmultiflot.
  si $f_k^* > 0$ *et* $|C_{k-1} \bigcap p_k| > 1$ **alors**
    $e_k \leftarrow$premier arc saturé sur le chemin de s_k à t_k ;
    $C_k \leftarrow [C_{k-1} - (C_{k-1} \bigcap p_k)] \bigcup \{e_k\}$;
    //$C_{k-1} \bigcap p_k$ est un ensemble d'arcs dont les capacités résiduelles sont égales à zéro
    //On supprime dans C_{k-1} les arcs se trouvant sur p_k sauf e_k
  sinon
    //soit $f_k^* > 0$ et on a au plus un arc de p_k dans C_{k-1}
    //soit $f_k^* = 0$: plus d'un arc de $\widehat{C}$ est autorisé sur p_k. Rien à faire
    $C_k \leftarrow C_{k-1}$;
  fin si ;
fin faire ;
$\widehat{C} \leftarrow C_K$;
fin *Minmulticoupe* ;

A la fin de la procédure, on a au plus un arc de $\widehat{C}$ sur chaque chemin p_k tel que $f_k^* > 0$.

3.2.3 Exemple

Appliquons notre algorithme sur l'arborescence donnée dans la figure 3.2.

Les sommets sont ordonnés selon l'ordre lexicographique : a, b, c, d, e, f, g *et* h . Les sources sont numérotées de s_1 à s_7 de la racine vers les feuilles.

A la première étape de la procédure *Maxmultiflot*, nous routons F_7, nous obtenons $f_7 = 2$ et l'arc (f, h) est saturé : $C_0 = \{(f, h)\}$.

A la deuxième étape, nous routons F_6, nous obtenons $f_6 = 3$ et l'arc (c, f) est saturé : $C_0 = \{(c, f), (f, h)\}$.

A la troisième étape, nous ne pouvons pas router F_5 car (c, f) est saturé, d'où $f_5 = 0$ et $C_0 = \{(c, f), (f, h)\}$.

A la quatrième étape, nous routons F_4, nous obtenons $f_4 = 1$ et les arcs (c, e) et (b, c) sont saturés : $C_0 = \{(b, c), (c, e), (c, f), (f, h)\}$.

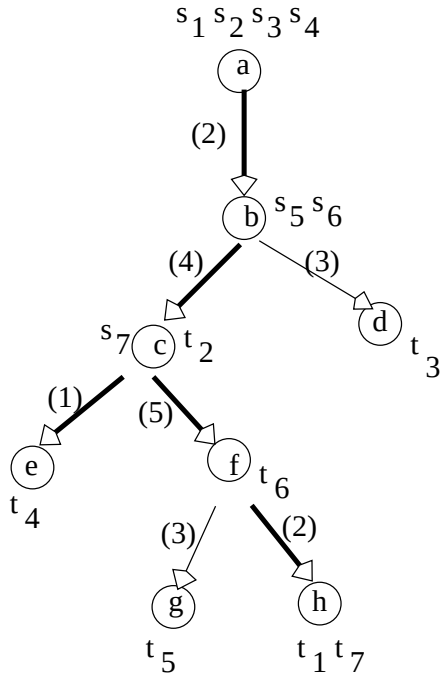


FIG. 3.2 – Application de l'algorithme

A la cinquième étape, nous routons F_3, nous obtenons $f_3 = 1$ et l'arc (a, b) est saturé : $C_0 = \{(a, b),\ (b, c),\ (c, e),\ (c, f),\ (f, h)\}$.

A la sixième étape, nous ne pouvons pas router F_2 car (a, b) et (b, c) sont saturés, d'où $f_2 = 0$ et $C_0 = \{(a, b),\ (b, c),\ (c, e),\ (c, f),\ (f, h)\}$.

A la septième étape, nous ne pouvons pas router F_1 car (a, b), (b, c), (c, f) et (f, h) sont saturés, d'où $f_1 = 0$ et $C_0 = \{(a, b),\ (b, c),\ (c, e),\ (c, f),\ (f, h)\}$.

Les valeurs des flots obtenus à la fin de la procédure $Maxmultiflot$ sont donc :

$f_7^* = 2,\ f_6^* = 3,\ f_5^* = 0,\ f_4^* = 1,\ f_3^* = 1,\ f_2^* = 0,\ f_1^* = 0.$

Nous obtenons $\Phi = 7$ et $C_0 = \{(a, b),\ (b, c),\ (c, e),\ (c, f),\ (f, h)\}$.

A la première étape de la procédure $Minmulticoupe$, $f_1^* = 0$, d'où

$C_1 = C_0 = \{(a, b),\ (b, c),\ (c, e),\ (c, f),\ (f, h)\}$.

A la deuxième étape, $f_2^* = 0$, d'où $C_2 = C_1 = C_0 = \{(a, b),\ (b, c),\ (c, e),\ (c, f),\ (f, h)\}$.

A la troisième étape, $f_3^* = 1$ et (a, b) est le seul arc de p_3 dans C_2, d'où $C_3 = C_2 = C_1 = C_0 = \{(a, b),\ (b, c),\ (c, e),\ (c, f),\ (f, h)\}$.

A la quatrième étape, $f_4^* = 1$ et (a, b) est le premier arc saturé de p_4, (b, c) et (c, e) doivent être retirés de C_3, d'où $C_4 = C_3 - \{(b, c),\ (c, e)\} = \{(a, b),\ (c, f),\ (f, h)\}$.

A la cinquième étape, $f_5^* = 0$, d'où $C_5 = C_4 = \{(a,b), (c,f), (f,h)\}$.

A la sixième étape, $f_6^* = 3$ et (c,f) est le seul arc de p_6 dans C_5, d'où $C_6 = C_5 = C_4 = \{(a,b), (c,f), (f,h)\}$.

A la septième étape, $f_7^* = 2$ et (c,f) est le premier arc saturé de p_7, (f,h) doit être retiré de C_6, d'où $C_7 = C_6 - \{(f,h)\} = \{(a,b), (c,f)\}$.
A la fin de la procédure *Minmulticoupe*, nous obtenons $\widehat{C} = \{(a,b),(c,f)\}$ et $\Gamma = 7$.
$\Gamma = \Phi = 7$ et on a un et un seul arc de $\widehat{C}$ sur chaque chemin p_k sauf p_1 qui peut admettre deux arcs de $\widehat{C}$ car $f_1^* = 0$.

3.2.4 Optimalité de l'algorithme

L'algorithme consiste à exécuter la procédure $Maxmultiflot$ puis la procédure *Minmulticoupe* afin d'obtenir un multiflot $\widehat{F}$ et une multicoupe $\widehat{C}$. Les flots sont routés dans l'ordre décroissant de la numérotation et $\widehat{C}$ est obtenu selon l'ordre croissant de celle-ci. Pour prouver l'optimalité de l'algorithme, nous montrerons d'abord que $\widehat{C}$ est une multicoupe dont le retrait séparerait chaque paire $\{s_k, t_k\}$, $k = 1, .., K$, puis qu'il n'existe pas de saut de dualité entre les valeurs du multiflot et de la multicoupe.

Lemme 3.2 : L'ensemble $\widehat{C} \subseteq E$ obtenu à la fin de l'algorithme contient au moins un arc sur chaque chemin p_k de s_k à t_k, $k = 1, .., K$: $\widehat{C}$ est une multicoupe.

Preuve : Nous donnons une preuve par récurrence sur les ensembles C_i et les chemins p_i.

1. Dans la procédure $Maxmultiflot$, nous saturons au moins un arc sur chaque p_k, $k = 1, .., K$, donc il existe au moins un arc saturé sur chaque p_k, $k = 1, .., K$ dans l'ensemble C_0.

2. Supposons que le lemme est vrai pour l'ensemble C_{i-1} $(i > 0)$.

3. Montrons que le lemme est toujours vrai pour C_i.

Si $f_i^* = 0$ alors $C_i = C_{i-1}$ et le lemme est vrai pour C_i.

Étudions maintenant $C_i \bigcap p_k$ pour i tel que $f_i^* > 0$ et pour tous les chemins p_k, $k = 1, .., K$.

Il existe au moins un arc de p_i dans C_{i-1} et à l'étape i nous gardons un arc de p_i, donc le lemme est vrai pour $k = i$.

Tous les chemins p_k, $k = 1, .., K$, $k \neq i$ vérifient un des trois cas suivants :

(a)$p_k \bigcap p_i = \emptyset$. Il y a un arc de p_k dans C_{i-1} et au pas i de l'algorithme, nous supprimons uniquement des arcs de p_i pour obtenir C_i, les arcs supprimés ne peuvent appartenir à p_k puisque $p_k \bigcap p_i = \emptyset$: il y a donc un arc de p_k dans C_i également.

(b)$p_k \bigcap p_i \neq \emptyset$ et $k < i$. Dans l'algorithme, on route F_k après F_i (dans $Maxmultiflot$) et on obtient C_k avant C_i (dans *Minmulticoupe*) : on a donc $C_i \subseteq C_{i-1} \subseteq C_k$. Il existe un arc e_k de p_k dans C_{i-1} et comme on obtient C_k avant C_i dans *Minmulticoupe* : e_k devient l'unique

arc de p_k dans C_k, donc e_k est l'unique arc de p_k dans C_{i-1} (car $C_{i-1} \subseteq C_k$). Deux sous-cas doivent maintenant être considérés :

cas b-1 : $e_k \notin p_i$ et e_k n'est pas supprimé par *Minmulticoupe* au pas i car nous supprimons uniquement des arcs de p_i.

cas b-2 : $e_k \in p_i$ et d'après la numérotation des sources, nous savons que s_i est sur un sommet situé, dans l'arborescence, en dessous du sommet auquel s_k appartient (voir figure 3.3). e_k est le premier arc saturé de C_{i-1} sur le chemin de s_k à t_k, c'est donc aussi le premier arc saturé sur le chemin de s_i à t_i et donc e_k $(= e_i)$ n'est pas supprimé par *Minmulticoupe* au pas i.

(c)$p_k \bigcap p_i \neq \emptyset$ et $k > i$ (voir figure 3.3). Dans l'algorithme, on route F_k avant F_i. Il existe un arc $v \in p_k$ que l'on sature dans $Maxmultiflot$ soit avant de router F_k $(f_k^* = 0)$ soit en routant F_k $(f_k^* > 0)$. Pour tout j tel que $j < k$ et $f_j^* > 0$ nous avons $v \notin p_j$ (sinon, comme on route F_j après F_k, il y aurait un arc saturé v sur p_j et F_j serait égal à 0). $v \in C_0$ et on ne retire pas v de C_j, pour tout $j < k$, dans *Minmulticoupe* car $v \notin p_j$; donc $v \in C_{k-1}$. Mais $i < k$ donc $C_{k-1} \subseteq C_i$ et donc $v \in C_i$.

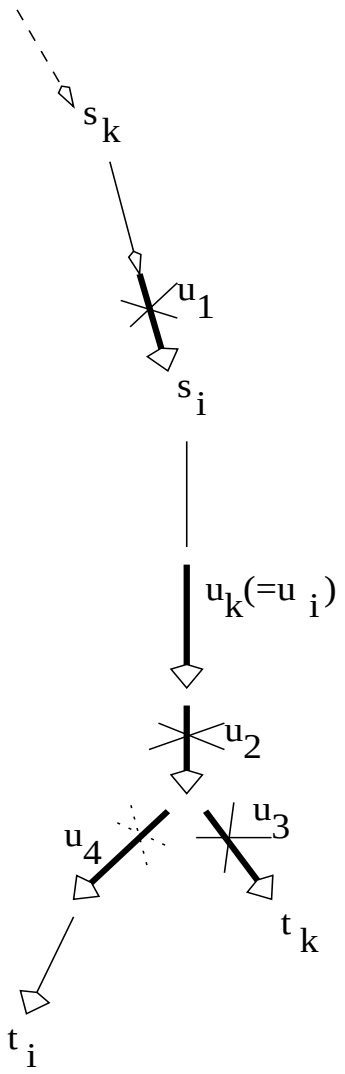
Finalement, pour tout $k = 1, .., K$ il existe un arc de p_k dans C_i : C_i est une multicoupe.

Lemme 3.3 : Soit Φ et Γ les valeurs du multiflot entier $\widehat{F}$ et de la multicoupe $\widehat{C}$ obtenus à la fin de l'algorithme. Alors, $\Phi = \Gamma$, $\widehat{F}$ est un multiflot entier maximal et $\widehat{C}$ est une $s_k - t_k$ multicoupe minimale.

Preuve : A la fin de l'algorithme, $\widehat{C}$ contient un et un seul arc sur chaque chemin p_k tel que $f_k^* > 0$. La solution du programme $(IMCP)$ est donnée par $[e \in \widehat{C} \Rightarrow c_e = 1\,;$ $e \notin \widehat{C} \Rightarrow c_e = 0\,;$ pour tout $e \in E]$. Les conditions des écarts complémentaires données par les contraintes (3) et (4) sont vérifiées. En effet, si $f_k^* > 0$, alors par la procédure *Minmulticoupe* nous savons qu'il existe seulement un arc de p_k dans $\widehat{C}$ donc $\sum_{e \in p_k} c_e = 1$; de plus, tous les arcs dans $\widehat{C}$ appartiennent à C_0 et sont des arcs saturés, donc si $c_e > 0$, alors $\sum_{k\, t.q.\, e \in p_k} f_k^* = u_e$. Les solutions de $IMCP$ et $IMFP$ associées à $\widehat{C}$ et $\widehat{F}$ vérifient les conditions des écarts complémentaires de la programmation linéaire donc elles sont optimales et $\Phi = \Gamma$. De plus, les solutions sont entières et donc il n'y a pas de saut de dualité pour ces deux problèmes.

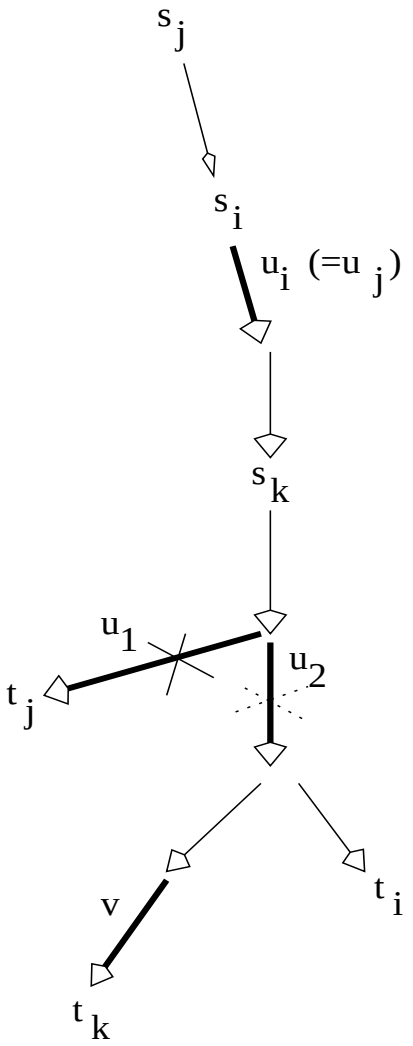
3.2.5 Complexité de l'algorithme

Premièrement, il est facile de vérifier que la procédure *Minmulticoupe* peut être implémentée en $O\left(min\left(Kn,\, n^2\right)\right)$. En effet, pour tout $k \in \{1, ..., K\}$, $|C_k| \leq n$ et le chemin p_k a moins de n arcs. Il y a au plus $min(K, n)$ flots positifs f_k^*, car chaque flot $f_k^* > 0$ sature au



case b-2

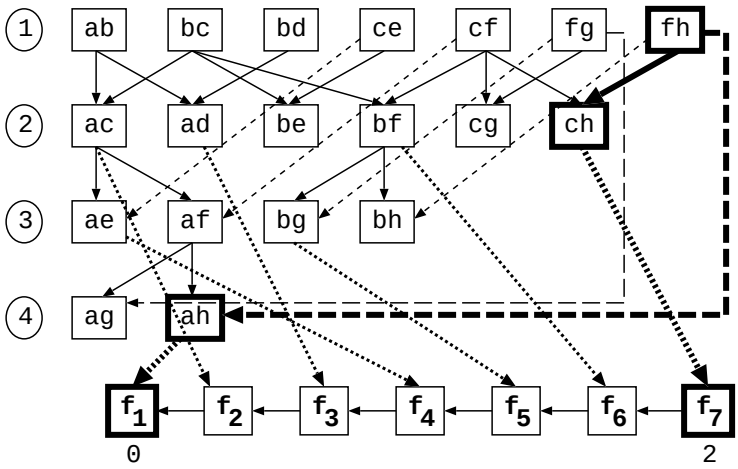
cas b-2. $k < i$ on supprime l'arc u_1 *à* l'étape j, $j < k$;
on supprime les arcs u_2 et u_3 *à* l'étape k ;
on supprime l'arc u_4 *à* l'étape i ;
on ne supprime pas l'arc u_k *à* l'étape i ;



case c

cas c. $k > i$ on supprime l'arc u_1 *à* l'étape j ;
on supprime l'arc u_2 *à* l'étape i ;
on ne supprime pas l'arc v *à* l'étape i ;

FIG. 3.3 – Preuve du Lemme 3.2 : pas i avec $p_k \bigcap p_i \neq \emptyset$



F_7 est routé ($f_7^* = 2$) et nous le supprimons de L, l'arc fh est saturé ;
fh, ch, ah et F_1 sont supprimés de la structure de données, f_1^* est mis à 0.

FIG. 3.4 – $K = O\left(n^2\right)$: structure de données associée à la figure 3.2

moins un arc de l'arbre, et le calcul de e_k et de C_k peut se faire en $O(n)$. La partie **sinon** de la procédure est en $K \times O(1)$, soit $O(K)$. Sa complexité globale est donc $O(n\,min(K,n)+K)$ *i.e.* $O\left(min\left(Kn,\, n^2\right)\right)$.

Deuxièmement, la procédure $Maxmultiflot$ peut être implémentée en $O\left(Kn\right)$. En effet, au pas 1 de la procédure, chaque source est vue une seule fois, donc ce pas peut être effectué en $O\left(max\left(K,n\right)\right)$. Considérons maintenant le pas 3. Router chaque flot F_k s'effectue en $O(n)$ dans un arbre. Nous devons étudier deux cas.

Si $K = O(n)$ alors le pas 3 est en $O(Kn)$, soit $O\left(n^2\right)$, donc la complexité globale de $Maxmultiflot$ est $O\left(n + n^2\right)$, soit $O\left(n^2\right)$.

Si $K = O\left(n^2\right)$ alors le pas 3 est en $O(Kn)$, soit $O\left(n^3\right)$, donc la complexité globale de $Maxmultiflot$ est $O\left(n^2 + n^3\right)$, soit $O\left(n^3\right)$. Nous allons utiliser une structure de données particulière afin d'améliorer la complexité de la procédure $Maxmultiflot$ (pour qu'elle soit identique à celle de $Minmulticoupe$).

Nous utilisons une liste chaînée, notée L, des flots ordonnés de F_K à F_1 (dernière ligne de la figure 3.4) et une liste des arcs de E ordonnés selon l'ordre lexicographique (première ligne de la figure 3.4). Il existe plusieurs manières d'obtenir un chemin de longueur j ($j > 2$) : nous avons choisi d'ajouter un arc à la fin d'un chemin de longueur $j-1$. Par exemple, le chemin ah de la figure 3.2 est obtenu par af et fh (et non par ab et bh). Pour construire un chemin de longueur j, nous définissons donc deux pointeurs, l'un partant d'un arc du niveau 1 et l'autre

partant d'un chemin de longueur $j-1$. Nous définissons également un pointeur partant de chaque chemin et allant au flot correspondant, s'il existe, du dernier niveau. Ainsi nous avons au plus deux pointeurs pour chaque chemin à chaque niveau $j > 2$. Les pointeurs décrivent les inclusions des chemins (pas toutes les inclusions, mais suffisamment pour assurer que tous les chemins contenant un arc e puisssent être atteints à partir de e dans notre structure de données).

Les noms des sommets ayant été donnés selon l'ordre lexicographique, construire la liste du niveau j nécessite de couvrir une seule fois la liste du niveau $j-1$ et dans le même temps la liste du niveau 1. La liste du premier niveau est ordonnée selon l'ordre lexicographique donc les listes des niveaux $2, .., j$ sont naturellement obtenues selon l'ordre lexicographique en $O(n)$.

Il y a au plus $n-1$ chemins de longueur j pour tout j dans $\{1, ..., n-1\}$, le nombre total de chemins est ainsi au plus de n^2, et le nombre total de pointeurs est inférieur à $2n^2$, car nous définissons au plus deux pointeurs pour chaque chemin. De plus, nous assignons un pointeur partant de chaque chemin et allant au flot correspondant dans L s'il existe (K pointeurs). Finalement, le nombre total de pointeurs est inférieur à $2n^2+K$, ainsi nous pouvons construire notre structure de données en $O\left(n^2\right)$.

Pour prendre en compte cette nouvelle structure de données, nous avons besoin de modifier légèrement la procédure $Maxmultiflot$ dans le cas où $K = O\left(n^2\right)$: les changements sont indiqués entre crochets dans la procédure $Maxmultiflotbis$.

Cette nouvelle procédure va nous permettre de ne rencontrer qu'une seule fois chaque flot (nul ou non nul) dans notre structure de données. A chaque itération de la boucle 3 de la procédure $Maxmultiflotbis$, nous routons d'abord un flot positif F_k et nous le supprimons de L. Nous considérons ensuite les nouveaux arcs saturés. Grâce à notre structure de données, nous mettons à zéro tous les flots non encore routés qui correspondent à des chemins contenant de nouveaux arcs saturés ; nous supprimons ces flots de la structure de données. Il y a au plus n flots positifs à router. Chaque routage s'effectue en $O(n)$ et implique la suppression d'une partie de la structure de données. Le routage total s'effectue ainsi en $O\left(n^2\right)$ et la structure est parcourue une seule fois pour supprimer les $K = O\left(n^2\right)$ flots de la structure : la complexité globale de la procédure $Maxmultiflotbis$ quand $K = O\left(n^2\right)$ est donc en $O\left(n^2\right)$.

Finalement, la procédure $Maxmultiflotbis$ peut s'effectuer en $O\left(min\left(Kn,\, n^2\right)\right)$. Comme on doit exécuter $Maxmultiflotbis$ avant $Minmulticoupe$, la complexité globale de l'algorithme est en $O\left(min\left(Kn,\, n^2\right)\right)$.

Prenons une chaîne orientée sur laquelle se trouvent $\frac{n}{2}$ flots, chaque chemin étant de longueur $\frac{n}{2}$. La complexité globale de l'algorithme est alors $O\left(2 \times \frac{n}{2} \times \frac{n}{2}\right)$, soit $O\left(n^2\right)$.

Algorithm 3 Modifications pour $IMFP$ dans une arborescence.

procédure $Maxmultiflotbis$;
entrée : $T = (V, E)$, soit $a, b, c...$ un ordre lexicographique, ordonné selon la largeur de l'arbre, sur les sommets de V
$(s_k, t_k) \in V^2 \ \forall k \in \{1, .., K\}$.
sortie : $(F_1^*, ..., F_k^*, ..., F_K^*)$ *un multiflot maximal.* C_0 l'ensemble des arcs saturés par le multiflot.
1. On numérote les sources (ainsi que les puits et les flots) de la racine vers les feuilles de 1 à K ;
2. $C_0 \leftarrow \emptyset$; **[*si*** $K = O(n^2)$ ***alors ajouter*** $L \leftarrow$**liste**$\{F_k, F_{k-1}, .., F_1\}$ **;]**
3. **pour** $k = K$ à 1 **faire** **[*si*** $K = O(n^2)$ ***alors remplacer par*** **tant que** $L \neq \emptyset$ **faire]**
[*si* $K = O(n^2)$ ***alors ajouter*** $k \leftarrow$**indice du premier flot de** L **;]**
//on explore les sources du bas vers le haut
on route le flot maximal F_k^* de s_k à t_k, en respectant les capacités résiduelles ;
$E_k \leftarrow$ { nouveaux arcs saturés par F_k^* } ; **[*si*** $K = O(n^2)$ ***alors ajouter :*** $L \leftarrow L - \{F_k\}$ **;]**
$C_0 \leftarrow C_0 \bigcup E_k$;
//il existe au moins un arc de plus dans C_0 *si* $f_k^* > 0$;
[***si*** $K = O(n^2)$ ***alors ajouter :*** **pour** F_i **tel que** $p_i \cap E_k \neq \emptyset$ **faire**
$f_i^* \leftarrow 0$; $L \leftarrow L - \{F_i\}$ **fin faire** ;
//$\exists$ ***un arc saturé sur*** p_i **]**
fin faire ;
fin $Maxmultiflotbis$;

Nous pouvons remarquer aisément à l'aide du contre-exemple suivant que la procédure gourmande utilisée pour résoudre $(IMCP)$ et $(IMFP)$ dans les arborescences ne peut s'appliquer aux arbres orientés. Modifions légèrement l'exemple donné dans la figure 3.2 : ajoutons à l'arbre un arc $(a'b)$ avec un poids de 1 ($u_{a'b} = 1$) et un flot f_0 avec s_0 en a' et t_0 en e. La solution, du multiflot, obtenue par l'algorithme est la solution de la figure 3.2 avec $f_0 = 0$ et $\Phi = 7$ alors que la solution optimale du multiflot est la suivante : $F^* = (1, 0, 0, 2, 0, 0, 3, 2)$ avec $\Phi^* = 8$ (voir figure 3.5).

Nous avons également essayé de trouver un autre ordre pour le routage des flots et pour le choix des arcs de la coupe optimale, mais pour chaque ordre que nous avons créé, nous avons trouvé un contre-exemple dans lequel cet ordre ne permettait pas d'obtenir une solution optimale. On peut néanmoins remarquer que l'on peut mettre l'arbre orienté en niveaux, en suivant l'orientation de l'arbre comme sur la figure 3.5. Mais on peut également remarquer que cette mise en niveaux de l'arbre peut ne pas faciliter la résolution du problème. En effet, sur l'exemple de la figure 3.5, s_0 et s_4 sont au même niveau, tout comme t_0 et t_4, mais si l'on route f_4 avant de router f_0 alors on ne pourra pas obtenir une solution optimale pour le multiflot car : $f_5 + f_6 + f_7 \leq 5$ car ces flots passent tous sur l'arc (c,f) et $f_0 + f_1 + f_2 + f_3 + f_4 \leq 2$ car

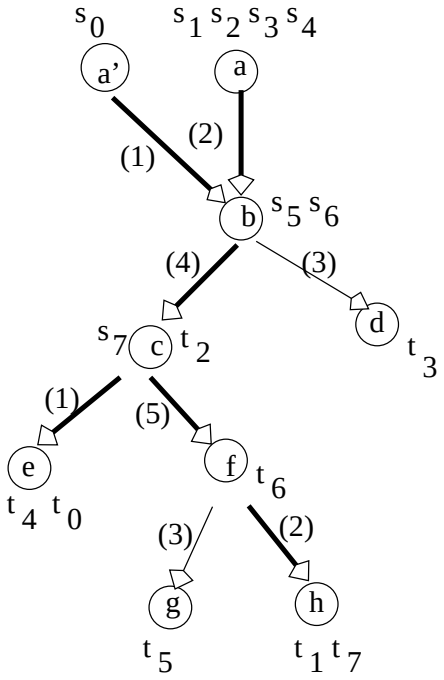


FIG. 3.5 – Solution pour $IMFP$ dans un arbre orienté

l'arc (c,e) sera saturé par f_4, donc $f_0 = 0$ et $f_1 + f_2 + f_3 + f_4 \leq 2$ car tous ces flots passent sur l'arc (a,b). Ainsi $f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 \leq 7$ et on sait d'après ce qui précède qu'il existe une solution optimale de valeur 8 pour le multiflot.

Notons que la complexité de l'algorithme a été améliorée par Costa [17] dans le cas où l'on s'intéresse à la résolution des problèmes de COUPE MULTITERM et de FLOT MULTITERM dans les arborescences. En effet, l'auteur montre que résoudre les problèmes de COUPE MULTITERM et de FLOT MULTITERM dans une arborescence, ayant L feuilles, équivaut à résoudre $IMCP$ et $IMFP$ avec un nombre de couples $\{s_k, t_k\}$ égal à L, $k = 1, .., L$, toutes les sources étant situées à la racine de l'arborescence et chaque puits étant affecté à une feuille.

La complexité de la procédure $Maxmultiflot$ adaptée au problème de FLOT MULTITERM dans une arborescence est dans ce cas en $O(Lh)$ avec h la hauteur de l'arborescence, car le nombre de flots est égal à L et chaque chemin est de longueur h.

L'auteur a également montré que l'on peut résoudre le problème de COUPE MULTITERM dans une arborescence en utilisant une procédure simplifiée, pouvant s'effectuer en $O(n)$.

Chapitre 4

Résolution de $IMFP$ et $IMCP$ dans les arbres non orientés

4.1 Orientation d'un arbre

Avant d'étudier une instance de $IMFP$ ou de $IMCP$ dans un arbre non orienté, on peut vérifier s'il est possible d'orienter les arêtes afin d'obtenir un problème orienté équivalent. Pour tout $k \in \{1, .., K\}$, l'instance doit contenir un chemin orienté de s_k à t_k, ou de t_k à s_k. Si c'est le cas, l'instance appartient à une famille de problèmes que l'on peut résoudre en temps polynomial (voir section 3.1). Ceci peut être fait en $O(m)$: premièrement, on oriente les arêtes de p_1 allant de s_1 vers t_1 ; ensuite, tant que c'est compatible avec les orientations précédentes et que l'on choisit une chaîne qui a des arêtes communes avec le précédent chemin sélectionné, on oriente les arêtes de p_k, $k = 1, .., K$, soit de s_k vers t_k, soit de t_k vers s_k (dans ce cas, on permute s_k et t_k). Si aucune de ces orientations n'est compatible alors on arrête le processus (voir figure 4.1).

En appliquant cette méthode, nous vérifions bien s'il s'agit d'une instance que l'on peut orienter ou non. Si à la fin du processus, une orientation de l'arbre est obtenue alors le problème est bien un problème de multiflot maximal dans un arbre orienté. Par contre, si le processus s'arrête avant d'avoir obtenu une orientation de l'arbre, alors on ne peut obtenir une orientation sans changer le problème. En effet, cela signifie qu'il existe au moins deux arcs de directions opposées sur le chemin de s_k à t_k et on ne peut changer l'orientation d'un de ces arcs sans changer également l'orientation de l'autre, car les flots ont été orientés en choisissant des chaînes qui ont au moins une arête commune avec les chemins déjà orientés, ainsi même si on

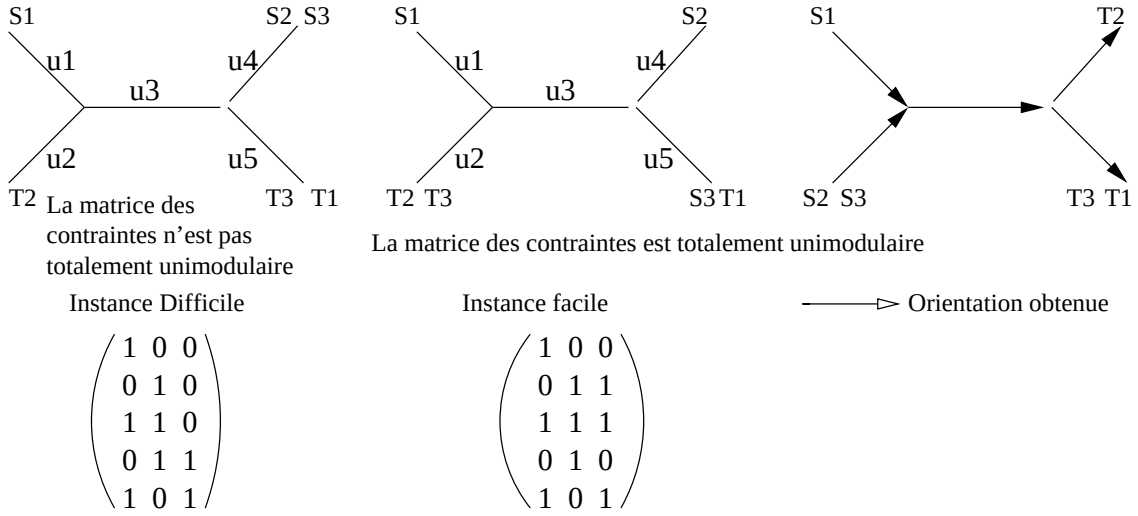


FIG. 4.1 – Instances faciles et difficiles dans des arbres non orientés.

permute s_k et t_k, les deux arcs auront toujours des directions opposées.

Nous nous intéressons maintenant à la résolution de $IMCP$ et de $IMFP$ dans les arbres non orientés, c'est-à-dire les arbres pour lesquels nous ne pouvons obtenir une orientation sans changer le problème.

4.2 Résolution de $IMCP$

Le fait qu'il n'existe probablement pas d'algorithme de complexité polynomiale permettant de résoudre un problème NP-difficile incite à utiliser pour sa résolution une méthode énumérative. La plus couramment utilisée consiste en un algorithme de séparation et d'évaluation (branch and bound) afin de résoudre de façon exacte le problème.

La programmation semi-définie (SDP) est une méthode relativement récente en optimisation combinatoire. Elle fournit souvent des bornes de bonne qualité et même des solutions approchées avec garantie de performance, mais la résolution numérique d'un SDP est en général beaucoup plus coûteuse en temps que celle d'un programme linéaire. Une relaxation SDP peut être obtenue à partir d'un programme linéaire continu en ajoutant certaines contraintes non linéaires mais convexes. La SDP peut donc être considérée comme une généralisation de la programmation linéaire (PL). Une autre façon d'obtenir une relaxation SDP est de partir d'un modèle discret et d'effectuer la relaxation en supprimant une contrainte de rang sur une matrice regroupant les variables du problème. Pour plus de détails, nous renvoyons à [78],

où sont présentés différents problèmes duaux lagrangiens de plusieurs problèmes classiques d'optimisation quadratique en variables bivalentes. La thèse d'habilitation de Helmberg [45] contient également de nombreux exemples d'application de la SDP en optimisation combinatoire, ainsi que des méthodes de résolutions numériques. Par ailleurs, l'outil numérique SB (Spectral Bundle) [46] utilisé pour la résolution des programmes SDP présents dans cette section est une implémentation de ces algorithmes [47]. L'efficacité de ce solveur a été illustré dans [82] pour la résolution du QAP (Quadratic Assignment Problem) où l'auteur montre que la SDP permet d'obtenir de meilleures bornes que la PL pour ce problème. De plus, au cours de la résolution des programmes SDP, la valeur de la borne fournie par la PL est atteinte en un temps comparable et même souvent inférieur à celui nécessaire à la résolution du PL.

4.2.1 Notations, définitions

Dans cette section, nous donnons quelques notations et définitions :

Soient A et B deux matrices $n \times n$ réelles symétriques. Un produit scalaire $\bullet$ des matrices A et B est défini par :

$$A \bullet B = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij} b_{ij}$$

Nous pouvons maintenant écrire toute forme quadratique $x^T Ax + b^T x + c$ sous la forme $A \bullet xx^T + b^T x + c$. Notons $\mathbf{d}(A)$ la diagonale de la matrice A.

Notons $PL\{0,1\}$ une relaxation linéaire du programme considéré et notons PL la relaxation linéaire continue de ce programme.

4.2.2 Construction du programme SDP associé à $(IMCP)$

Pour écrire une relaxation SDP associée à $(IMCP)$, nous avons utilisé la règle $LI1$ présentée dans [82]. Roupin propose dans [82] une recette générique qui permet d'obtenir une relaxation SDP à partir d'une relaxation linéaire d'un problème. Pour cela, il part de la relaxation linéaire du problème et utilise des règles pour transformer les contraintes linéaires en contraintes quadratiques.

Rappelons ici le programme linéaire $(IMCP)$, avec $M = K$ car le graphe considéré est un arbre :

$$(IMCP) \quad \left| \begin{array}{lll} \text{Minimiser} & \sum_{e \in E} u_e \, c_e & \\ \text{Sous les contraintes} & \sum_{e \in p_i} c_e \ \geq 1 & \forall i = 1, .., M \\ & c_e \in \{0,1\} & \forall e \in E \end{array} \right. \qquad (4)$$

$PL\{0,1\}$	SDP
$X = xx^T$	$X \succeq xx^T$
PL	
$1 \geq x \geq 0$	$\begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix}$ $\mathbf{d}(X) = x$
$b \leq a^T x \leq b'$	Regle $LI1$ $aa^T \bullet X - (b+b')\, a^T x + bb' \leq 0$

TAB. 4.1 – Contraintes d'un programme linéaire et contraintes du programme SDP correspondant

La relaxation continue de $(IMCP)$, notée $(CMCP)$, est la suivante :

$$(CMCP) \quad \left| \begin{array}{lll} \text{Minimiser} & \sum_{e \in E} u_e\, c_e = u^T c & \\ \text{Sous les contraintes} & \sum_{e \in p_i} c_e \ \geq 1 & \forall i = 1, .., M \\ & c_e \geq 0 & \forall e \in E \end{array} \right. \qquad (4)$$

Le schéma que nous avons choisi d'appliquer afin d'obtenir une relaxation SDP est fourni dans le tableau 4.1.

Nous avons les contraintes SDP classiques pour les problèmes à variables bivalentes ainsi que la règle utilisée pour transformer les inégalités linéaires. Nous allons voir plus en détail la construction des inégalités.

La solution consiste à transformer les contraintes linéaires suivantes : $b \leq a^T x \leq b'$ en un type unique de contraintes dans le programme SDP en effectuant les opérations suivantes : $\left(a^T x - b\right)\left(a^T x - b'\right) \leq 0$ et ainsi obtenir les contraintes quadratiques qui en découlent : $aa^T \bullet xx^T - (b+b')\, a^T x + bb' \leq 0$. Si l'on relâche la contrainte $X = xx^T$ en $X \succeq xx^T$, nous obtenons la contrainte : $aa^T \bullet X - (b+b')\, a^T x + bb' \leq 0$. Pour le problème de la multicoupe, nous n'avons pas de borne supérieure b' pour appliquer la règle $LI1$, mais une solution est de noter que les longueurs L_{i} de chaque chaîne peuvent jouer ce rôle. Si l'on applique ce schéma à la relaxation linéaire continue $(CMCP)$, nous obtenons le programme (SDP_{MC}) suivant :

$$(SDP_{MC}) \begin{cases} \text{Min} & u^T c & \\ \text{s.c.}q. & e_i e_i^T \bullet C - (1+L_i)\, e_i^T c + L_i \leq 0 \;\; i = 1, ..., K & (8) \\ & \mathbf{d}(C) = c & [A] \\ & \begin{bmatrix} 1 & c \\ c^T & C \end{bmatrix} \succeq 0 & [B] \end{cases}$$

Les contraintes (4) : $\sum_{e \in p_i} c_e \;\geq 1 \, \forall i = 1, .., K$, peuvent se formuler de la manière suivante : $e_i^T c \geq 1 \, \forall i = 1, .., K$ avec e_i le vecteur tel que $e_{ij} = 1$ si et seulement si l'arête j est sur p_{i}. Nous obtenons ainsi, les contraintes quadratiques (8) :

$\left(e_i^T c - 1\right)\left(e_i^T c - L_i\right) \leq 0 \;\; i = 1, ..., K$, soit : $e_i e_i^T \bullet C - (1+L_i)\, e_i^T c + L_i \leq 0 \;\; i = 1, ..., K$

Rappelons que les contraintes A et B impliquent que : $\forall e$, $0 \leq c_{\text{e}} \leq 1$.

4.2.3 Comparaison des bornes de la programmation linéaire et de la SDP

Roupin a montré dans [82] que la relaxation SDP, obtenue à partir de la relaxation linéaire continue en appliquant la recette générique, est meilleure que cette relaxation linéaire continue. Rappelons la proposition 5 de [82] :

proposition : Si (X, x) est une solution admissible de $PL\,\{0,1\}$ alors la contrainte $aa^T \bullet X - (b+b')\, a^T x + bb' \leq 0$ est équivalente à $b \leq a^T x \leq b'$. Si (X, x) n'est pas une solution entière et si $aa^T \bullet X - (b+b')\, a^T x + bb' \leq 0$ et $X - xx^T \succeq 0$, alors soit $aa^T \left(X - xx^T\right) = 0$, soit $b < a^T x < b'$ ou soit $a^T x \notin [b, b']$.

On déduit de cette proposition que les contraintes (8) sont équivalentes aux contraintes (4) de $(CMCP)$ si la solution de (SDP_{MC}) est entière, c'est à dire si (X, x) est une solution admissible de $PL\,\{0,1\}$. Par contre pour toute solution admissible de PL, alors :

a) Soit $e_i e_i^T \left(C - cc^T\right) = 0$, la solution fractionnaire vérifie une contrainte supplémentaire, du programme (SDP_{MC}), qui est toujours vérifiée par les solutions entières.

b) Soit $1 < e_i^T c < L_i$.

c) Soit $e_i^T c \notin [1, L_i]$ et la solution n'est pas admissible.

La transformation des contraintes linéaires par la règle $LI1$ permet donc de rendre non admissibles pour (SDP_{MC}) des solutions fractionnaires admissibles pour le programme linéaire continu $(CMCP)$.

Nous avons résolu la relaxation SDP, (SDP_{MC}), afin de la comparer avec la relaxation linéaire continue $(CMCP)$. Nous avons construit des instances aléatoires pour $IMCP$ sur les arbres. Nous savons que ce problème est Max-SNP difficile sur les arbres (confère section

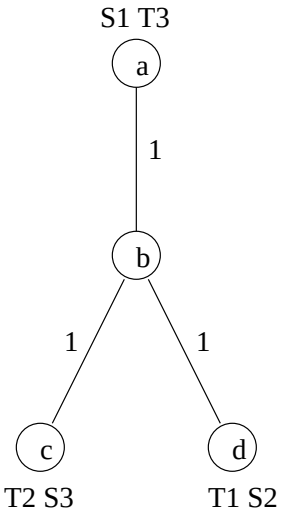


FIG. 4.2 – $IMCP$ avec $K = 3$ dans une étoile.

2.4.2.1) et le fait qu'il n'y ait qu'une seule chaîne entre une source et un puits sur un arbre permet à la formulation arêtes-chaînes d'être polynomiale en la taille du problème. Toutes les instances dont nous parlerons dans cette étude seront des instances dont la solution obtenue par la relaxation linéaire continue ($CMCP$) contient des variables fractionnaires. Nous savons que $IMCP$ sur un arbre orienté est un problème polynomial (voir section 3.1), on peut donc avoir de nombreuses instances générées aléatoirement sur les arbres dont la relaxation continue du problème donne une solution optimale entière.

Donnons tout d'abord un exemple simple d'application. Prenons une étoile à 3 arêtes de capacités unitaires et 3 couples $\{s_k, t_k\}$ comme dans la figure 4.2. En effet, comme cela a été dit dans la section 2.4.2.1, $IMCP$ sur une étoile avec des capacités unitaires est encore un problème Max SNP-difficile. Nous pouvons également remarquer qu'il est impossible d'orienter cette étoile, donc la matrice des contraintes de ($IMCP$) associée à cette instance n'est pas totalement unimodulaire.

La solution optimale continue C_c^* de ($CMCP$) est donnée par : $c^{c*}_{(a,b)} = c^{c*}_{(b,c)} = c^{c*}_{(b,d)} = \frac{1}{2} = 0.5$ et $v\left(C_c^*\right) = 1.5$.

La solution C^*_{SDP} obtenue par (SDP_{MC}) est donnée par : $c^{SDP*}_{(a,b)} = 0.66668571$, $c^{SDP*}_{(b,c)} = 0.66663330$, $c^{SDP*}_{(b,d)} = 0.66666916$ et $v\left(C^*_{SDP}\right) = 2$.

Nous remarquons que la borne fournie par la SDP est meilleure que la borne de la relaxation linéaire continue dans ce cas particulier. Nous remarquons également que les variables, dont les valeurs sont fractionnaires, n'ont pas toutes exactement les mêmes valeurs dans la solution

SDP, ceci se confirme dans tous les tests que nous avons effectués et nous permet d'avoir une base de départ pour une heuristique permettant de calculer une solution entière. En effet, nous pouvons dans ce cas mettre la variable, qui a la plus grande valeur dans la solution SDP, à 1 et faire de même jusqu'à obtenir une solution admissible.

Nous avons effectué des comparaisons similaires sur de nombreuses instances aléatoires. On remarque (voir le tableau 4.2), comme pour l'exemple précédent, que la borne inférieure fournie par la relaxation SDP est meilleure que la solution de la relaxation linéaire continue. L'amélioration de la solution est comprise entre 0.1% et 13%. La relaxation SDP fournit la solution optimale entière dans 26% des cas (Rappelons que toutes les instances étudiées sont des instances pour lesquelles la solution de la relaxation linéaire continue est fractionnaire). Par contre au niveau du temps de calcul, la SDP montre sa principale faiblesse, puisque le programme linéaire continu est par exemple résolu en moins de 1 minute pour des instances dans lesquelles le nombre d'arêtes et le nombre de flots sont supérieurs à 500, tandis que le programme SDP est résolu en plus d'une heure pour des instances de cette taille. Compte tenu du temps de calcul élevé de la SDP pour notre problème, celle-ci ne peut pas être utilisée seule dans une procédure de séparation et d'évaluation.

4.2.4 Utilisation de la SDP dans un branch-and-bound

Utiliser la SDP dans une procédure de Branch-and-Bound implique généralement un nombre de nœuds explorés dans l'arbre de recherche moins important, mais son temps de calcul étant beaucoup trop élevé, nous avons décidé de construire un branch-and-bound mixte, utilisant à la fois la programmation linéaire et une heuristique utilisant la programmation semi-définie positive pour obtenir le plus rapidement possible une solution entière optimale.

4.2.4.1 Branchement et bornes

A chaque nœud de l'arbre de recherche, nous branchons en considérant l'arête la plus dense, c'est-à-dire l'arête sur laquelle le nombre de chaînes, reliant les couples $\{s_k, t_k\}$, $k = 1, .., K$, est maximal. Pour obtenir la borne supérieure, nous résolvons en premier lieu la relaxation linéaire continue du problème à l'aide de Cplex 7.0 [51] et nous construisons ensuite notre relaxation SDP uniquement sur le sous-problème engendré par les variables fractionnaires de la solution continue. En effectuant cette réduction de l'instance du problème, nous pouvons résoudre un programme SDP, qui comprend généralement moins de 100 variables et dont la résolution prend entre quelques secondes et quelques minutes. Nous répétons ensuite cette

	Solution entière	Solution continue	Solution entière optimale
SDP	26%	74%	26%
SDP+réduction	60%	40%	50%

TAB. 4.2 – Récapitulatif des principaux résultats pour $IMCP$ dans les arbres non orientés

réduction tant que nous obtenons une solution avec des variables entières et fractionnaires en utilisant la SDP pour les résoudre. A la fin, s'il reste des variables continues, nous utilisons une heuristique pour obtenir une solution entière. Cette heuristique met toutes les variables continues restantes à 0 et trie les arêtes selon leur capacité. Ensuite tant que la solution n'est pas admissible, nous mettons à 1 la variable ayant la plus petite capacité. Si nous utilisons cette méthode sans la programmation linéaire et sans le branch-and-bound, la SDP est alors utilisée comme une heuristique et nous obtenons une solution optimale entière dans 50% des cas comme on peut le voir dans le tableau récapitulatif 4.2, où SDP signifie que l'on a utilisé uniquement la programmation semi-définie pour la résolution et SDP+réduction signifie que l'on a utilisé l'heuristique précédente. Ce tableau indique le pourcentage des instances pour lesquelles la SDP et l'heuristique précédente ont fourni une solution continue ou entière, et pour l'heuristique, dans le cas où la solution obtenue est entière, si celle-ci est optimale.

La borne inférieure étant, quant à elle, fournie par la relaxation linéaire continue calculée précédemment à chaque nœud. Nous avons effectué un parcours en profondeur d'abord de l'arbre de recherche.

4.2.4.2 Résultats numériques

Nous avons implémenté notre branch-and-bound en langage C en utilisant Cplex 7.0 pour résoudre la relaxation linéaire continue du problème. Nous avons également utilisé Cplex pour obtenir la solution optimale entière afin de comparer avec les résultats obtenus par notre méthode, dans laquelle nous avons utilisé le solveur SB ([46]) pour résoudre la relaxation SDP. La machine sur laquelle nous avons effectué nos tests est une UltraSparc Sun $400 Mhertz$. Nous avons résumé les résultats dans les tableaux 4.3 et 4.4, où :

M est le nombre d'arêtes,

K le nombre de paires s_k-t_k,

Opt et Opt* sont respectivement les valeurs de la solution optimale continue de ($CMCP$) et de la solution optimale entière de ($IMCP$),

les quatre dernières colonnes indiquent le nombre de nœuds dans l'arbre de recherche (1 si aucun branchement n'est nécessaire) et le temps en secondes selon qu'il s'agisse du branch-

M	K	Opt	Opt*	Nb ndsCplex	TempsCplex	Nb ndsPL+SDP	TempsPL+SDP
50	100	880.5	908	1	0	3	2
50	140	973.333	998	6	2	7	8
50	160	1126	1139	3	1	5	13
50	180	980.5	1008	2	1	3	11
50	220	1259.5	1420	8	3	9	20
50	260	1142	1316	7	4	8	28
100	140	1471	1488	1	0	3	10
100	140	1214	1215	1	0	3	7
100	220	1914	1922	3	3	7	39
100	260	1869	1879	1	1	3	21
200	150	1992	1992	1	1	3	16
200	300	2860.5	1881	2	3	3	32
200	400	3081	3084	1	2	3	40
200	450	3520.5	3559	4	5	7	152
200	450	3708.5	3760	6	7	7	181

TAB. 4.3 – $IMCP$: Résultats numériques 1.

and-bound de Cplex ou de notre branch-and-bound.

Comme on peut le voir sur le premier tableau de résultats, tableau 4.3, la programmation linéaire continue fournit déjà une borne de qualité pour le problème de la multicoupe sur un arbre, il s'ensuit que le nombre de nœuds explorés dans l'arbre de recherche des différents branch-and-bound est très réduit. Notons que Cplex peut parfois déterminer à la racine si sa primalisation est optimale, grâce à l'utilisation de coupes (de Gomory) dans la relaxation continue du problème. Nous constatons que notre méthode prend plus de temps que Cplex. Le fait que nous résolvions la relaxation SDP sur des problèmes réduits aux variables fractionnaires ne permet donc pas une résolution plus rapide que celle de Cplex si le graphe est un arbre. Mais le temps de calcul de notre méthode, compris entre quelques secondes et 4 minutes pour les instances difficiles, est beaucoup plus court que celui constaté lors de la résolution de la relaxation SDP sur les instances non réduites. En fait, le temps de calcul augmente (quelques minutes) quand le nombre de variables fractionnaires dans la solution continue est important (supérieur à 100). Comme on peut le voir dans le second tableau de résultats numériques, tableau 4.4, constitué d'instances beaucoup plus grandes, le temps de calcul ainsi que le nombre de nœuds explorés restent comparables à ceux des instances plus petites.

Cplex reste donc plus efficace que notre méthode même sur de grandes instances. Mais si nous comparons notre méthode à notre branch-and-bound sans utiliser la SDP, mais unique-

M	K	Opt	Opt*	Nb ndsCplex	TempsCplex	Nb ndsPL+SDP	TempsPL+SDP
300	200	2621.5	2626	1	4	3	15
300	250	2517	2523	2	5	3	40
300	300	4161.5	4163	2	6	3	20
300	450	4444	4450	3	7	5	139
400	250	3221	3241	1	4	3	30
400	250	3843	3845	1	4	3	29
400	300	4281.5	4296	2	9	5	115
400	350	4109	4129	1	6	3	41
500	600	6698	6717	1	12	3	42
500	700	6528	6531	1	15	3	51
500	700	7267	7269	1	16	3	50
1000	1000	12970.5	12978	1	39	3	110
1500	800	15485.5	15489	2	104	3	155
2000	1000	16653	16654	1	75	3	220

TAB. 4.4 – $IMCP$: Résultats numériques 2.

ment en utilisant la programmation linéaire, nous obtenons ainsi la borne supérieure uniquement par l'intermédiaire de notre heuristique, alors notre méthode est meilleure pour pratiquement toutes les instances. On peut donc en déduire l'importance des coupes générées par Cplex lors du calcul de la borne inférieure.

4.2.4.3 Améliorations

Afin d'améliorer notre méthode, nous pouvons décider de résoudre le programme SDP sur le problème non réduit uniquement à la racine, nous obtenons ainsi une valeur qui est très proche de l'optimum, ou égale à l'optimum, et nous construisons donc notre branch-and-bound sans utiliser de nouveau la SDP. Ainsi, même si le temps de calcul augmente à la racine, le temps de calcul global peut s'en trouver diminué car le nombre de nœuds explorés devrait diminuer tout comme le temps de calcul en ces nœuds qui devrait lui aussi décroître. Cette approche semble donc intéressante pour résoudre des instances difficiles, dont la solution n'est trouvée qu'après un grand nombre de nœuds dans l'arbre de recherche du branch-and-bound.

Une autre amélioration peut consister à recalculer la borne supérieure L_{i} que nous avons ajoutée aux contraintes (4) de $(CMCP)$, nous pouvons le faire aisément après chaque résolution d'un programme SDP car nous connaissons les valeurs des variables et donc nous pouvons facilement calculer le nombre maximum d'arêtes coupées sur chaque chaîne, ce qui

nous donnera un nouveau L_{i} pour chaque chaîne.

Les performances de Cplex nous incitent également à générer des coupes pour améliorer la borne inférieure obtenue à chaque nœud. En effet, il suffit en général à Cplex de générer un très petit nombre de coupes (de Gomory) pour obtenir une borne inférieure de très bonne qualité, lui permettant le plus souvent d'obtenir l'optimum entier à la racine de l'arbre de recherche. Il serait donc intéressant pour nous également de générer ces coupes de Gomory et une étude plus complète nous permettrait sans doute d'obtenir des coupes spécifiques au problème de multicoupe, ce qui nous permettrait d'obtenir une borne inférieure de meilleure qualité et ainsi de construire un Branch and Bound efficace.

4.3 Résolution de $IMFP$

Nous étudions dans cette section les méthodes que l'on peut appliquer pour résoudre $IMFP$. Comme dans la section précédente, un algorithme de séparation et d'évaluation basé sur la programmation linéaire semble l'approche la plus logique. Nous donnons également quelques pistes pour appliquer la programmation semi-définie positive (SDP) au problème de multiflot maximal en nombres entiers.

4.3.1 Programmation linéaire en nombres entiers

$IMFP$ est un problème linéaire en nombres entiers, l'utilisation d'un Branch and Bound utilisant la programmation linéaire semble être la solution la plus évidente de résolution. Si nous utilisons Cplex 7.0 pour résoudre ($IMFP$) sur certaines des instances difficiles, dans les arbres non orientés, qui ont été générées pour résoudre $IMCP$, alors nous obtenons les résultats du tableau 4.5, où M est le nombre d'arêtes, K le nombre de paires s_{k}-t_{k}, Opt et Opt* sont respectivement les valeurs de la solution optimale continue de ($CMFP$) et de la solution optimale entière de ($IMFP$). Les deux dernières colonnes indiquent le nombre de nœuds dans l'arbre de recherche (1 si aucun branchement n'est nécessaire) et le temps en secondes du branch-and-bound de Cplex.

Comme pour $IMCP$, on remarque que Cplex est très performant pour résoudre $IMFP$ dans les arbres non orientés. De plus le saut d'intégrité entre la solution continue et la solution entière est très petit et semble même être inférieur par rapport à celui de $IMCP$. Cplex reste donc très efficace pour ce problème et comme précédemment, par l'ajout de coupes, il arrive à décider à la racine de l'arbre de recherche s'il est à l'optimum. Il serait donc encore intéressant d'avoir une approche polyédrale pour résoudre ($IMFP$) afin de générer les coupes de Gomory

M	K	Opt	Opt*	Nb ndsCplex	TempsCplex
50	100	880.5	880.5	1	1
50	140	973.33	973	1	2
100	220	1914	1914	1	3
200	450	3520.5	3520	3	13
300	200	2621.5	2621	1	2
300	450	4444	4444	1	6
1000	1000	12970.5	12970	1	23
1500	800	15485.5	15485	1	7

TAB. 4.5 – $IMFP$: Résultats numériques.

et des coupes spécifiques au problème de multiflot pour les intégrer dans un Branch and Bound.

4.3.2 Programmation semi-définie

Comme nous l'avons précisé à de nombreuses reprises, $IMFP$ consiste à trouver un multiflot maximal en nombres entiers. Or, l'élaboration d'une relaxation SDP pour les problèmes en nombres entiers reste un domaine peu exploré. Une solution envisageable permettant d'appliquer la SDP au problème de multiflot est de décomposer les flots. En effet, un flot peut prendre des valeurs qui sont uniquement comprises entre 0 et la capacité minimale sur la chaîne sur laquelle il doit être routé. Soit R_k^{min} la capacité minimale sur la chaîne p_k, le flot F_k peut alors se décomposer en $log\, R_k^{min}$ "flots", et soit $R^{min} = max\left(R_k^{min}\right),\ k = 1,..,K$, $IMFP$ avec K flots à router devient un problème contenant un nombre de variables en $O\left(K\, log\, R^{min}\right)$ dans un arbre non orienté. La taille du problème SDP à résoudre peut ainsi devenir très grande si R^{min} est grand et qu'aucune homotétie des capacités n'est possible.

Chapitre 5

Résolution des problèmes dans les anneaux

Étudions maintenant le cas des réseaux en anneau. Un anneau est un graphe connexe dans lequel tous les sommets sont de degré 2 et $|V| = |E|$, c'est-à-dire que $n = m$.

De nombreuses applications dans les réseaux de télécommunication utilisent cette structure d'anneau ([15], [61]). On peut notamment citer les SONET (Synchronous Optical Network) dans lesquels les équipements de transmission en fibres optiques sont configurés en anneaux. Il peut être intéressant de considérer des anneaux orientés ou non orientés en fonction du problème que l'on souhaite résoudre.

Avant de résoudre les problèmes de coupe et de flot, nous pouvons effectuer quelques simplifications dans l'anneau.

5.1 Simplifications

Notons tout d'abord qu'il est facile de transformer une instance non orientée d'un problème de multiflot ou d'un problème de multicoupe en une instance orientée. En effet, chaque flot, correspondant à un couple $\{s_k, t_k\}$, ne peut passer que par deux chemins, soit dans le sens des aiguilles d'une montre, soit dans le sens inverse des aiguilles d'une montre. On choisit une orientation, par exemple dans le sens des aiguilles d'une montre et à chaque couple $\{s_k, t_k\}$ on associe deux couples : $\{s_k, t_k\}$ dont le flot F_k correspondant va de s_k à t_k et $\{s_{k+K}, t_{k+K}\}$ dont le flot F_{k+K} correspondant va de t_k à s_k de manière à respecter l'orientation, un exemple est donné dans la figure 5.1.

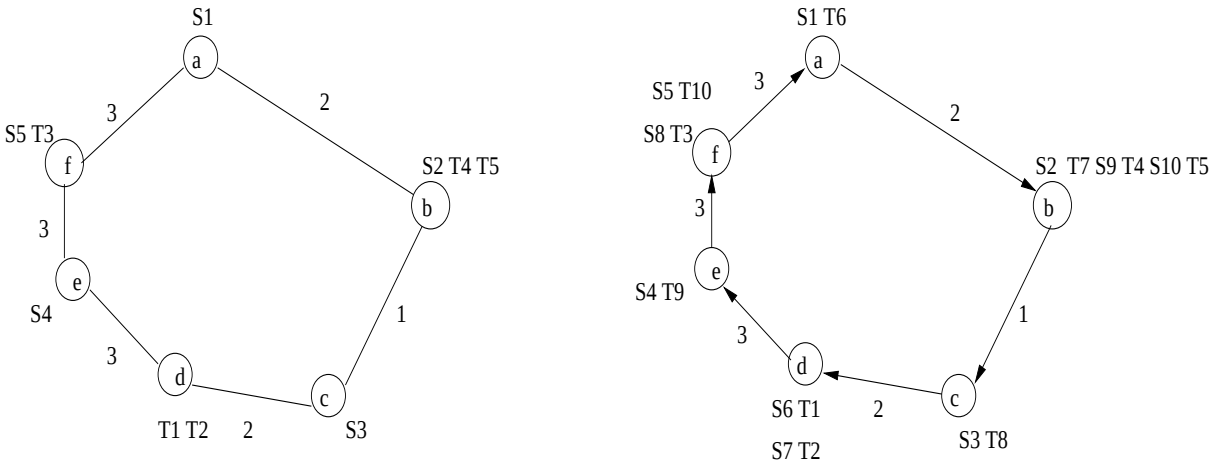


FIG. 5.1 – Orientation d'un anneau.

Nous pouvons désormais considérer des anneaux orientés, car résoudre les problèmes de multiflot et de multicoupe avec K couples à router ou à séparer dans un anneau non orienté équivaut à résoudre ces mêmes problèmes avec $2K$ couples à router ou à séparer dans un anneau orienté.

Nous pouvons ensuite effectuer quelques simplifications dans l'anneau orienté obtenu. Si un chemin p_k, d'une source s_k à un puits t_k est inclus dans un autre chemin p_l, d'une source s_l à un puits t_l, alors nous pouvons supprimer le couple $\{s_l, t_l\}$ de l'anneau. En effet, il est toujours plus intéressant de router le flot F_k que le flot F_l car tout ce qui peut circuler sur p_l peut circuler sur p_k et le nombre d'arcs de p_k est inférieur à celui de p_l, de plus si le chemin p_k est coupé, alors le chemin p_l le sera également, donc on peut ne garder que le couple $\{s_k, t_k\}$. Un exemple de cette simplification est donné sur la figure 5.2 : le couple $\{s_4, t_4\}$ est supprimé car le chemin p_4 comprend le chemin p_5 et le couple $\{s_1, t_1\}$ est également supprimé car le chemin p_1 comprend le chemin p_2.

Une autre simplification du problème consiste à réduire n'importe quelle suite d'arcs (ou partie de chemin) qui ne comporte ni source ni puits en un seul arc dont le poids sera égal au poids minimal des arcs de ce chemin. En effet, pour couper un tel chemin, l'arc de poids minimal est toujours meilleur que les autres ; de plus, cet arc de poids minimal est celui qui limite tout flot routé sur ce chemin. Les autres arcs de ce chemin ne jouent donc aucun rôle que ce soit dans le flot maximal ou dans la coupe minimale : ils peuvent être retirés du graphe. Dans la figure 5.2, on supprime le couple $\{s_4, t_4\}$ de l'anneau, on peut donc également supprimer l'arc (d, e) de l'anneau car seule la source s_4 était affectée au sommet e.

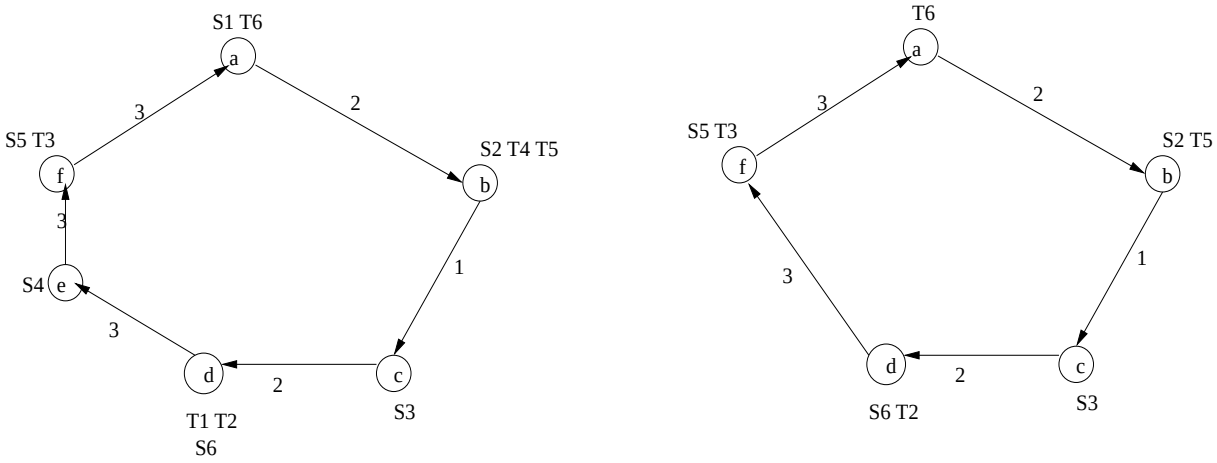


FIG. 5.2 – Simplification dans un anneau.

Nous pouvons également supprimer les arcs sur lesquels ne passe aucun flot.

Dans la suite nous considérerons des anneaux sur lesquels toutes les simplifications possibles ont été faites.

Nous allons proposer une méthode de résolution pour les problèmes de FLOT MULTITERM et de COUPE MULTITERM dans les anneaux puis nous proposons un algorithme polynomial permettant de résoudre $IMCP$ sur un anneau.

5.2 Polynomialité des problèmes de coupe et de flot multiterminaux

Les problèmes de FLOT MULTITERM et de COUPE MULTITERM sont des cas particuliers des problèmes de multiflot et de multicoupe, on peut donc également ne considérer que des anneaux orientés. Après avoir effectué les simplifications de la section précédente, il ne reste plus qu'un arc entre deux terminaux.

Pour résoudre le problème du FLOT MULTITERM, il suffit de router le maximum de flots entre deux terminaux successifs et de répéter cette opération pour tous les couples de terminaux successifs. Pour le problème de COUPE MULTITERM, de la même manière, il suffit de couper l'arc compris entre deux terminaux successifs et de répéter cette coupe pour tous les couples de terminaux successifs. Cela revient donc à couper tous les arcs de l'anneau orienté obtenu après simplifications.

Les problèmes de FLOT MULTITERM et de COUPE MULTITERM sont donc polynomiaux

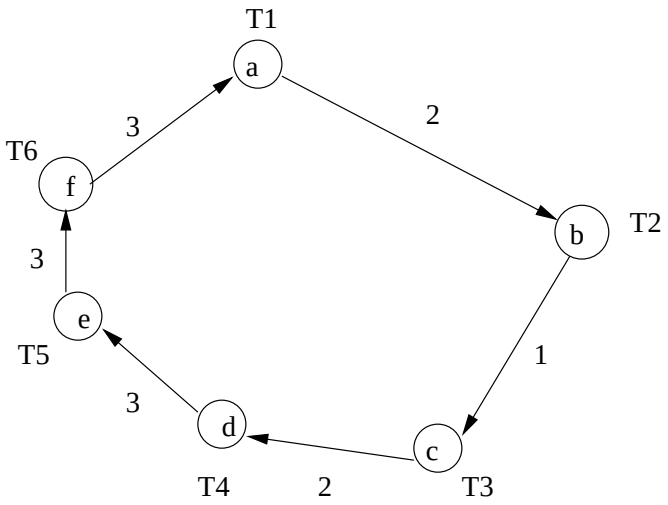


FIG. 5.3 – Instance pour COUPE MULTITERM et FLOT MULTITERM sur un anneau orienté.

dans les anneaux et peuvent être résolus en $O(n)$.

La figure 5.3 donne un exemple d'instance sur laquelle on peut appliquer cette méthode.

Si nous appliquons la méthode proposée, nous obtenons les solutions suivantes :

Pour le FLOT MULTITERM, nous devons router le flot maximal entre tous les terminaux successifs, nous obtenons donc :

$v(F_{T_1T_2}) = 2$, $v(F_{T_2T_3}) = 1$, $v(F_{T_3T_4}) = 2$, $v(F_{T_4T_5}) = 3$, $v(F_{T_5T_6}) = 3$, $v(F_{T_6T_1}) = 3$ et $v(F*) = 14$.

Pour la COUPE MULTITERM, nous devons couper tous les arcs :

$C* = \{(a,b), (b,c), (c,d), (d,e), (e,f), (f,a)\}$ et $v(C*) = 14$.

Cette solution est reportée sur la figure 5.4.

5.3 $IMCP$ dans les anneaux

En ce qui concerne $IMCP$ dans les réseaux en anneaux, nous proposons un algorithme polynomial permettant de résoudre ce problème, cet algorithme est fondé sur l'algorithme gourmand défini pour les problèmes de multiflot et de multicoupe dans les arborescences.

5.3.1 Introduction

L'idée de l'algorithme est la suivante. Considérons l'un des chemins p_k, de s_k à t_k, choisi, pour l'instant, au hasard, et noté p^*. Notons R le nombre d'arcs de p^* : $p^* = (e_1, e_2, .., e_R)$.

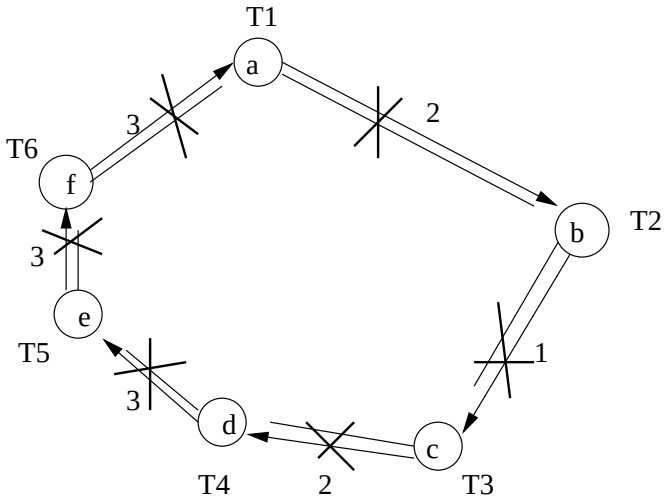


FIG. 5.4 – Solution pour COUPE MULTITERM et FLOT MULTITERM sur un anneau orienté.

Toute coupe doit contenir au moins un arc de p^*. Nous allons faire une énumération en calculant pour chaque arc e_j, $j = 1, .., R$, la multicoupe minimale qui contient cet arc et qui est notée C_j^*. Nous obtiendrons ensuite facilement la multicoupe minimale C^*, qui est la multicoupe C_j^* de valeur minimale : $C^* = argmin_{j=1,..,R}\left(C_j^*\right)$.

5.3.2 Calcul d'une multicoupe C_j^*

Si l'arc e_j appartient à la coupe, on peut supprimer cet arc et l'anneau devient une chaîne orientée. La figure 5.5 montre comment on peut décomposer un anneau orienté en chaînes orientées. Un couple $\{s_k, t_k\}$ appartient à une chaîne si et seulement si la source s_k et le puits t_k appartiennent tous les deux à cette chaîne. Il est donc inutile de conserver les sources et les puits dont les chemins correspondants ont été coupés par l'arc e_j.

Nous pouvons appliquer la simplification suivante : si la racine de la chaîne orientée ne comporte pas de source, alors nous pouvons supprimer l'arc menant de la racine au sommet suivant, de même, si la feuille de la chaîne orientée ne comporte pas de puits, nous pouvons supprimer l'arc menant du sommet précédent à la feuille et cela récursivement.

Nous devons résoudre un problème de multicoupe minimale sur une chaîne orientée. Une chaîne orientée est un cas particulier d'une arborescence et nous pouvons appliquer l'algorithme présenté dans la section 3.2 pour trouver un multiflot maximal $F_{j_1}^*$ puis une multicoupe minimale $C_{j_1}^*$ sur cette chaîne.

La multicoupe C_j^* sur l'anneau est obtenue de la manière suivante : $C_j^* = C_{j_1}^* \bigcup \{e_j\}$ et

$v\left(C_j^*\right) = v\left(C_{j_1}^*\right) + u_{e_j}$, avec u_{e_j} la capacité de l'arc e_j.

5.3.3 Multicoupe minimale sur l'anneau

Pour chaque arc e_j, $j = 1, .., R$, nous calculons la multicoupe C_j^* associée, comme indiqué dans le paragraphe précédent.

Une solution optimale C^* pour la multicoupe dans l'anneau d'origine sera alors obtenue par la solution C_j^*, $j = 1, .., R$, dont la valeur $v\left(C_j^*\right)$ est minimale. Cet algorithme est présenté dans la procédure *MulticoupeAnneau*.

Algorithm 4 $IMCP$ dans un anneau.

procédure $MulticoupeAnneau$;
entrée : $G = (V, E)$ un anneau. $(s_k, t_k) \in V^2$ $\forall k \in \{1, .., K\}$.
sortie : C^* *une multicoupe minimale.*
1. On sélectionne un chemin p^* ;
   $R \leftarrow$nombre d'arcs de p^* ;
2. $C^* \leftarrow \emptyset$;
3. **pour** $j = 1$ à R **faire**
   //on transforme l'anneau en chaînes
      3.1. $V_{j_1} \leftarrow V$; $E_{j_1} \leftarrow E - \{e_j\}$; $G_{j_1} = (V_{j_1}, E_{j_1})$;
   //on trouve un multiflot maximal et une multicoupe minimale sur ces chaînes
      3.2. $F_{j_1}^* \leftarrow Maxmultiflot\left(G_{j_1}\right)$;
         $C_{j_1}^* \leftarrow Minmulticoupe\left(G_{j_1}, F_{j_1}^*\right)$;
      3.3. $C_j^* \leftarrow C_{j_1}^* \cup \{e_j\}$; $v\left(C_j^*\right) = v\left(C_{j_1}^*\right) + u_{e_j}$;
   fin faire ;
4. $C^* = C_{min}^*$ tel que $v\left(C_{min}^*\right) = min_{j=1,..,R} v\left(C_j^*\right)$;
fin $MulticoupeAnneau$;

5.3.4 Exemple

Appliquons la procédure *MulticoupeAnneau* sur l'exemple de la figure 5.5, le chemin qui a été choisi est celui reliant s_2 à t_2. Celui-ci comporte 2 arcs : (b,c) et (c,d), on peut donc énumérer sur ces 2 arcs en supprimant soit l'arc (b,c), soit l'arc (c,d) de l'anneau. Quand l'arc (b,c) est supprimé, le couple $\{s_2, t_2\}$ ne peut figurer sur la chaîne obtenue car il n'y a plus de chemin reliant s_2 à t_2, et quand l'arc (c,d) est supprimé, les couples $\{s_2, t_2\}$ et $\{s_3, t_3\}$ ne figurent pas sur la chaîne obtenue pour les mêmes raisons.

Si on applique $Maxmultiflot$ et *Minmulticoupe* sur les 2 chaînes précédentes, on obtient le résultat présenté sur la figure 5.6.

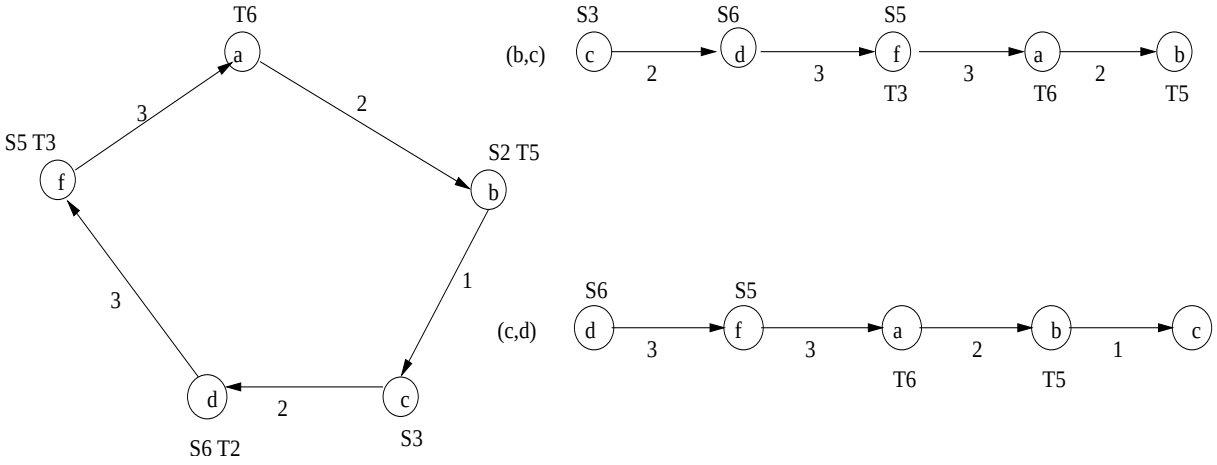


FIG. 5.5 – Décomposition d'un anneau en 2 chaînes.

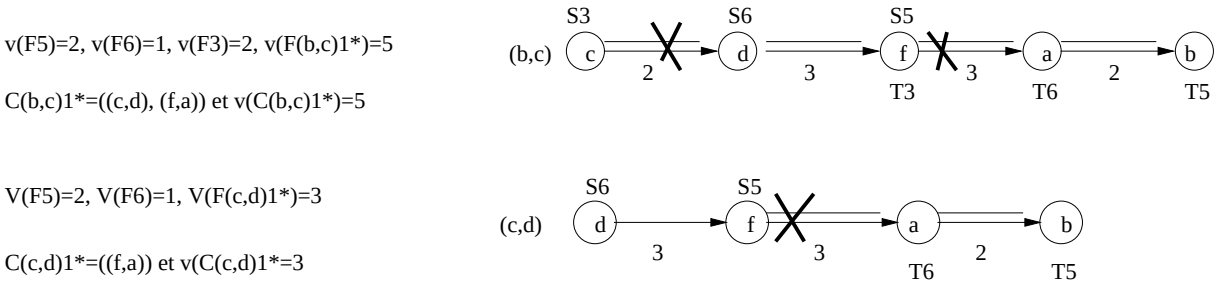


FIG. 5.6 – Résolution de $IMFP$ et $IMCP$ sur les 2 chaînes.

Si l'arc supprimé e_j est l'arc (b,c), alors on obtient $v\left(F^*_{(b,c)_1}\right) = v\left(C^*_{(b,c)_1}\right) = 5$, tandis que s'il s'agit de (c,d), alors on obtient $v\left(F^*_{(c,d)_1}\right) = v\left(C^*_{(c,d)_1}\right) = 3$.

Nous obtenons donc les solutions suivantes : $v\left(C^*_{(b,c)}\right) = v\left(C^*_{(b,c)_1}\right) + u_{(b,c)} = 5 + 1 = 6$ et $v\left(C^*_{(c,d)}\right) = v\left(C^*_{(c,d)_1}\right) + u_{(c,d)} = 3 + 2 = 5$.

La solution optimale de notre exemple est donc la seconde solution : $C^* = ((c,d),\ (f,a))$ et $v(C^*) = 5$.

5.3.5 Optimalité de l'algorithme

Les ensembles C^*_j, $\forall j = 1, .., R$ obtenus à la fin de l'algorithme sont bien des multicoupes car en appliquant la procédure *Minmulticoupe* sur la chaîne G_{j_1}, nous savons que $C^*_{j_1}$ est une multicoupe optimale sur G_{j_1} (voir le paragraphe 3.2.4), de plus nous savons par construction que e_j coupe tous les chemins qui ne sont pas sur G_{j_1}, ainsi C^*_j coupe tous les chemins de l'anneau et C^*_j est la multicoupe minimale qui contient e_j.

Parmi toutes les multicoupes obtenues, nous prenons celle de poids total minimal, la multicoupe C^* est donc bien une multicoupe minimale sur l'anneau.

5.3.6 Complexité de l'algorithme

La complexité de $IMCP$ et de $IMFP$ sur une arborescence, et donc sur une chaîne orientée, est en $O\left(min\left(Kn,\ n^2\right)\right)$. De plus, on effectue la boucle de la procédure *MulticoupeAnneau* pour tous les arcs d'un chemin ($R \leq n$). La complexité de l'algorithme est donc au plus en $O\left(n\, min\left(Kn,\ n^2\right)\right)$, i.e. $O\left(min\left(Kn^2,\ n^3\right)\right)$. En effet, bien que l'on applique $Maxmultiflot$ et *Minmulticoupe* sur des chaînes orientées, et non sur des arborescences, on ne peut réduire la complexité des procédures $Maxmultiflot$ et *Minmulticoupe*.

Montrons sur un exemple que la procédure *MulticoupeAnneau* est bien en $O\left(min\left(Kn^2,\ n^3\right)\right)$:

Pour diminuer le nombre de passages dans la boucle de la procédure *MulticoupeAnneau*, nous pouvons choisir, à l'étape 1. de l'algorithme, le chemin de longueur minimale de l'anneau. L'anneau de la figure 5.7 comporte $K = n$ flots, avec n pair, et chaque chemin, de s_k à t_k, est de longueur $\frac{n}{2}$. Router chaque flot et chercher l'arc saturé le plus haut sur un chemin peuvent s'effectuer en $\frac{n}{2}$ pas. Le chemin de longueur minimale a pour longueur $\frac{n}{2}$. Supprimer un arc d'un chemin pour obtenir une chaîne supprime également $\frac{n}{2}$ chemins, il reste donc au maximum $\frac{n}{2}$ flots sur chacune des chaînes G_{j_1}, $\forall j = 1, .., \frac{n}{2}$. On effectue une fois $Maxmultiflot$ et *Minmulticoupe* sur ces chaînes, on en déduit donc que la procédure *MulticoupeAnneau* s'effectue donc en $O\left(\frac{n}{2} \times 2 \times \frac{n}{2} \times \frac{n}{2}\right)$ soit en $O\left(n^3\right)$.

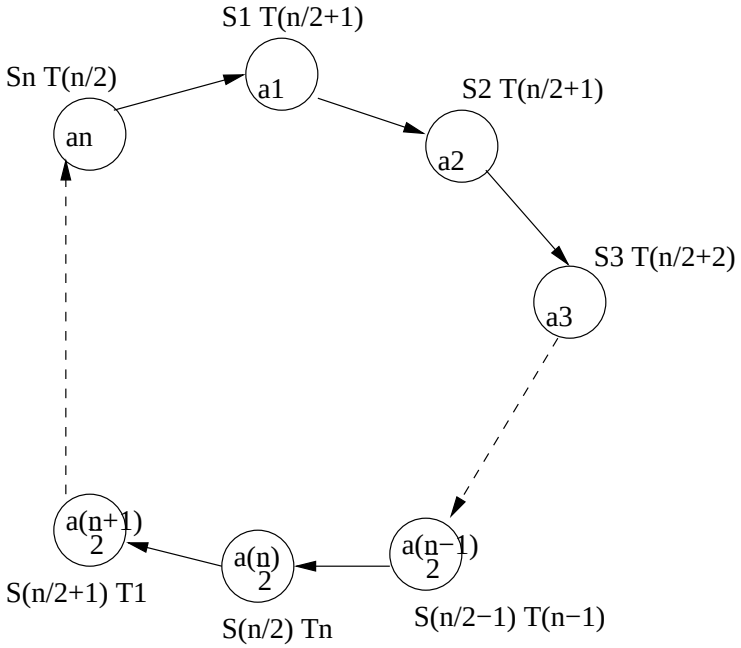


FIG. 5.7 – Instance pour laquelle $IMCP$ se résout en $O\left(n^3\right)$.

5.4 $IMFP$ dans les anneaux

Nous montrons sur l'exemple de la figure 5.8 qu'il peut exister un saut de dualité entre la valeur de la multicoupe minimale et la valeur du multiflot maximal. En effet, il est facile de vérifier que la solution optimale de $(IMCP)$ est égale à 10, en coupant deux arcs, et que la solution optimale de $(IMFP)$ est égale à 7, en routant par exemple 2 unités de flot F_1 de s_1 vers t_1, 2 unités de flot F_2 de s_2 vers t_2 et 3 unités de flot F_3 de s_3 vers t_3. Nous pouvons également remarquer que les solutions optimales continues de $(CMFP)$ et $(CMCP)$ ont pour valeur 7.5.

Nous n'avons pas encore trouvé la complexité de $IMFP$ dans les anneaux, mais il existe différentes pistes de recherche intéressantes que nous poursuivons à l'heure actuelle. Nous espérons ainsi pouvoir résoudre ce problème avec un algorithme polynomial, au moins dans certains cas particuliers.

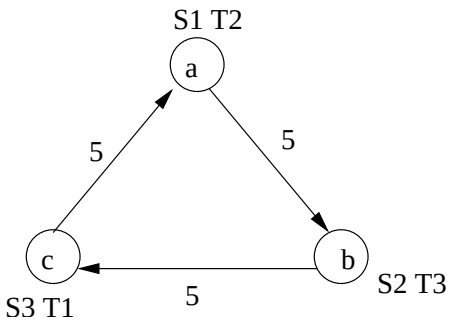


FIG. 5.8 – Instance pour $IMFP$ et $IMCP$ dans un anneau orienté.

Chapitre 6

Conclusion

Nous dressons dans cette conclusion une synthèse générale des travaux présentés, en tenant compte de tous les résultats antérieurs, et nous donnons également quelques directions de recherche.

Nous avons étudié dans cette thèse les problèmes de multicoupe minimale et de multiflot maximal en nombres entiers, ainsi que plusieurs problèmes connexes. Nous avons tenté de fournir un panorama, le plus large possible, des résultats de complexité et des méthodes de résolution qui ont été proposés jusqu'à aujourd'hui pour l'ensemble de ces problèmes. La première remarque que l'on peut faire concerne la difficulté de ces problèmes. L'ensemble des problèmes présentés dans cette thèse sont NP-difficiles dans des graphes quelconques. La plupart des problèmes sont difficiles à approximer et il faut le plus souvent considérer des graphes particuliers ou fixer le nombre de flots pour obtenir des problèmes plus faciles, c'est-à-dire des problèmes que l'on sait résoudre, soit de manière approchée, soit de manière optimale, par un algorithme polynomial.

Il est intéressant de noter que les résultats de complexité et d'approximation, présentés pour les problèmes de multiflot, de multicoupe et pour les problèmes connexes, diffèrent en fonction de l'orientation ou de la non orientation du graphe considéré, mais pas toujours de la même manière. En effet, dans certains cas, les problèmes orientés sont plus faciles à résoudre : par exemple $IMFP$ et $IMCP$ sont polynomiaux dans les arbres orientés mais Max SNP-difficiles dans les arbres non orientés. Dans d'autres cas les problèmes orientés sont plus difficiles : approximer le problème de la coupe multiterminaux dans un graphe orienté semble plus difficile que dans un graphe non orienté ; de la même manière, le problème de la maximisation du nombre de chemins disjoints par les arêtes est NP-difficile pour $K = 2$ dans

les graphes orientés mais est polynomial pour tout K fixé dans les graphes non orientés.

Compte tenu du fait que $IMFP$ et $IMCP$ sont Max SNP-difficiles dans les arbres non orientés, nous avons cherché à connaître la complexité de ces problèmes dans les arbres orientés. Nous avons ainsi pu montrer que $IMFP$ et $IMCP$ sont polynomiaux dans les arbres orientés. Nous avons montré que l'on peut adapter un algorithme de circulation de flot de coût minimal à $IMFP$, ce qui permet d'avoir un algorithme polynomial en $O\left(max\left(K^2 log\, n,\, n^2 log^2 n\right)\right)$ permettant de résoudre $IMFP$ dans les arbres orientés de manière exacte. Nous avons ensuite proposé un algorithme gourmand polynomial en $O\left(min\left(Kn,\, n^2\right)\right)$ pour résoudre $IMFP$ et $IMCP$ dans les arborescences. Il est fondé sur la relation de dualité qui lie ces deux problèmes.

Nous avons proposé un algorithme polynomial en $O(m)$ permettant de savoir si une instance de $IMFP$ ou de $IMCP$ dans un arbre non orienté pouvait se réduire à une instance dans un arbre orienté que l'on sait résoudre en temps polynomial. Nous avons ensuite proposé un algorithme de branch-and-bound utilisant à la fois la programmation semi-définie et la programmation linéaire pour résoudre de manière exacte $IMCP$ dans les arbres non orientés.

Nous avons également proposé des algorithmes polynomiaux, en $O(n)$ pour les problèmes de coupe et de flot multiterminaux, et en $O\left(min\left(Kn^2,\, n^3\right)\right)$ pour $IMCP$, dans les réseaux en anneau. Nous avons montré qu'il peut exister un saut de dualité entre la valeur d'une solution du multiflot et la valeur d'une solution de la multicoupe dans les anneaux.

L'ensemble des résultats significatifs de complexité et d'approximation concernant tous les problèmes étudiés est récapitulé dans le tableau 6.1, qui classe ces résultats selon le type de problème et le type de graphe considérés.

Les résultats présentés dans cette thèse ouvrent de nombreuses perspectives de recherche. Nous envisageons en premier lieu d'adapter l'utilisation de la programmation semi-définie aux problèmes en nombres entiers, ce qui nous permettrait d'avoir une nouvelle approche de résolution pour $IMFP$, que l'on pourrait intégrer dans un branch-and-bound. Nous envisageons également d'adapter la méthode de résolution utilisée pour résoudre $IMCP$ dans les arbres de manière à pouvoir résoudre ces problèmes, ainsi que les problèmes de multiflot, dans les graphes quelconques. Cette adaptation requiert une réflexion sur la formulation à utiliser. En effet, dans un graphe quelconque le nombre de chaînes entre une source et le puits associé devient exponentiel. Il serait alors peut-être préférable d'utiliser une formulation dont le nombre de variables est polynomial en la taille du problème. Dans ce cas il faudra établir de nouvelles relaxations semi-définies. Il serait également intéressant d'intégrer les coupes de Gomory et de trouver d'autres coupes valides pour ces problèmes afin de les résoudre plus efficacement. De plus, il reste encore de nombreuses questions ouvertes, comme par exemple

connaître la complexité du problème du biflot dans les graphes bipartis orientés augmentés (cf Section 2.4.3), ou savoir s'il est possible d'approximer $IMCP$ avec un ratio constant dans les graphes non orientés, ou encore connaître la complexité de $IMFP$ dans les anneaux. Il faudrait donc se pencher sur ces problèmes de complexité. Il serait également intéressant d'obtenir des algorithmes permettant de résoudre des problèmes dont on sait uniquement qu'ils sont polynomiaux, comme par exemple $IMCP$ dans les arbres orientés.

	IMFP	IMCP	FlotInSep	CapChem	ChemDisjAre	Coupe Multiterm	Flot Multiterm
Graphes non orientés	Max SNP-diff. [35] NP-diff. appro. avec $m^{\frac{1}{2}-\varepsilon}$[42] Algo. O(logK)-appro. [34]	Max SNP-diff [23] Algo. O(logK)-appro. [34]	NP-diff. [70] Algo. $O(\sqrt{m})$ – appro. [6]	NP-diff. [70] Algo. $O(\sqrt{m})$ – appro. [6]	NP-diff. [70] Algo. $O(\sqrt{m})$ – appro. [6] Polyn. pour K fixé [80]	Max SNP-diff. [23] Algo. 1.3438-appro. [53]	
Graphes orientés	Max SNP-diff. [35] NP-diff appro. avec $m^{\frac{1}{2}-\varepsilon}$[42]	Max SNP-diff. [23] $C^* \leq 108F^{*3}$ et $C^* \leq 39ln(K+1)F^{*2}$ [12]	NP-diff. [29] Algo. $O(\sqrt{m})$ – appro. [6] NP-diff. appro. avec $m^{\frac{1}{2}-\varepsilon}$[42]	NP-diff. [29] Algo. $O(\sqrt{m})$ – appro. [6] NP-diff. appro. avec $m^{\frac{1}{2}-\varepsilon}$[42]	NP-diff. [29] Algo. $O(\sqrt{m})$ – appro. [6] NP-diff. appro. avec $m^{\frac{1}{2}-\varepsilon}$[42]	Max SNP-diff. [23] Algo. 2logK-appro. [33] Polyn. si sans circuit $O(n^3)$ [18]	Polyn. si sans circuit $O(n^3)$ [18]
Arbres orientés	**Polyn.** $O(max(K^2logn, n^2log^2n)$ [16]	**Polyn.** [16]	**Polyn.** [16]	**Polyn.** [16]	**Polyn.** [16]	Polyn. [18]	Polyn. [18]
Arbores.	**Polyn.** $O(min(Kn, n^2))$ [16]	**Polyn.** $O(min(Kn, n^2))$ [16]	**Polyn.** $O(min(Kn, n^2))$ [16]	**Polyn.** $O(min(Kn, n^2))$ [16]	**Polyn.** $O(min(Kn, n^2))$ [16]	Polyn. O(n) [17]	Polyn. O(Kh) [17]
Arbres bi-orientés	Max SNP-diff. [25]		Max SNP-diff. [25]	Max SNP-diff. [25]	Max SNP-diff. Algo. $(5/3+\varepsilon)$-appro. [25]		
Arbres non orientés	Max SNP-diff. Algo. 1/2-appro. [35]	Max SNP-diff. Algo. 2-appro. [35]	Max SNP-diff. Algo. 1/2-appro. [35]		Polyn. [35]	Polyn. O(n) [17]	Polyn. $O(n^2)$ [17]
Graphes bipartis	NP-diff. pour K $\geq$ 3 [18]					Polyn. si orientés [18]	Polyn. si orientés [18]
Graphes planaires	NP-diff. [70]	NP-diff. [23] Algo. appro. facteur constant [90]	NP-diff. [70]	NP-diff. [70]	NP-diff. [70]	NP-diff. [23]	
Anneaux	Polyn. $O(n^3)$avec demandes et capacités uniformes [61]	**Polyn.** $O(min(Kn^2, n^3)$	NP-diff avec demandes [15]		Polyn. [30]	**Polyn.** $O(n)$	**Polyn.** $O(n)$

TAB. 6.1 – Principaux résultats de complexité et d'approximation.

Glossaire des problèmes

$IMFP$: Problème du multiflot maximal en nombres entiers. Il consiste à maximiser la somme des valeurs des flots entiers routés de chaque source à chaque puits correspondant, en respectant les contraintes de capacité et de conservation de flot.

$IMCP$: Problème de la multicoupe minimale. Il consiste à trouver un ensemble d'arêtes de poids minimal dont le retrait coupe chaque chaîne menant de chaque source à chaque puits correspondant.

Coupe Multiterm : Problème de la coupe multiterminaux. Il consiste à trouver un ensemble d'arêtes de poids minimal dont le retrait sépare deux à deux les sommets appartenant à un ensemble donné.

Flot Multiterm : Problème de flot multiterminaux. Il consiste à maximiser la somme totale des flots routés entre toutes les paires de sommets d'un ensemble donné.

K-Coupe : Problème de la K-coupe. Il consiste à partitionner les n sommets du graphe en K paquets non vides de manière à minimiser la somme des poids des arêtes permettant ce partionnement des sommets.

FlotInSep Max : Problème des flots inséparables maximaux. Il consiste à router chaque flot, vérifiant les contraintes de capacité, sur une unique chaîne de manière à maximiser la somme des flots routés.

CapChem Max : Problème des multichemins maximaux sous contraintes de capacité. Il consiste à maximiser le nombre total de chaînes reliant les paires de sommets sachant que le nombre total de chaînes pouvant passer sur chaque arête ne peut excéder la capacité de celle-ci.

ChemDisjAre Max : Problème de la maximisation du nombre de chemins disjoints par les arêtes. Il consiste à maximiser le nombre total de chaînes reliant les paires de sommets de manière à ce que chaque arête n'appartienne qu'à une seule chaîne.

Bibliographie

[1] R. Ahuja, T. Magnanti et J. Orlin. 1993. *Networks Flows, Theory, Algorithms, Applications*. Prentice Hall, Englewood Cliffs, New Jersey.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan et M. Szegedy. 1992. Proof verification and hardness of approximation problems. Proceedings 33rd Annual Symposium On Foundations of Computer Science, IEEE Computer Society, L.A Calif., 14-23.

[3] A.A. Assad. 1978. Multicommodity network flows. A survey. Networks 8, 37-91.

[4] A. Balakrishnan, T.L. Magnati et P. Mirchandani. 1998. Network design, in *Automated Bibliographies in Combinatorial Optimization,* eds. M. Dell'Amico, F. Maffioli et S. Martello. Wiley.

[5] C. Barnhart, C. Hane et P. Vance. 1998. Using branch-and-price to solve origin-destination integer multicommodity flow problems. Operations Research 32, 3, 208-220.

[6] A. Baveja et A. Srinisvasan. 2000. Approximation algorithms for disjoint paths and related routing and packing problems. Mathematics of Operations Research 25, 255-280.

[7] C. Berge. 1983 (révision des éditions précédentes 1970, 1973). *Graphes*. Gauthier-Villars. Paris. 400 p. (en français).

[8] D. Bertsimas, C-P. Teo et R. Vohra. 1999. Analysis of LP relaxations for multiway and multicut problems. Networks 34, 102-114.

[9] L. Brunetta, M. Conforti et M. Fischetti. 2000. A polyhedral approach to an integer multicommodity flow problem. Discrete applied Mathematics 101, 13-26.

[10] G. Calinescu, H. Karloff, et Y. Rabani. 1998. An improved approximation algorithm for Multiway Cut. Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pages 48-52, Dallas, Texas 230026 May. ACM Press.

[11] P. Camion. 1963. Matrices totalement unimodulaires et problèmes combinatoires. Université de Bruxelles, Thèse et Rapport Euratom , f.

[12] J. Cheriyan, H. Karloff et Y. Rabani. 2001. Approximating directed multicuts. FOCS. 320-328.

[13] S. Chopra et M.R. Rao. 1991. On the multiway cut polyhedron. Networks 21, 51-89.

[14] M. Chrobak. 2001. Reconstructing polyatomic structures from discrete X-rays : NP-completeness proof for three atoms. Theoretical Computer Science, 259, 81-98.

[15] S. Cosares et I. Saniee. 1994. An optimization problem related to balancing loads on SONET rings. Telecommunication Systems 3, 165-181.

[16] M-C. Costa, L. Létocart et F. Roupin. 2002. A greedy algorithm for multicut and integral multiflow in rooted trees. Operations Research Letters, à paraître.

[17] M-C. Costa. 2002. Polynomial algorithms to solve multiway cut and flow problems on trees. ECCO XV, Lugano. CEDRIC report.

[18] M-C. Costa, L. Létocart et F. Roupin. 2002. Minimal multicut and maximal integer multiflow : a survey. En soumission à European Journal of Operational Research.

[19] M-C. Costa, F-R. Monclar et M. Zrikem. 2002. Variable neighborhood search for the optimization of cable layout problem. Journal of Intelligent Manufacturing 13(5), 353-365.

[20] M-C. Costa, A. Hertz et M. Mittaz. 2002. Bounds and heuristics for the shortest capacitated paths problem. Journal of Heuristics 8(4), 449-466.

[21] W.H. Cunningham. 1991. The optimal multiterminal cut problem. DIMACS Series in Disc. Math. and Theor. Comput. Sci. 5, 105-120.

[22] W.H. Cunningham et J.F. Geelen. 1998. Integral solutions of linear complementary problems. Mathematics of Operations Research 23, 61-68.

[23] E. Dalhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour et M. Yannakakis. 1994. The complexity of multiway cuts. SIAM J. Comput. 23, 864-894.

[24] P. Erdos et L.A. Szekely. 1994. On weighted multiway cuts in trees. Math. Programming 65, 93-105.

[25] T. Erlebach et K. Jansen. 2001. The maximum edge-disjoint paths problem in bidirected trees. SIAM Journal on Discrete Mathematics 14, 3, 326-355.

[26] S. Even, A. Itaï et A. Shamir. 1976. On the complexity of timetable and multicommodity flow problems. SIAM J. Comp. 5, 691-703.

[27] L.R. Ford et D.R. Fulkerson. 1958. A suggested computation for maximal multicommodity network flows. Management Science, 5, 97-101.

[28] L.R. Ford et D.R. Fulkerson. 1962. Flows in networks. Princeton University Press.

[29] S. Fortune, J. Hopcroft et J. Wyllie. 1980. The directed subgraph homeomorphism problem. Theoretical Computer Science, Vol. 10, No. 4, 111-121.

[30] A. Frank. 1985. Edge-disjoint paths in planar graphs. J. Combin. Theory Ser. B 38, 164-178.

[31] A. Frank. 1990. Packing paths, circuits and cuts-a survey. In B. Korte, L. Lovasz, H.J. Prömel and A. Schrijver. *Paths, Flows and VLSI-Layout.* Algorithms and combinatorics 9. Springer-Verlag. Berlin. 47-100.

[32] V. Gabrel, A. Knippel et M. Minoux. 1999. Exact solution of multicommodity network optimization problems with general step cost functions. Oper. Res. Lett. 25, 15-23.

[33] N. Garg, V.V. Vazirani et M. Yannakakis. 1994. Multiway cuts in directed and node weighted graphs. In Proceedings ICALP, 487-498.

[34] N. Garg, V.V. Vazirani et M. Yannakakis. 1996. Approximate max-flow min-(multi)cut theorems and their applications. SIAM J. Comput. 25, 2, 235-251.

[35] N. Garg, V.V. Vazirani et M. Yannakakis. 1997. Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica 18, 3-20.

[36] J.F. Geelen. 1997. A generalization of Tutte's characterization of totally unimodular matrices. Journal of Combinatorial Theory, Séries B 70, 101-117.

[37] B. Gendron, T.G. Crainic et A. Frangioni. 1999. Multicommodity capacitated network design, in *Telecommunications Network Plannings,* eds. B. Sanso et P. Soriano. Kluwer Academic. 1-19.

[38] A.V. Goldberg, E. Tardos, R.E. Tarjan. 1990. Network flow algorithms. In B. Korte, L. Lovasz, H.J. Prömel and A. Schrijver. *Paths, Flows and VLSI-Layout.* Algorithms and combinatorics 9. Springer-Verlag. Berlin. 329-371.

[39] O. Goldschmidt, et D.S. Hochbaum. 1994. A polynomial algorithm for the k-cut problem for fixed k. Mathematics of operations research, Vol. 19, No 1.

[40] M. Gondran et M. Minoux. 1995 (révision des éditions précédentes 1971, 1985). *Graphes et Algorithmes.* Eyrolles. Paris. 588 p. (en français). *Graphs and Algorithms.* Wiley. 1986 (en anglais).

[41] F. Granot et M. Penn. 1996. Polynomial algorithms for (integral) maximum two-flows in vertex \ edge-capacited planar graphs. Discrete Applied Mathematics 70, 267-283.

[42] V. Guruswani, S. Khanna, R. Rajaraman, B. Shepherd et M. Yannakakis. 1999. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. Proc. ACM Sympos. Theory Comput. ACM Atlanta, GA, 19-28

[43] P.L. Hammer, P. Hansen et B. Simeone. 1984. Roof duality, complementarity and persistency in quadratic 0-1 optimization. Mathematical Programming 28, 121-155.

[44] W. Hasan et R. Motwani. 1995. Coloring away communication in parallel query optimization. Proceeding of the 21st VLDB Conference. Zurrich. Switzerland.

[45] C. Helmberg. 2000. Semidefinite programming for combinaorial optimization. Habilitationsschrift TU Berlin, ZIB-report ZR-00-34.

[46] C. Helmberg. 2000. A C++ implementation of the spectral bundle method. Manual version 1.1.1, ZIB-report ZR-00-35. http ://www.zib.de/helmberg/SBmethod.

[47] C. Helmberg et F. Rendl. 1997. A spectral bundle method for semidefinite programming. ZIB Preprint SC-97-37.

[48] A.J. Hoffman et J.B. Kruskal. 1956. Integral boundary points of convex polyhedra. Dans H.W. Khun et A.W. Tucker eds. 1956. *Linear inequalities and related systems.* Ann. Math. Study 38, Princeton University Press, Princeton, N.J., 223-246.

[49] T.C. Hu. 1963. Multicommodity network flows. Oper. Res. 11, 344-360.

[50] T.C. Hu. 1973. Two-commodity cut-packing problem. Discrete Mathematics 4, 108-109.

[51] ILOG SA. Ilog CPLEX 7.0 Reference Manual, 2000.

[52] A. Itai. 1978. Two-commodity flow. Journal of the Association for Computing Machinery 25,596-611.

[53] D. Karger, P. Klein, M. Thorup, C. Stein et N. Young. 1999. Better Rounding Algorithms for a Geometric Embedding Relaxation of Minimum Multiway Cut. STOC'99.

[54] R.M. Karp. 1975. On the complexity of combinatorial problems. Networks 5, 45-68.

[55] J. Kleinberg. 1996. Approximation algorithms for disjoint paths problems. PhD thesis, MIT, Cambridge, MA.

[56] J. Kleinberg et E. Tardos. 1995. Disjoint paths in densely embedded graphs. Proc. 36th IEEE FOCS'95.

[57] J. Kleinberg et E. Tardos. 1995. Approximation for the disjoint paths problem in high-diameter planar networks. Proc. 27th ACM STOC'95, 26-35.

[58] P. Kolman et C. Scheideler. 2002. Improved bounds for the unsplittable flow problem. SODA.

[59] E. Korach. 1982. Packing of T-cuts, and other aspects of dual integrality. Ph.D Thesis, University of Waterloo.

[60] E. Korach et M. Penn. 1993. A fast algorithm for maximum integral two-commodity flow in planar graphs. Discrete Applied Mathematics 47, 77-83.

[61] P. Kubat, A. Shulman, R. Vachani et J. Ward. 1996. Multicommodity flows in ring networks. INFORMS Journal on Computing Vol. 8, No. 3, 235-242.

[62] F.T. Leighton et S. Rao. 1999. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. J. ACM 46(6), 787-832.

[63] L. Létocart et F. Roupin. 2002. A semidefinite approach to solve multicut in trees. JOPT'02, Montréal, Canada.

[64] M.S. Levine. 2000. Fast randomized algorithms for computing minimum $\{3, 4, 5, 6\}$-way cuts. SODA, 735-742.

[65] A. Lobel. 1997. Vehicle scheduling in public transit and lagrangian pricing . ZIB Berlin. Preprint SC 97-16.

[66] M.V. Lomonosov. 1979. Multiflow feasibility depending on cuts. Graph Theory Newsl. 9, 4.

[67] M.V. Lomonosov. 1985. Combinatorial approaches to multiflow problems. Disc. Appl. Math. 11, 1-94.

[68] J.F. Lynch. 1975. The equivalence of theorem proving and the interconnection problem. ACM SIGDA Newslett. 5, 31-65.

[69] R. Manor et M. Penn. 1998. An extended planar algorithm for maximum integral two-flow. Networks 32, 67-76.

[70] M. Middendorf et F. Pfeiffer. 1993. On the complexity of the disjoint path problem. Combinatorica 13, 97-107.

[71] M. Minoux. 2001. Discret cost multicommodity network optimization problems and exact solution methods. Annals of Operations Research, 106, 19-46.

[72] J. Naor et L. Zosin. 1997. A 2-approximation algorithm for the directed multiway cut problem. Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS'97).

[73] H. Okamura at P.D. Seymour. 1981. Multicommodity flows in planar graphs. J. Combin. Theory Ser B 31, 75-81.

[74] A. Onaga et O. Kabusho. 1971. On feasibility conditions of multicommodity flows in networks. IEEE Trans. on Circuit Theory CT 18, 425-429.

[75] J.B. Orlin. 1993. A faster strongly polynomial minimum cost flow algorithm. Operations Research 41, 2. 338-349.

[76] C.H. Papadimitriou et M. Yannakakis. 1991. Optimization, approximation and complexity classes. J. Comput. System Sci. 43, 425-440.

[77] C. Picouleau. Ordonner-Réordonner. Habilitation à Diriger les Recherches. Paris VI et CEDRIC-CNAM. 2000.

[78] S. Poljak, F. Rendl et H. Wolkowicz. 1995. A recipe for semidefinite relaxation for (0,1)-quadratic programming. Journal of Global Optimization, 7, 51-73.

[79] S. Rajagopalan. 1994. Two commodity flows. Oper. Res. Letters, 15, 151-156.

[80] N. Robertson et P.D. Seymour. 1995. Graphs minors XIII : The disjoint paths problem. J. Comb. Theory, Ser. B, 63, 65-110.

[81] B. Rothschild et A. Whinston. 1966. On two commodity network flows. Operations Research 14, 377-387.

[82] F. Roupin. 2002. From linear to semidefinite programming : a scheme to obtain semidefinite relaxations for combinatorial problems. CEDRIC Report.

[83] M. Sakarovitch. 1973. Two commodity network flows and linear programming. Mathematical Programming 4, 1-20.

[84] A. Schrijver. 1990. Homotopic routing methods. In B. Korte, L. Lovasz, H.J. Prömel and A. Schrijver. *Paths, Flows and VLSI-Layout.* Algorithms and combinatorics 9. Springer-Verlag. Berlin. 329-371.

[85] A. Sebo. 1993. Integer plane multicommodity with a fixed number of demands. J. Combin. Theory, Ser. B, 59, 163-171.

[86] P.D. Seymour. 1978. A two-commodity cut survey. Discrete Mathematics 23, 177-181.

[87] P.D. Seymour. 1979. A short proof of the two-commodity flow problem. Journal of Combinatorial Theory, Séries B 26, 370-371.

[88] D. B. Shmoys. 1997. Cut problems and their application to divide-and-conquer. In D. Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Company. Boston. 192-234.

[89] A. Srivastav et P. Stangier. 2000. On complexity, representation and approximation of integral multicommodity flows. Discrete Applied Mathematics 99, 183-208.

[90] E. Tardos et V.V. Vazirani. 1993. Improved bounds for the max-flow min- multicut ratio for planar and $K_{r,r}$-free graphs. Inform. Proc. Lett. 47, 77-80.

[91] M. Yannakakis. 1985. On a class of totally unimodular matrices. Mathematics of Operations Research 10, 280-304.