Algorithmes Combinatoires et Relaxations par Programmation Linéaire et Semidéfinie. Application à la Résolution de Problèmes Quadratiques et d'Optimisation dans les Graphes.

Frédéric Roupin

Habilitation à Diriger des Recherches

Université Paris-Nord

Spécialité : Informatique

Présentée par

Frédéric Roupin

Algorithmes Combinatoires et Relaxations par Programmation Linéaire et Semidéfinie. Application à la Résolution de Problèmes Quadratiques et d'Optimisation dans les Graphes.

Soutenue le 24 novembre 2006 devant le jury :

Président	Pr. Christian Lavault
Rapporteurs	Pr. Monique Guignard
	D.R. Claude Lemaréchal
	Pr. Michel Minoux
Examinateurs	Pr. Alain Billionnet
	Pr. Marie-Christine Costa
	Pr. Philippe Michelon
	Pr. Gérard Plateau

Remerciements

Je tiens tout d'abord à adresser mes plus vifs remerciements à Monique Guignard, Claude Lemaréchal et Michel Minoux. Je suis heureux et particulièrement fier qu'ils m'aient fait l'honneur d'accepter d'être rapporteurs de cette habilitation. La lecture de leurs articles et ouvrages ont eu une influence considérable sur les travaux présentés dans ce document.

Je remercie Gérard Plateau pour m'avoir si chaleureusement accueilli et pour avoir coordonné et facilité mes démarches à l'Université Paris Nord, Christian Lavaux pour m'avoir fait l'honneur de présider ce jury, et Philippe Michelon pour avoir accepté sans hésitation de faire partie de ce jury.

Mes remerciements vont également à tous les membres de l'équipe Optimisation Combinatoire du laboratoire Cédric avec lesquels il est si agréable de travailler. La plupart d'entre eux sont co-auteurs des travaux présentés dans ce document : mon ancien directeur de thèse, Alain Billionnet, qui m'a fait découvrir la recherche et dont les conseils, la grande culture et la lucidité scientifiques m'ont toujours été précieux, Marie-Christine Costa pour son enthousiasme et son énergie, Sourour Elloumi, Alain Faye avec qui je partage le même bureau à l'IIE, Christophe Picouleau, Agnès Plateau, Eric Soutif, et enfin Lucas Létocart, Cédric Bentz, et Nicolas Derhy actuels ou anciens thésards que j'ai (ou que j'ai eu) le plaisir de co-encadrer avec Marie-Christine.

Enfin, je remercie ma femme Caroline pour sa patience et pour avoir supporté mes horaires de travail souvent non conventionnels, et mes deux fils Olivier et Julien pour toutes les questions surprenantes (et souvent difficiles) qu'ils me posent chaque jour.

Résumé

Cette synthèse de travaux de recherche concerne l'algorithmique dans les graphes et l'utilisation de la programmation linéaire et semidéfinie positive (SDP) dans le cadre de la résolution exacte ou approchée de plusieurs problèmes fondamentaux de l'Optimisation Combinatoire. L'approche semidéfinie, qui conduit à des relaxations convexes mais non-linéaires, a permis d'obtenir de remarquables résultats théoriques en approximation et devient à présent utilisable en pratique (tout comme la programmation linéaire qui en est un cas particulier).

Nos travaux comportent une forte composante algorithmique et des études de complexité de plusieurs problèmes d'optimisation dans les graphes. Nous considérons tout d'abord le problème de la recherche d'un sous-graphe dense de taille fixée pour lequel nous présentons un algorithme polynomial avec garanties de performances fondé sur la programmation linéaire et quadratique. Puis, nous étudions les problèmes de multiflots entiers et de multicoupes pour lesquels nous avons identifié de nombreux cas polynomiaux dans des graphes particuliers importants en pratique : arborescences, grilles, anneaux. D'une part, les solutions fractionnaires fournies par certaines relaxations linéaires de ces problèmes sont le point de départ d'algorithmes de résolution efficaces. D'autre part, les propriétés des programmes linéaires utilisés nous permettent également d'élaborer des algorithmes purement combinatoires et de démontrer leur validité (matrices totalement unimodulaires, théorème des écarts complémentaires).

Nous proposons également des approches systématiques pour élaborer des relaxations semidéfinies pour les programmes quadratiques, modèles de très nombreux problèmes combinatoires et continus. Plus précisément, nous étudions les liens entre relaxations semidéfinies et des relaxations lagrangiennes partielles de programmes quadratiques contenant des contraintes linéaires. En particulier, les fonctions quadratiques constantes sur une variété affine sont entièrement caractérisées. Ceci permet de facilement comparer les différentes familles de contraintes redondantes proposées dans la littérature dans l'approche semidéfinie dans le cadre unifié de l'approche lagrangienne. Puis, nous présentons un algorithme pour élaborer des relaxations semidéfinies à partir de relaxations linéaires existantes. L'objectif est de profiter des résultats théoriques et expérimentaux obtenus dans l'approche linéaire. Nous avons développé un logiciel (SDP_S) grâce à ces résultats. Il permet de formuler automatiquement et facilement des relaxations semidéfinies pour tout problème pouvant être formulé comme un programme quadratique en variables bivalentes. Notre méthode peut se généraliser à certains programmes à variables mixtes.

Enfin, nous appliquons les méthodes décrites précédemment à une série de problèmes combinatoires classiques. Nos expérimentations montrent que l'approche semidéfinie est à présent pertinente dans la pratique sous certaines conditions. Premièrement, nous présentons des méthodes de séparation/évaluation efficaces fondées sur la SDP pour la résolution exacte des problèmes MAX 2SAT et VERTEX-COVER. Deuxièmement, nous proposons plusieurs bornes par SDP de grande qualité pour des problèmes particulièrement difficiles à résoudre par les approches linéaires : K-CLUSTER, CMAP (un problème de placement de tâches avec contraintes de ressources), et le problème de l'affectation quadratique (QAP). Pour ce dernier nous présentons également un algorithme de coupes performant fondé sur la programmation semidéfinie. Afin d'obtenir des algorithmes efficaces en pratique, nous mettons en oeuvre non seulement nos méthodes d'élaboration de relaxations SDP, mais également des techniques algorithmiques issues de l'approximation polynomiale, ainsi que des outils spécifiques de résolution numérique des programmes semidéfinis.

Table des matières

Publications jointes au dossier d'habilitation

Le texte intégral de ces publications est donné en annexe. La numérotation correspond à celle indiquée dans les références à la fin du présent document.

Roupin (1997) [98] On approximating the memory-constrained module allocation problem. *Information Processing Letters*, 61(4) pp 205-208.

Roupin (1997) [97] A fast Heuristic for the module allocation problem. Proceedings of the 15th *IMACS World Congress*, 24-29 Août, Berlin, vol. 1, pp 405-410.

Billionnet, Roupin (1997) [25] Linear Programming to approximate quadratic 0-1 maximization problems. Proceedings of the 35th *Southeast ACM conferenc*e, 2-4 avril, Murfreesboro, USA, pp 171-173.

Cung, Van Hoeve, Roupin (2000) [40, 71] A parallel Branch-and-Bound algorithm using a semidefinite programming relaxation for the maximum independent set. *ROADEF'2000*, 26-28 Janvier, Nantes. Rapport de recherche de W. J. Van Hoeve, 1999.

Delaporte, Jouteau, Roupin (2002) [44] SDP_S : a Tool to formulate and solve Semidefinite relaxations for Bivalent Quadratic problems. Logiciel et documentation disponibles à http ://semidef.free.fr. Présenté à *ROADEF'2003*, 26-28 Février, Avignon.

Costa, Létocart, Roupin (2003) [37] A greedy algorithm for multicut and integral multiflow in rooted trees. *Operations Research Letters* vol. 31(1), pp 21-27.

Elloumi, Roupin, Soutif (2003) [46] Comparison of Different Lower Bounds for the Constrained Module Allocation Problem. Rapport scientifique CEDRIC 473. En révision pour *RAIRO*.

Roupin (2004) [100] From Linear to Semidefinite Programming : an Algorithm to obtain Semidefinite Relaxations for Bivalent Quadratic Problems. *Journal of Combinatorial Optimization* 8(4) pp 469-493.

Roupin (2004) [101] L'approche par Programmation Semidéfinie en Optimisation Combinatoire. Article invité dans le *Bulletin de la ROADEF* No 13 pp 7-11.

Costa, Létocart, Roupin (2005) [38] Minimal multicut and maximal integer multiflow : a survey. *European Journal on Operations Research* 162(1), pp 55-69. Mise à jour avec Cédric Bentz : A bibliography on multicut and integer multiflow problems. Rapport scientifique CEDRIC 654, 2004.

Faye, Roupin (2005) [49] A cutting planes Algorithm based upon a Semidefinite relaxation for the Quadratic Assignment Problem. European Symposium on Algorithms, *ESA 2005*, 3-6 Octobre, Majorque, Espagne, *Lecture Notes in Computer Science* No 3669, pp 850-861.

Aubry, Delaporte, Jouteau, Ritte, Roupin (2005) [9] User guide for SDP_SX. Documentation du logiciel SDP_SX, interface graphique de SDP_S.

Bentz, Costa, Roupin (2006) [17] Maximum integer multiflow and minimum multicut problems in uniform grid graphs. A paraître dans *Journal of Discrete Algorithms*. Disponible en ligne.

Faye, Roupin (2006) [48] Partial Lagrangian relaxation for General Quadratic Programming. A paraître dans *4'OR, A Quarterly Journal of Operations Research*. Disponible en ligne.

Billionnet, Roupin (2006) [26] A Deterministic Approximation Algorithm for the Densest k-Subgraph Problem. A paraître dans *International Journal of Operational Research*. Rapport de Recherche CEDRIC 486 disponible à http ://cedric.cnam.fr.

Bentz, Costa, Létocart, Roupin (2006) [16] Minimal multicut and maximal integer multiflow in rings. Rapport scientifique CEDRIC 1050. Soumis à *Information Processing Letters*.

Introduction Générale

La programmation mathématique est une approche fréquemment utilisée pour résoudre des problèmes d'optimisation combinatoire. Son large spectre d'application, son caractère générique, et les progrès fulgurants accomplis ces dernières années par les outils de résolution numérique des programmes linéaires et non-linéaires ont rendu son utilisation quasi-systématique pour résoudre de larges classes de problèmes. Une bonne partie de l'art du chercheur opérationnel réside alors dans l'élaboration d'une modélisation adéquate. Car si certaines propriétés d'un problème combinatoire sont intrinsèques, les formulations sont nombreuses et très inégales en termes d'efficacité. Elaborer ces formulations, les comparer théoriquement et en pratique, et enfin comprendre et en maîtriser le comportement numérique en utilisant les outils informatiques adaptés constituent donc des étapes essentielles lors de la résolution d'un problème avec cette approche.

Mais loin d'être un simple outil de formulation, la programmation mathématique permet également de guider la recherche et la conception d'algorithmes purement combinatoires ou de prouver leur validité. C'est particulièrement le cas pour les problèmes d'optimisation issus de la théorie des graphes, où la notion de dualité est fortement présente. Qu'il s'agisse d'obtenir des bornes, de démontrer une inégalité, de déterminer la complexité d'un problème, ou de prouver la correction d'un algorithme combinatoire, la programmation mathématique est une approche particulièrement efficace et élégante pour parvenir au résultat.

Ces notions sont récurrentes dans le présent document qui regroupe un ensemble de travaux concernant d'une part l'élaboration et l'utilisation de relaxations semidéfinies et linéaires pour la résolution exacte ou approchée de plusieurs problèmes souvent $\mathcal{NP}$-difficiles, et d'autre part l'utilisation de ces programmes mathématiques comme outil de preuve dans des approches purement combinatoires. La programmation semidéfinie positive (SDP) y tient une place importante. Approche récente en optimisation combinatoire, la SDP a tout d'abord été rendue populaire grâce aux résultats spectaculaires qu'elle a permis d'obtenir en approximation polynomiale. C'est un cas particulier de la programmation convexe, et elle peut être vue comme une généralisation de la programmation linéaire. Elle a néanmoins été exclue dans un premier temps du champ pratique

car peu de logiciels permettaient de résoudre efficacement les SDP. Nos travaux montrent que pour certains problèmes la programmation semidéfinie est à présent une approche pertinente qui permet d'obtenir des bornes de grande qualité dans des temps parfois plus courts que d'autres approches.

Dans la première partie, nous utilisons *la programmation linéaire pour construire des solutions (approchées ou exactes) ou pour démontrer la validité d'algorithmes purement combinatoires* pour plusieurs problèmes d'optimisation dans les graphes. Plus précisément, nous présentons un algorithme approché avec garantie de performance au pire cas pour le problème du sous-graphe dense de taille fixée (chapitre 2), ainsi que plusieurs algorithmes polynomiaux pour les problèmes de multiflots entiers et de multicoupes dans des graphes particuliers importants en pratique : arborescences, anneaux, grilles (chapitre 3).

Dans la deuxième partie, nous décrivons des *approches systématiques pour élaborer des relaxations semidéfinies pour les programmes quadratiques*. Premièrement, après une courte introduction à la programmation semidéfinie (chapitre 4), nous étudions les liens entre relaxations semidéfinies et l'approche lagrangienne pour les programmes quadratiques qui contiennent des contraintes linéaires (chapitre 5). Ce schéma général permet d'appréhender les différentes familles de contraintes redondantes connues dans un cadre unifié, et de mieux comprendre le comportement numérique et l'efficacité des différentes relaxations semidefinies proposées dans la littérature, puis d'en concevoir de nouvelles sur des bases théoriques solides. Deuxièmement, nous présentons une méthode pour élaborer des relaxations semidéfinies à partir de relaxations linéaires existantes (chapitre 6). Cette démarche a l'avantage d'utiliser les résultats des études théoriques et expérimentales déjà effectuées, et de plus garantit d'obtenir de meilleures relaxations que celles de départ. Le caractère mécanique de notre approche a par ailleurs trouvé un aboutissement informatique dans le développement d'un logiciel formulant automatiquement des relaxations semidéfinies sans qu'aucune expertise de l'utilisateur dans ce domaine ne soit nécessaire.

Enfin, la troisième partie traite de *l'application des méthodes et algorithmes précédents pour la résolution exacte ou approchée de plusieurs problèmes combinatoires classiques*. Nous nous intéressons tout d'abord à l'utilisation de la programmation semidéfinie dans le cadre de la résolution exacte de problèmes combinatoires (chapitre 7), puis à l'obtention de bornes et à la résolution approchée de problèmes particulièrement difficiles où l'approche par programmation linéaire se révèle souvent insuffisante (chapitre 8).

Un bilan des résultats obtenus et les perspectives de recherche liées aux travaux présentés concluent ce document (chapitre 9).

Première partie

Relaxations linéaires et algorithmes combinatoires pour certains problèmes d'optimisation dans les graphes

1

Introduction

De nombreux problèmes combinatoires peuvent être formulés comme des programmes linéaires en nombres entiers ou mixtes. L'objectif habituel de ce type de démarche est de pouvoir utiliser des logiciels de modélisation et de résolution numérique pour cette catégorie de programmes mathématiques. Ces outils sont en effet devenus particulièrement simples d'emploi et très efficaces ces quinze dernières années. Nous donnerons des éléments pour adopter une telle démarche dans la deuxième partie de ce document pour la programmation semidéfinie. Ici, si nous allons effectivement utiliser la programmation linéaire pour résoudre de manière exacte ou approchée plusieurs problèmes combinatoires, nous lui adjoindrons également des approches combinatoires. Dans plusieurs cas, la programmation linéaire aura même totalement disparu des algorithmes finaux, et ne subsistera que dans les preuves de correction de ces derniers. Cependant, les résultats très forts concernant la dualité en programmation linéaire (en particulier le théorème des écarts complémentaires), et la notion de matrice totalement unimodulaire seront des outils essentiels pour élaborer nos algorithmes. Dans les deux prochains chapitres, nous allons tout d'abord décrire un algorithme approché avec une garantie de performance au pire cas pour le problème $\mathcal{NP}$-difficile du sous-graphe dense de taille donnée (chapitre 2), puis identifier et résoudre des cas particuliers polynomiaux des problèmes de multicoupes et de multiflots entiers en effectuant des hypothèses supplémentaires sur la structure des graphes décrivant les instances (chapitre 3).

Les travaux présentés dans cette partie constituent un prolongement de ceux de ma thèse [96] qui traite de l'approximation de programmes quadratiques en 0-1 soumis à des contraintes linéaires, et de son application aux problèmes de placement et de partition de graphes. En particulier, nous allons utiliser le schéma général de construction de solutions approchées pour les programmes quadratiques en 0-1 utilisant à la fois les relaxations continues du programme initial et d'une formulation linéaire de celui-ci (voir [25] fourni dans l'annexe). Le succès de cette méthode dépend fortement des sauts d'intégrité entre les différents programmes considérés, mais également de la possibilité de construire des solutions continues particulières. Nous retrouverons cette problématique dans le prochain chapitre. Au lieu d'utiliser une approche déterministe

comme la précédente, d'autres auteurs ont proposé des algorithmes approchés aléatoires fondés sur la programmation linéaire, en utilisant notamment les travaux fondateurs de P. Raghavan et C. Thompson [93, 94] pour rendre déterministes leurs algorithmes. Plus récemment, des résultats d'approximation spectaculaires ont été obtenus par la programmation semidéfinie en construisant aléatoirement des solutions admissibles à partir des solutions optimales des SDP et en calculant l'espérance du ratio d'approximation obtenu [60], les techniques utilisées devenant de plus en plus sophistiquées (voir par exemple la méthode d'arrondi RPR^2 présentée dans [53]). Comme nous l'illustrerons dans le prochain chapitre, ces approches aléatoires sont beaucoup plus difficiles à mettre en oeuvre pour des problèmes avec des contraintes autres que celles d'intégrité sur les variables.

Tous ces travaux relèvent cependant de la même démarche : écrire une relaxation du problème facile à résoudre par programmation mathématique afin de borner la valeur optimale de l'instance, puis utiliser une *solution* optimale de la relaxation pour construire une solution admissible du problème initial (si possible optimale ou au moins en garantissant une performance au pire cas). La dernière étape (optionnelle) consiste à résoudre la relaxation par un algorithme combinatoire afin de diminuer la complexité totale de l'approche.

2

Un algorithme approché pour le problème du sous-graphe dense

2.1 Introduction

Soit un entier $1 \leq k \leq n$, et $G = (V, E)$ un graphe non orienté de n sommets $\{v_1, \ldots, v_n\}$, dont les arêtes $[v_i, v_j]$ sont valuées positivement par une matrice $W = (W_{ij})$ (on a $W_{ii} = 0$ pour tout i dans $\{1, \ldots, n\}$). Le problème K-CLUSTER consiste à trouver un sous-graphe de G possédant k sommets (appelé k-cluster) de poids maximal. Il est facile de constater que ce problème est $\mathcal{NP}$-difficile même dans le cas non-pondéré puisqu'il existe une clique de taille k si et seulement si il existe un k-cluster de valeur $\frac{k(k-1)}{2}$. En fait, K-CLUSTER reste $\mathcal{NP}$-difficile même dans les graphes bipartis [34]. Ce problème a de nombreuses applications et il a donc été largement étudié. C'est pourquoi il apparaît dans la littérature sous différents noms : HEAVIEST k-SUBGRAPH PROBLEM [18], P-DISPERSION PROBLEM, P-DEFENCE-SUM PROBLEM [77], et DENSEST K-SUBGRAPH PROBLEM lorsque les arêtes ne sont pas pondérées. Nous allons présenter pour ce dernier problème, noté DSP, un algorithme déterministe avec garantie de performance au pire cas fondé sur la programmation linéaire [26]. A la section 8.2 du chapitre 8 nous discuterons de l'approche semidéfinie pour le problème plus général K-CLUSTER.

2.2 Approche déterministe ou aléatoire ?

Dans DSP, nous devons trouver un sous-graphe de k sommets de densité maximale dans un graphe $G = (V, E)$ non orienté de n sommets $\{v_1, \ldots, v_n\}$. Une formulation du problème comme programme quadratique en variables $0 - 1$ est :

$$(DSP) \left\{ \max \; f(x) = \sum_{i<j,\, [v_i, v_j] \in E} x_i x_j \; : \; \sum_{i=1}^{n} x_i = k,\; x \in \{0,1\}^n \right\}$$

où $x_i = 1$ si et seulement si le sommet v_i est pris dans le sous-graphe. Pour K-CLUSTER il existe un certain nombre d'algorithmes approchés aléatoires fondés soit sur des relaxations semidéfinies [52, 63, 64, 72, 105], soit sur des relaxations linéaires [63, 64]. Ces travaux peuvent donc

être également appliqués à (DSP). Pour ceux utilisant la programmation semidéfinie, l'idée est d'interpréter la solution optimale d'une relaxation semidéfinie du problème comme un champ de vecteurs unitaires, puis de construire une solution admissible en utilisant un vecteur pris uniformément distribué sur la sphère unité. Cette idée a été utilisée pour la première fois dans [60] et a permis l'élaboration du célèbre algorithme 0,878-approché pour MAX-CUT (voir la section 7.3 du chapitre 7 pour une présentation plus détaillée de cette méthode). Rappelons qu'un algorithme ϵ-approché pour un problème de maximisation (Π) renvoie pour toute instance I de (Π) une solution admissible de valeur v_I^a telle que $\frac{v_I^a}{v_I^*} \geq \epsilon$, où v_I^* est la valeur optimale de I. La valeur de $0 \leq \epsilon \leq 1$ est le ratio d'approximation de l'algorithme.

Cependant les algorithmes obtenus par cette démarche sont aléatoires. Se pose alors le problème d'en déduire des algorithmes déterministes possédant les mêmes ratios de performance au pire cas que ceux des algorithmes approchés correspondants. Nous renvoyons le lecteur à l'excellente synthèse présentée dans [104] pour une présentation complète des techniques pour rendre déterministes certains algorithmes aléatoires. Dans les travaux précédemment cités, les auteurs n'effectuent pas explicitement cette dernière étape, mais certains évoquent la méthode des probabilités conditionnelles, qui est une procédure effectivement polynomiale mais parfois de complexité très élevée. Dans [65] cette dernière est par exemple égale à $O\left(n^{30}\right)$ pour le problème 3-VERTEX COLORING.

De plus, il existe une difficulté supplémentaire pour les problèmes K-CLUSTER et DSP : respecter la contrainte de cardinalité, i.e. obtenir un sous-graphe contenant exactement k sommets. Il faut donc "réparer" la solution tout en essayant de dégrader le moins possible le ratio d'approximation. Nous allons à présent discuter ce point en utilisant la programmation linéaire, mais la problématique reste la même si on utilise la programmation semidéfinie. Remarquons que DSP peut être également formulé comme :

$$(P)\ \left\{ \max\ f_L(x) = \sum_{i<j,\,[v_i,v_j]\in E} \min\left(x_i, x_j\right)\ :\ \sum_{i=1}^{n} x_i = k,\ x \in \{0,1\}^n \right\}$$

La relaxation continue de (P) peut être formulée comme le programme linéaire continu :

$$(\underline{PL})\ \left\{ \max \sum_{i<j,\,[v_i,v_j]\in E} z_{ij}\ :\ \sum_{i=1}^{n} x_i = k,\ x \in [0,1]^n,\ z_{ij} \leq x_i \text{ et } z_{ij} \leq x_j\ \forall i<j,\ [v_i,v_j] \in E \right\}$$

Un algorithme très simple (mais doublement aléatoire) attribué à M.X. Goemans est cité dans [52]. L'idée est de partir d'une solution optimale (x', z') de $(\underline{PL})$, puis d'arrondir les n composantes de x' à 1 avec la probabilité $\sqrt{x'_i}$. Ainsi, la probabilité qu'une arête $[v_i, v_j]$ soit présente dans le sous-graphe obtenu vaut au moins z'_{ij}, et donc l'espérance du nombre total d'arêtes est supérieure ou égale à la valeur optimale de $(\underline{PL})$. Mais le sous-graphe obtenu peut contenir

plus de k sommets. Plus précisément, on peut vérifier que l'espérance du nombre de sommets est inférieur ou égal à $\sqrt{nk}$. Il faut donc sélectionner k sommets parmi ces $\sqrt{nk}$ sommets (en les choisissant aléatoirement par exemple). Ce qui conduit à une perte de $\frac{k^2}{(\sqrt{nk})^2}$ dans le ratio, et finalement produit une solution $\frac{k}{n}$-approchée. On peut rendre déterministe la première étape de cet algorithme aléatoire en utilisant les résultats de [93, 94]. On montre alors que la solution entière x obtenue vérifie $\sum_i x_i \leq \sqrt{nk} + O(\sqrt{nlog(n)})$ avec une grande probabilité [7]. Dans [63] les auteurs ne procèdent pas à cette première étape mais en revanche ces derniers sélectionnent les k sommets finaux en utilisant une heuristique gourmande qui consiste à retirer un à un $n-k$ sommets de façon à perdre le moins d'arêtes possible à chaque étape (cette heuristique est bien connue et déjà utilisée dans [8, 105]). Cela conduit a un ratio d'approximation égal à $\left(\frac{1}{1+k^{-\frac{1}{3}}}\right)^2 \left(\frac{k}{n} - \frac{1}{2}n^2 e^{-\frac{n}{3}^{\frac{1}{9}}}\right)$. Ainsi, même si n est très grand, la "réparation" finale coûte un facteur de $\left(\frac{1}{1+k^{-\frac{1}{3}}}\right)^2$ dans le ratio $\frac{k}{n}$. Par exemple, pour obtenir un ratio de $\frac{0{,}95k}{n}$ on doit avoir $\left(\frac{1}{1+k^{-\frac{1}{3}}}\right)^2 \geq 0,95$ et donc k doit être plus grand que 57038. De plus, on ne tient pas compte ici du terme supplémentaire $O(\sqrt{nlog(n)})$ puisque la première étape aléatoire est conservée par ces auteurs. Ces quelques réflexions montrent la grande difficulté d'obtenir pour notre problème des algorithmes déterministes de mêmes complexité et ratio d'approximation que les algorithmes aléatoires présentés dans la littérature. Il est néanmoins possible de parvenir à de tels résultats en effectuant des hypothèses supplémentaires sur le graphe G. Ainsi dans [7], en partant d'une formulation quadratique en variables 0-1, les auteurs montrent qu'il existe un schéma d'approximation polynomial pour DSP si G est partout dense (i.e. si tous les degrés des sommets de G sont en $\Omega(n)$), ou si G est un graphe dense (i.e. le nombre d'arêtes dans G est en $\Omega\left(n^2\right)$) et que $k = \Omega\left(n\right)$.

2.3 Un algorithme déterministe pour la résolution approchée de DSP

Considérant les résultats de la section précédente, nous avons proposé dans [26] un algorithme entièrement déterministe fondé sur une solution optimale particulière de ($\underline{PL}$). Aucune des étapes de notre algorithme n'étant aléatoire, et chacune d'entre elles pouvant être implémentée en $O\left(n^2\right)$, la complexité totale de notre approche est celle de la programmation linéaire. Le ratio d'approximation obtenu (valable pour tout $k \geq 3$) est $\max\left(d, \frac{8k}{9n}\right)$ où $d = \frac{2|E|}{n(n-1)}$ est la densité du graphe G. L'approche utilisée est celle que nous avons proposée dans [25]. Le principe est de considérer une solution particulière de la relaxation linéaire ($\underline{PL}$) comme une solution admissible de la relaxation continue de (DSP) (programme quadratique), puis de construire une solution en 0-1 et donc admissible de (DSP) tout en préservant au maximum à chaque étape le ratio d'approximation. Nous décrivons ci-après les étapes et résultats principaux de notre algorithme (voir [26] dans l'annexe pour les détails et les preuves).

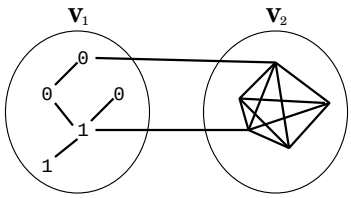

Fig. 2.1. Solution x^Q de la relaxation continue de (DSP) telle que $f\left(x^Q\right) \geq f\left(x^{L\beta}\right)$

1. Résoudre $(\underline{PL})$ pour obtenir une solution optimale x^L.
2. Construire $x^{L\beta}$, solution optimale particulière de $(\underline{PL})$, à partir de x^L. Toutes les composantes non entières de $x^{L\beta}$ sont égales entre elles. On montre que $f\left(x^{L\beta}\right) \geq \frac{k}{n} f_L\left(x^{L\beta}\right)$.
3. Construire x^Q, solution admissible particulière de la relaxation continue de (DSP), à partir de $x^{L\beta}$. (V_1, V_2) étant une partition de V, x^Q est telle que : $f\left(x^Q\right) \geq f\left(x^{L\beta}\right)$, $x_i^Q \in \{0,1\}$ pour tout i dans V_1, $0 < x_i^Q < 1$ pour tout i dans V_2, et le sous-graphe engendré par les sommets de V_2 est une clique (voir la figure 2.1).
4. Construire x^P, solution admissible de (DSP), à partir de x^Q comme suit. Soient $T_{1\times 1} = \left\{(i,j) \in V_1^2 : i < j,\, [v_i, v_j] \in E\right\}$, $T_{2\times 2} = \left\{(i,j) \in V_2^2 : i < j,\, [v_i, v_j] \in E\right\}$, $s = \mid V_1 \mid$, et $C_i = \sum_{j \in \Gamma(i) \cap V_1} x_i^Q$, on considère le programme :

$$\begin{cases} \max \sum_{i \in V_2} C_i x_i + \sum_{(i,j) \in T_{2\times 2}} x_i x_j + \sum_{(i,j) \in T_{1\times 1}} x_i^Q x_j^Q \\ \text{s.c.} \quad \sum_{i \in V_2} x_i = k - s \\ \qquad x_i \in \{0,1\}\,, \forall i \in V_2 \end{cases}$$

Sa valeur optimale est $\sum_{i \in V_{2[k-s]}} C_i + \frac{(k-s)(k-s-1)}{2} + \sum_{(i,j) \in T_{1\times 1}} x_i^Q x_j^Q$, où $V_2[k-s]$ est l'ensemble des indices des $k-s$ variables x_i $(i \in V_2)$ ayant les $k-s$ plus grands C_i. x^P est alors définie par : $x_i^P = x_i^Q$ pour i dans V_1, $x_i^P = 1$ pour i dans $V_2[k-s]$, et $x_i^P = 0$ sinon.

On montre alors que :

Théorème 1 [26] *DSP admet un algorithme déterministe approché qui fournit une solution x^P de (DSP) telle que $f(x^*) \leq \frac{n}{k}\left(f\left(x^P\right) + \frac{|V_2|}{8}\right)$, où x^* est une solution optimale de (DSP). De plus, la complexité de cet algorithme est celle de la programmation linéaire.*

corollaire 1 [26] *DSP admet un algorithme approché qui fournit une solution x^P de (DSP) telle que $\frac{f(x^*)}{f(x^P)} \leq \frac{9n}{8k}$, où x^* est une solution optimale de (DSP).*

Un autre résultat de notre étude est que le saut d'intégrité entre (DSP) et sa relaxation continue (un programme quadratique) est borné. Le ratio entre les deux valeurs optimales vaut en effet au plus $\frac{1}{2}$ (voir le corollaire 3.4 de [26] donné en annexe).

2.4 Conclusion

L'algorithme déterministe $\max\left(d, \frac{8k}{9n}\right)$-approché ($d$ est la densité du graphe G) que nous avons présenté peut être facilement appliqué à des instances de grande taille puisque sa complexité est celle de la résolution d'un programme linéaire continu comportant $O\left(n^2\right)$ contraintes. En effet, même si les preuves concernant la valeur du ratio au pire cas sont assez techniques, la partie algorithmique permettant d'obtenir la solution approchée pour DSP est simple à implémenter et de complexité faible ($O\left(n^2\right)$). De plus, le ratio d'approximation que nous avons obtenu n'est pas une espérance comme c'est le cas pour les approches aléatoires évoquées dans l'introduction. Certes, nous n'avons pas obtenu celui de l'algorithme (doublement) aléatoire de M.X. Goemans fondé sur le même programme linéaire. Cependant, rendre ce dernier algorithme totalement déterministe pose de réels problèmes, puisque que le ratio d'approximation obtenu dans [63] pour rendre seulement la dernière phase aléatoire déterministe est meilleur que le notre seulement si n est très grand et k supérieur à 4481 (pour avoir $\left(\frac{1}{1+k^{-\frac{1}{3}}}\right)^2 \geq \frac{8}{9}$).

La programmation mathématique nous a non seulement fourni ici une borne de la valeur optimale mais également une solution fractionnaire qui nous a permis en passant par un programme quadratique continu (relaxation d'une formulation en variables 0-1 du problème) de construire de façon combinatoire une solution approchée pour DSP. Cette démarche est générale et nous l'avons appliquée à d'autres problèmes dans [25, 96]. Nous retrouverons une approche semblable dans le cadre de l'approche par programmation semidéfinie pour ce problème (et pour d'autres) au chapitre 8.

3

Multiflots entiers et multicoupes dans des graphes particuliers

3.1 Introduction

Nous présentons dans ce chapitre un ensemble de résultats de complexité et d'algorithmes polynomiaux (le plus souvent purement combinatoires) pour les problèmes de multiflots entiers (noté IMFP pour "Integral MultiFlow Problem") et multicoupes (noté MCP pour "MultiCut Problem") dans des graphes particuliers : anneaux, arborescences et grilles. Ce travail s'est effectué dans le cadre de deux thèses (Lucas Létocart [85] et Cédric Bentz [13]), co-encadrées par Marie-Christine Costa, professeur au CNAM, et moi-même. Ces recherches se poursuivent actuellement dans les mêmes conditions d'encadrement avec la thèse de Nicolas Derhy en particulier pour des variantes avec des contraintes de cardinalité supplémentaires [14].

On considère un graphe $G = (V, E)$ orienté ou non dont chaque arête e (resp. arc a) est valuée par un entier naturel u_e (resp. u_a), et une liste $\mathfrak{L}$ de K paires de terminaux $\{s_k, t_k\}$, $k \in \{1, \ldots, K\}$, placés aux sommets de G. Dans IMFP, on cherche à maximiser la somme de K flots entiers $f_1, \ldots, f_K$ routés respectivement de s_k à t_k, qui respectent donc les contraintes de capacité des arêtes (ou arc) et celles de conservation. Dans MCP, on doit déterminer un sous-ensemble C de E de poids total minimum qui déconnecte toutes les paires de terminaux, i.e. tel qu'il n'existe pas de chaîne d'extrémités s_k et t_k pour tout k dans $\{1, \ldots, K\}$ (ou de chemin de s_k à t_k dans le cas orienté) dans le graphe $G' = (V, E \setminus C)$.

Nous avons présenté dans [38] une synthèse des nombreux travaux concernant ces deux problèmes, leurs variantes, et leurs cas particuliers. La bibliographie ainsi que le tableau résumant les résultats de complexité et d'approximation rappelés ou présentés dans cet article ont été mis à jour dans [15]. Ces deux documents sont fournis dans l'annexe. Par conséquent, nous allons rappeler dans cette section uniquement les définitions et formulations des problèmes qui nous seront directement utiles par la suite. Notons simplement que pour $K = 1$ on retrouve les problèmes polynomiaux classiques de coupe minimale/flot maximal, mais que IMFP et MCP sont $\mathcal{NP}$-difficiles et même APX-difficiles pour $K \geq 3$ (il n'existe donc pas d'algorithme approché

avec un ratio d'approximation fixé), et ceci même dans les graphes planaires [57].

IMFP et MCP possèdent un lien de dualité naturel qui va jouer un rôle essentiel dans la conception et les preuves des algorithmes que nous allons présenter. La programmation mathématique n'apparaîtra donc pas forcément dans les algorithmes finaux, mais son rôle est essentiel. Considérons l'ensemble Π de toutes les chaînes d'extrémités s_k et t_k (ou chemins de s_k à t_k dans le cas orienté), k variant dans $\{1,\dots,K\}$. $M =\mid \Pi \mid$, le cardinal de $\Pi = \{p_1,\dots,p_M\}$ peut évidemment être très grand dans un graphe quelconque. Notons ϕ_i le flot entier routé sur la chaîne (ou chemin si G est orienté) p_i. On peut alors formuler IMFP et MCP comme les programmes linéaires suivants :

$$(P-IMFP)\begin{cases} \max \sum_{i=1}^{M} \phi_i & \\ \text{s.c.} \sum_{i \text{ tel que } e\in p_i} \phi_i \leq u_e \ \forall e \in E & (1) \\ \phi_i \in \mathbb{N} \qquad\qquad \forall i \in \{1,\dots,M\} & \end{cases}$$

$$(P-MCP)\begin{cases} \min \sum_{e\in E} u_e\, c_e & \\ \text{s.c} \sum_{e\in p_i} c_e \geq 1 \ \forall i \in \{1,\dots,M\} & (2) \\ c_e \in \{0,1\} \ \forall e \in E & \end{cases}$$

Il existe bien sûr d'autres formulations de IMFP et de MCP comme programmes linéaires. Si nous relâchons à présent les contraintes d'intégrité sur les variables de $(P-IMFP)$ et $(P-MCP)$ en les remplaçant par des contraintes de positivité, nous constatons que les deux programmes linéaires continus obtenus sont duaux. Les conditions des écarts complémentaires pour un couple de solutions optimales continues (ϕ^*, c^*) s'écrivent :

$$(A) \qquad \phi_i^* \left(\textstyle\sum_{e\in p_i} c_e - 1\right) = 0 \quad \forall i \in \{1,\dots,M\}$$
$$(B) \qquad c_e^* \left(\textstyle\sum_{i \text{ tel que } e\in p_i} \phi_i - u_e\right) = 0 \quad \forall e \in E$$

Pour des solutions entières, (A) et (B) signifient respectivement qu'il existe une seule arête de p_i dans la coupe lorsque $\phi_i^* > 0$, et que si l'arête e n'est pas saturée alors nécessairement $c_e^* = 0$. Ces deux programmes linéaires et ces conditions vont nous être très utiles dans les sections suivantes dans le cas de graphes particuliers pour élaborer et prouver la validité de nos algorithmes de résolution de IMFP et MCP.

3.2 Anneaux

Le cas particulier des anneaux que nous étudions dans cette section est important en pratique, puisqu'il est utilisé par exemple dans le domaine des télécommunications (voir [35]). Considérons

un anneau $\mathcal{A} = (V, E)$ non orienté dont chaque arête e est valuée par un entier naturel u_e. Ici chaque flot peut être divisé et suivre deux chemins possibles, alors que dans le cas orienté, une seule possibilité de routage est permise. Cependant, il est facile de vérifier que l'on peut toujours se ramener au cas orienté en doublant le nombre de terminaux : à chaque paire $\{s_k, t_k\}$, $k \in \{1, \ldots, K\}$, on associe $\{s_{k+K}, t_{k+K}\}$ où s_{k+K} (resp. t_{k+K}) est placée au même sommet que t_k (resp. s_k). C'est pourquoi on supposera pour la suite que $\mathcal{A}$ est orienté.

Dans un anneau, les programmes $(P - IMFP)$ et $(P - MCP)$ sont de taille raisonnable puisqu'ici $M = K < n =\mid V \mid$. En effet, plusieurs simplifications et réductions sont possibles. Premièrement, un chemin peut être réduit à un seul arc (celui de plus faible capacité) s'il contient des terminaux uniquement à ses extrémités. Deuxièmement, deux arcs adjacents (u, v) et (v, w) peuvent être remplacés par un seul si il existe uniquement une source en u ou v. Troisièmement, toute paire $\{s_k, t_k\}$ telle que le chemin associé contient celui d'une autre paire peut être supprimé (son flot peut être routé sur le chemin plus court, et il également sera "coupé" par les arcs supprimés pour le plus court). Ces réductions peuvent être effectuées en $O(K, n)$ (voir [16] dans l'annexe pour les détails). Malheureusement, il peut exister un écart entre les solutions optimales de IMFP et MCP dans les anneaux. Prenons par exemple $V = \{v_1, v_2, v_3\}$, trois arcs de capacité égale à 5, $s_1 = t_2 = v_1$, $s_2 = t_3 = v_2$ et $s_3 = t_1 = v_3$. Les valeurs optimales de MCP et IMFP sont respectivement 10 et 7. En revanche, nous avons montré que le saut d'intégrité entre $(P-IMFP)$ et sa relaxation continue est plus petit que 1, ce qui constitue un corollaire du résultat suivant :

Théorème 2 [16] *IMFP peut être résolu en temps polynomial dans les anneaux en effectuant une recherche binaire sur la valeur de* $\sum_{i=1}^{K} f_i \leq K u_{max}$*, où* u_{max} *est la capacité maximum dans l'anneau.*

Dans $(P - IMFP)$, dans chaque contrainte de (1) soit les coefficients égaux à "1" sont consécutifs ou soit ceux valant "0" le sont (grâce à la structure en anneau). On ajoute alors la contrainte supplémentaire (Λ_0) : $\sum_{i=1}^{K} f_i = F$ pour F constante, afin d'obtenir un problème de décision. Puis on remplace chaque contrainte (Λ') dans laquelle les "1" ne sont pas consécutifs par $(\Lambda_0) - (\Lambda)$, ce qui conduit à une matrice totalement unimodulaire.

Pour le problème MCP, nous avons montré qu'une démarche purement combinatoire est possible :

Théorème 3 [16] *MCP peut être résolu en* $O\left(n^2\right)$ *dans les anneaux (une fois simplifiés et réduits, ces procédures s'exécutant en* $O\left(K + n\right)$*).*

Ici, nous avons utilisé le fait que MCP peut être résolu dans un chemin en $O(K + n)$ [66], et donc en $O(n)$ ici après les réductions. Puisque chaque chemin p_k correspondant à $\{s_k, t_k\}$ de $\mathcal{A}$ doit être coupé, l'idée est simplement d'appliquer cet algorithme à tous les chemins obtenus en supprimant successivement chaque arc d'un chemin quelconque p_{k_0} (si possible le plus court), et

finalement de conserver la meilleure solution.

Nous avons également proposé [16] des algorithmes plus efficaces, en $O(n)$, pour le cas particulier où les capacités de l'anneau sont toutes égales. On parle alors d'anneau "uniforme". On peut supposer dans ce cas qu'il existe exactement une source et un puits en chaque sommet du graphe en utilisant les réductions rappelées au début de cette section. On a donc $K = n$ et tous les chemins correspondants à $\{s_k, t_k\}$ sont de même longueur L. Il y a donc L flots passant sur chaque arc. Des solutions optimales des relaxations continues de $(P - IMFP)$ et $(P - MCP)$ peuvent être déterminées trivialement puisque :

Lemme 1 [16] *$f_k = \frac{U}{L}$ pour tout k dans $\{1, \ldots, n\}$ et $c_e = \frac{1}{L}$ pour tout $e \in E$ sont des solutions optimales respectives des relaxations continues de $(P - IMFP)$ et de $(P - MCP)$.*

Le résultat est immédiat en utilisant la dualité : ces deux solutions sont admissibles et leur valeur commune vaut $\frac{nU}{L}$. On en déduit alors aisément une solution optimale de MCP, car contenant $\lceil \frac{n}{L} \rceil$ arcs :

Théorème 4 [16] *MCP peut être résolu en $O(n)$ dans les anneaux uniformes. Une solution optimale est : pour $j \in \{1, \ldots, n\}$, $c_{e_j} = 1$ si $j \in \{1 + pL, p \in \{0, \ldots, \lceil \frac{n}{L} \rceil - 1\}\}$, et $c_{e_j} = 0$ sinon.*

Pour IMFP, l'idée est semblable : on construit une solution $\hat{f}$ admissible de valeur $\lfloor \frac{nU}{L} \rfloor$ et donc optimale. La complexité de l'algorithme est également $O(n)$, mais ce dernier et surtout la preuve de sa validité sont un peu plus compliqués. L'idée est d'arrondir successivement supérieurement ou inférieurement les valeurs des n composantes du multiflot optimal continu du Lemme 1 de façon à garantir à chaque étape j l'admissibilité et un écart inférieur à $(j+1)\frac{U}{L}$ entre les valeurs des solutions entières et continues partielles.

Algorithme 1 Multiflot entier optimal dans les anneaux uniformes ;

entrées : Un anneau uniforme orienté $\mathcal{R} = (V, E)$, $\mid V \mid = n$, U : capacité commune des arcs, $\mathfrak{L} = \{\{s_1, t_1\}, \ldots, \{s_n, t_n\}\}$ dont les chemins associés sont tous de même longueur L.

sortie : Un multiflot entier $\hat{f} = \left\{\hat{f}_j\right\}_{j \in \{1,\ldots,n\}}$ de valeur $\lfloor \frac{nU}{L} \rfloor$.

$\hat{f}_1 = \lfloor \frac{U}{L} \rfloor$;
pour $j = 1$ **à** $n - 1$ **faire**
  Si $j\frac{U}{L} - \lfloor j\frac{U}{L} \rfloor \geq \lceil \frac{U}{L} \rceil - \frac{U}{L}$ **alors** $\hat{f}_{j+1} = \lceil \frac{U}{L} \rceil$;
  sinon $\hat{f}_{j+1} = \lfloor \frac{U}{L} \rfloor$;
  FinSi
Fait

On démontre tout d'abord par récurrence sur $1 \leq k \leq n$ que $\sum_{j=1}^{k} \hat{f}_j = \lfloor k\frac{U}{L} \rfloor$, puis que les contraintes de capacité sont satisfaites à chaque étape de l'algorithme (voir [16] dans l'annexe pour le détail des démonstrations). Ce qui conduit au résultat suivant :

Théorème 5 [16] *IMFP peut être résolu en $O(n)$ dans les anneaux uniformes.*

Bien que les programmes linéaires aient disparu des algorithmes finaux pour le cas d'un anneau uniforme, c'est grâce à eux que nous sommes parvenus à construire des solutions optimales pour IMFP et MCP.

3.3 Arborescences

Le deuxième cas particulier de graphe que nous allons considérer est celui d'une arborescence $\mathcal{T} = (V, A)$. Dans le cas d'un arbre orienté les matrices des contraintes des programmes $(P - IMFP)$ et $(P - MCP)$ sont totalement unimodulaires. IMFP et MCP sont donc polynomiaux dans ce cas. Mais plutôt que de résoudre des programmes linéaires continus, nous avons utilisé dans [37] les conditions des écarts complémentaires (A) et (B) données dans l'introduction de ce chapitre (section 3.1) pour construire des algorithmes combinatoires pour nos deux problèmes.

Pour résoudre IMFP (algorithme 2), après avoir numéroté les sommets de $\mathcal{T}$ et les flots (étapes 1. et 2.), on part des feuilles de l'arborescence puis on remonte jusqu'à la racine en routant les flots dans l'ordre des sources rencontrées en saturant à chaque fois au moins un arc si le flot est non nul (étape 4.). Ainsi, on satisfait la condition (A) des écarts complémentaires.

Algorithme 2 Multiflot entier dans une arborescence ;

entrées : Une arborescence $\mathcal{T} = (V, A)$, $u \in \mathbb{N}^{|A|}$, $\mathfrak{L} = \{\{s_1, t_1\}, \ldots, \{s_K, t_K\}\}$

sorties : $(f_1^*, ..., f_k^*, ..., f_K^*)$ un multiflot entier maximal. C_0, l'ensemble des arcs saturés par le multiflot.

1. Numéroter les sommets de $\mathcal{T}$ dans un ordre lexicographique en largeur d'abord ;
2. Numéroter les flots en partant des sources depuis la racine jusqu'aux feuilles ;
3. $C_0 \leftarrow \emptyset$; $\quad$ [*si* $K = O(n^2)$ *alors ajouter :* $L \leftarrow \{f_K, f_{K-1}, ..., f_1\}$; $k \leftarrow K$]
4. Pour $k = K$ **à** 1 **faire** $\quad$ [*si* $K = O(n^2)$ *alors remplacer par* **tant que** $L \neq \emptyset$ **faire**]

Router la quantité maximale f_k^* de s_k à t_k en respectant les capacités résiduelles courantes ;
$E_k \leftarrow$ {nouveaux arcs saturés par f_k^*} ; $\quad$ [*si* $K = O(n^2)$ *alors ajouter :* $L \leftarrow L - \{f_k\}$]
$C_0 \leftarrow C_0 \bigcup E_k$;
[*si* $K = O(n^2)$ *alors ajouter :* **pour** f_i **tel que** $p_i \cap E_k \neq \emptyset$ **faire** $f_i^* \leftarrow 0$; $L \leftarrow L - \{f_i\}$ **Fait**]
Fait ;

La solution f^* renvoyée par l'algorithme est bien un multiflot entier, puisque nous avons supposé que la capacité u_a de chaque arc a est entière. Cet algorithme peut être implémenté en $O(\min(Kn, n^2))$ en traitant spécifiquement le cas où $K = O(n^2)$ pour lequel nous avons utilisé une structure de données plus sophistiquée (voir [37] donnée dans l'annexe). Les ajouts à effectuer dans ce cas sont donnés dans l'algorithme entre crochets.

Pour résoudre MCP (algorithme 3), nous utilisons également la condition (B) des écarts complémentaires. Nous partons de l'ensemble C_0 des arcs saturés par l'algorithme précédent pour sélectionner ceux qui feront partie de la multicoupe. On choisit un seul arc par chemin lorsque le flot routé correspondant est non nul (condition (A)). Les numérotations données par l'algorithme pour IMFP sont conservées, mais cette fois on redescend de la racine de $\mathcal{T}$ aux feuilles. Ce deuxième algorithme peut également être implémenté en $O\left(\min(Kn,\, n^2)\right)$. Puisque nous devons exécuter l'algorithme 2 pour obtenir l'ensemble C_0, la complexité reste la même.

Algorithme 3 Multicoupe dans les arborescences ;

entrées : Une arborescence $\mathcal{T} = (V, A)$, $u \in \mathbb{N}^{|E|}$, les chemins $\{p_1, \ldots, p_K\}$ associés à $\mathfrak{L} = \{\{s_1, t_1\}, \ldots, \{s_K, t_K\}\}$, f^*, C_0=$\{$arcs saturés par le multiflot $f^*\}$

sortie : Une multicoupe minimale $\widehat{C}$.

pour $k = 1$ **à** K **faire**
  si $f_k^* > 0$ **et** $|C_{k-1} \bigcap p_k| > 1$ **alors**
    $a_k \leftarrow \{$premier arc sur le chemin de s_k à $t_k\}$;
    $C_k \leftarrow [C_{k-1} - (C_{k-1} \bigcap p_k)] \bigcup \{a_k\}$;
    //$C_{k-1} \bigcap p_k$ est un ensemble d'arcs de capacité résiduelle nulle
    //On supprime tous les arcs de p_k dans C_{k-1} sauf a_k
  sinon
    //Soit $f_k^* > 0$ et il y 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
    $C_k \leftarrow C_{k-1}$;
  FinSi ;
Fait ;
$\widehat{C} \leftarrow C_K$;

On montre que :

Lemme 2 [37] *L'ensemble $\widehat{C} \subseteq E$ renvoyé par l'algorithme 3 contient au moins un arc sur chaque chemin p_k allant de s_k à t_k, pour tout k dans $\{1, .., K\}$: $\widehat{C}$ est une multicoupe.*

Finalement, on prouve le résultat suivant par dualité, en établissant que les deux solutions entières produites par nos deux algorithmes satisfont les conditions des écarts complémentaires (A) et (B) des relaxations continues des programmes linéaires $(P - IMFP)$ et $(P - MCP)$. Nous renvoyons le lecteur à [16] donnée dans l'annexe pour les détails de la preuve.

Théorème 6 [37] *Soient f^* et $\widehat{C}$ les solutions renvoyées par les algorithmes 2 et 3. On a $\sum_{a \in \widehat{C}} u_a = \sum_{k=1}^{K} f_k^*$, et donc f^* est un multiflot entier maximal et $\widehat{C}$ est une multicoupe optimale.*

Une nouvelle fois, nous avons obtenu deux algorithmes purement combinatoires en utilisant la programmation linéaire continue. Ici, l'égalité entre les valeurs optimales des problèmes IMFP et MCP a permis d'utiliser directement les conditions des écarts complémentaires. Ce sont ces

dernières et la dualité qui ont guidé l'élaboration des deux algorithmes et qui nous ont permis de prouver leur validité.

3.4 Grilles

Nous présentons dans cette section quelques résultats de [17] (donnée dans l'annexe) concernant les grilles $\mathcal{G} = (V, E)$ non orientées. Ces travaux ont été menés dans le cadre du stage de DEA de Cédric Bentz et de sa thèse au laboratoire CEDRIC du CNAM. Une grille est un graphe pouvant être représenté comme une grille rectangulaire possédant m lignes (numérotées de haut en bas) et n colonnes (numérotées de gauche à droite). Pour pouvoir définir IMFP et MCP, on associe à chaque arête e de $\mathcal{G}$ un entier naturel u_e, et on se donne une liste $\mathfrak{L}$ de K paires de terminaux ou *liaisons* $\{s_k, t_k\}$, $k \in \{1, \ldots, K\}$, placés aux sommets de $\mathcal{G}$. Par abus de langage on parlera de sommet terminal lorsqu'au moins un terminal sera placé sur celui-ci. Enfin, comme précédemment pour les anneaux, on parlera de grille uniforme si toutes les arêtes ont la même capacité c.

MCP est malheureusement $\mathcal{NP}$-difficile même dans les grilles uniformes lorsque les terminaux se trouvent tous sur la première ou la dernière ligne de la grille si on autorise plusieurs terminaux à être placés sur un même sommet (on utilise une réduction vers le problème classique de couverture des arêtes d'un graphe par les sommets [17] : Vertex-Cover). Il faut donc faire des hypothèses réductrices supplémentaires pour espérer obtenir des algorithmes polynomiaux. Un cas intéressant dans la pratique (en conception de circuits VLSI par exemple) est celui de grille *bilatérale*. Ici les terminaux se situent soient sur la première ou la dernière ligne de $\mathcal{G}$, et sont distincts deux à deux. Dans ce qui suit, on appellera *bords horizontaux* ces deux lignes particulières.

A. Frank a étudié dans [55] et [76] le problème EDP (Edge Disjoint Paths) de l'existence de chemins (reliant chaque s_k à t_k) disjoints par les arêtes dans les grilles bilatérales. Si on relie dans une grille bilatérale chaque terminal par une arête pondérée de capacité c au sommet de $\mathcal{G}$ où il se trouvait précédemment (rappelons que deux terminaux ne peuvent pas être reliés au même sommet), on obtient une grille bilatérale *augmentée*. Dans ce cas, MaxEDP, le problème d'optimisation associé à EDP, i.e. la maximisation du nombre de chemins disjoints par les arêtes peut être vu comme le cas particulier de IMFP où toutes les capacités valent 1. Une notion fondamentale intervenant dans ce type de graphe est celle la *densité* d'une bande verticale (région comprise entre deux colonnes consécutives) de la grille. Elle correspond aux nombres de liaisons $\{s_k, t_k\}$ qui la "traversent" (i.e. tout chemin allant de s_k à t_k traverse nécessairement cette bande). Une bande verticale sera dite *saturée* si sa densité vaut m. On peut alors définir la densité d du couple $(\mathcal{G}, \mathfrak{L})$ comme étant le maximum des densités des bandes de la grille $\mathcal{G}$. Le

théorème suivant (dû à A. Frank [55, 76]) donne les conditions nécessaires et suffisantes pour le problème de décision EDP dans une grille bilatérale :

Théorème 7 [A. Frank] *Si $\mathcal{G}$ est une grille bilatérale possédant m lignes et de densité d, et si $\mathfrak{L}$ est telle qu'au moins une liaison ne soit pas verticale, alors toutes les liaisons définies par $\mathfrak{L}$ peuvent être reliées par des chemins disjoints par les arêtes si et seulement si :*

(1) *$m > d$ et il y a au moins un sommet non terminal sur un bord horizontal,*

ou **(2)** *$m \geq d$ et :*

(2.a) *soit il existe une liaison telle que les deux terminaux sont sur le même bord horizontal.*

(2.b) *soit il existe un sommet non terminal situé sur un bord horizontal qui est : soit à gauche de la bande verticale saturée la plus à gauche, soit à droite de la bande verticale saturée la plus à droite.*

(2.c) *soit il existe deux sommets non terminaux du même bord horizontal qui ne sont pas séparés par une bande verticale saturée.*

Remarquons que pour satisfaire la condition 2.b, il suffit de supprimer une liaison dont l'un des terminaux est situé dans un coin de la grille. En appliquant ce théorème à MaxEDP dans une grille bilatérale augmentée (ici cas particulier de IMFP lorsque toutes les capacités valent 1), on obtient :

Proposition 1 *Soit Opt(MaxEDP) la valeur optimale de MaxEDP. Si $(G, \mathfrak{L})$ est tel que $m \geq d$ alors :*

- *Opt(*MaxEDP*) $= K$ si $m > d$ et s'il existe un sommet non terminal sur un bord horizontal ;*
- *Opt(*MaxEDP*) $= K$ si* 2.a*,* 2.b *ou* 2.c *sont satisfaites ;*
- *Opt(*MaxEDP*) $= K - 1$ sinon.*

Il reste donc le cas $m < d$ à régler. L'idée est de rendre nuls un certain nombre de flots (rappelons qu'ici ils valent tous au plus 1) afin de satisfaire la condition (2) du théorème 7 (contrainte (C) dans (INP)). La programmation linéaire va encore être utile pour formaliser ce problème :

$$(INP) \begin{cases} \max \sum_{i=1}^{K} \phi_i & \\ \text{s.c.} \sum_{i \text{ tel que } \{s_i,t_i\} \text{ doit traverser } v_j} \phi_i \leq m & \forall j \in \{1,\dots,n-1\} \quad (C) \\ \phi_i \in \{0,1\} & \forall i \in \{1,\dots,K\} \end{cases}$$

v_j est la $j^{\text{ème}}$ bande verticale (la numérotation commence à gauche). La matrice de (INP) étant totalement unimodulaire (les bandes traversées par chaque liaison sont consécutives), on peut donc obtenir une solution $\hat{\phi}$ en 0-1 en temps polynomial (en fait on peut même l'obtenir par un algorithme combinatoire [4]). De plus, puisqu'au moins une des contraintes de (INP) sera saturée, on aura $d = m$ si on ne conserve que les flots non nuls de $\hat{\phi}$ dans $\mathfrak{L}$. Ainsi la condition (2)

du théorème 7 est satisfaite. De plus, la condition (2.b) est soit déjà vérifiée soit elle peut l'être en annulant un flot supplémentaire. L'écart entre les valeurs optimales de MaxEDP et de (INP), notée N^*, vaut donc au plus 1. Dans le cas où m est impair, la condition (2.b) du théorème 7 est toujours satisfaite si $m = d$. On obtient donc que la valeur optimale de MaxEDP vaut celle de (INP). Sinon, dans le cas général, on prouve que :

Théorème 8 [17] *Si $(G, \mathfrak{L})$ satisfait $m < d$, alors on peut déterminer une solution optimale de MaxEDP en résolvant $\mathrm{O}(n^2)$ programmes linéaires continus.*

L'idée est de tester si l'instance vérifie les conditions 2.a, 2.b ou 2.c du théorème 7 à l'aide de programmes linéaires construits à partir de (INP) (voir les détails de la preuve dans [17]).

Ces résultats permettent de résoudre le problème plus général IMFP (i.e. le cas où $c \geq 2$) dans le cas des grilles bilatérales augmentées uniformes. On a :

Théorème 9 [17] *Soit $c \geq 2$. Si la condition 2.a n'est pas satisfaite, c est impair, $K = n$ et $d \leq m < \lceil \frac{dc}{c-1} \rceil$, alors Opt*(IMFP) $= Kc - 1$ *; sinon Opt*(IMFP) $=$*Opt*(INP)$\times c$.

La preuve, assez longue et technique, est donnée dans [17] en annexe de ce document. Il est cependant intéressant de constater que la résolution de ce dernier problème requiert seulement la résolution d'un seul programme linéaire continu (ce résultat est à comparer avec celui du theorème 8 pour MaxEDP). De plus, la valeur optimale de IMFP pour $c \geq 2$ vaut Opt*(INP)*$\times c$ si $m < d$ alors que celle de MaxEDP n'est pas toujours égale à Opt*(INP)* dans ce cas.

La grille étant uniforme, c peut être supposé égal à 1 pour le problème MCP (cela ne modifie pas les solutions du problème). Pour résoudre MCP la dualité va une nouvelle fois être utilisée en considérant (CD), le programme dual de la relaxation continue de (INP) (rappel : v_j est la $j^{\text{ème}}$ bande verticale) :

$$(CD) \begin{cases} \max \sum_{k=1}^{K} w_k + m \sum_{j=1}^{n-1} y_j & \\ \text{s.c.} \quad w_k + \sum_{j \text{ tel que } \{s_k,t_k\} \text{ doit traverser } v_j} y_j \geq 1 & \forall k \in \{1,\ldots,K\} \\ y_j \geq 0 & \forall j \in \{1,\ldots,n-1\} \\ w_k \geq 0 & \forall k \in \{1,\ldots,K\} \end{cases}$$

La matrice de (CD) étant totalement unimodulaire (celle de (INP) l'est), l'idée est de reconnaître une multicoupe lorsqu'on considère solution optimale entière de ce programme. Il suffit pour cela de réintroduire les contraintes 0-1 sur les variables y_j et w_i, et d'interpréter ces dernières comme des variables de décision. Les variables y_j sont ainsi associées aux arêtes de v_j, l'ensemble des arêtes de la $j^{\text{ème}}$ bande verticale de $\mathcal{G}$, et w_i à l'arête reliant la source s_i à la grille bilatérale. Toutefois, il pourrait exister d'autres types de coupes pour MCP, et donc de meilleures solutions. La dualité permet d'écrire que la valeur optimale de la relaxation continue

de (INP) vaut celle de (CD). Or dans les grilles bilatérales augmentées uniformes avec $c = 1$ la valeur optimale du multiflot continu vaut également cette valeur (voir [17] dans l'annexe pour la preuve). Cela démontre l'optimalité de la multicoupe obtenue.

Pour conclure cette section, signalons que dans [17] (fourni dans l'annexe), deux algorithmes combinatoires sont présentés pour résoudre (INP) et (CD). Ici encore, les preuves de validité des algorithmes, qui peuvent être implémentés en $O(K)$, utilisent le théorème des écarts complémentaires.

3.5 Conclusion

Dans ce chapitre, nous avons présenté un ensemble d'algorithmes, souvent purement combinatoires, conçus à partir de programmes linéaires en nombres entiers et en variables 0-1 (ainsi que leurs relaxations continues). Les notions de dualité, de matrice totalement unimodulaire, l'étude des sauts d'intégrité éventuels, ainsi que le théorème des écarts complémentaires nous ont permis d'élaborer des méthodes efficaces de résolution.

Dans la partie suivante, nous allons aborder un autre cas de programmation convexe, plus général que la programmation linéaire, et donc permettant d'obtenir de meilleures relaxations : la programmation semidéfinie.

Relaxations Semidéfinies pour les programmes quadratiques

4

Introduction à la programmation semidéfinie

Dans ce chapitre introductif nous donnons succinctement quelques éléments de la programmation semidéfinie. Puis, nous rappelons les relaxations semidéfinies basiques des programmes quadratiques à variables bivalentes. Enfin, nous présentons les liens entre relaxations semidéfinie, linéaire, quadratique convexe et approche lagrangienne pour ces mêmes programmes. Dans les chapitres suivants, nous nous appuierons sur ces résultats pour proposer une interprétation lagrangienne des relaxations semidéfinies pour les programmes quadratiques contenant des contraintes linéaires, puis nous détaillerons une méthode systématique pour élaborer des relaxations semidéfinies à partir d'une relaxation linéaire donnée. Le lecteur est invité à consulter [101] dans l'annexe de ce document pour une introduction moins technique, les tours d'horizon présentés dans [80, 108, 109], et l'excellent ouvrage [110] pour une présentation complète de la programmation semidéfinie et de ses applications.

4.1 Programmation semidéfinie

L'ensemble des matrices réelles carrées symétriques est noté S_n. Les produits scalaire et tensoriel de deux vecteurs x, y de $\Re^n$ sont respectivement $x^T y = \sum_{i=1}^n x_i y_i$ et $\left(xy^T\right)_{ij} = x_i y_j$ pour tout i et j dans $\{1, \ldots, n\}$. Remarquons que $xx^T \in S_n$. $\mathbf{d}(A)$ est la diagonale d'une matrice A de S_n, et pour $x \in \Re^n$, $\mathbf{diag}(x)$ est la matrice diagonale telle que $\mathbf{d}(\mathbf{diag}(x)) = x$. Enfin, pour A et B dans S_n on utilise le produit scalaire standard :

$$A \bullet B = Tr\left(A^T B\right) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} B_{ij}$$

Toute forme quadratique $x^T Ax$ peut donc s'écrire $A \bullet xx^T$. Une matrice A de S_n est positive (resp. définie positive) si et seulement si pour tout z non nul dans $\Re^n$, on a $z^T Az \geq 0$ (resp. $z^T Az > 0$). On note $S_n^+ = \{A \in S_n : A \succcurlyeq 0\}$ et $S_n^{++} = \{A \in S_n : A \succ 0\}$. Rappelons qu' A est positive si et seulement si tout mineur symétrique de A est positif. Un mineur symétrique d'une

matrice A de S_n est le déterminant d'une sous-matrice de A obtenue en choisissant de 1 à n lignes et les colonnes de mêmes indices. Définissons à présent le programme mathématique suivant :

$$(SDP)\begin{cases}\text{Minimiser} & f(x) = c^T x \\ \text{Sous contrainte} & F(x) = \sum_{i=1}^{m} x_i A_i - A_0 \succcurlyeq 0\end{cases} \tag{4.1}$$

où c est un vecteur de $\Re^m$, et $A_0, A_1, ..., A_m$ sont des matrices linéairement indépendantes de S_n (ce qui n'est pas restrictif). Considérons $\aleph$ l'espace affine défini par le repère $\mathfrak{R} = (-A_0, A_1, ..., A_m)$. On impose donc à $F(x)$, de coordonnées $x \in \Re^m$ dans le repère $\mathfrak{R}$ et donc élément de $\aleph$, d'être une matrice positive. La contrainte $\sum_{i=1}^{m} x_i A_i - A_0 \succcurlyeq 0$ est appelée *inégalité matricielle linéaire*. Un programme semidéfini est un problème d'optimisation convexe. En effet, la fonction f est linéaire, et si $F(x) \succcurlyeq 0$ et $F(y) \succcurlyeq 0$ on a $F(\lambda x+(1-\lambda)y) = \lambda F(x)+(1-\lambda)F(y) \succcurlyeq 0$, pour tout $0 \leq \lambda \leq 1$. La programmation semidéfinie peut être vue comme un programme linéaire *semi-infini* particulier en ce sens que la contrainte $F(x) \succcurlyeq 0$ est équivalente à l'infinité de contraintes $\forall z \in \Re^n$, $z^T F(x) z \geq 0$. Le cône fermé S_n^+ étant son propre cône polaire, le programme dual de (SDP) est $\max_{Z \succcurlyeq 0} \min_{x \in \Re^n} L(x, Z)$ où la fonction de Lagrange est égale à $L(x, Z) = c^T x + Z \bullet (A_0 - \sum_{i=1}^{m} x_i A_i)$. $L(x, Z)$ admet un minimum (égal à $A_0 \bullet Z$) à condition que $A_i \bullet Z = c_i \ \forall i \in \{1, \ldots, m\}$ (de façon à annuler tous les termes en x_i) sinon on obtient $-\infty$. le programme dual de (SDP) est donc :

$$(DSDP)\begin{cases}\text{Maximiser} & A_0 \bullet Z \\ \text{Sous contraintes} & A_i \bullet Z = c_i \ \forall i \in \{1, \ldots, m\} \\ & Z \succcurlyeq 0\end{cases} \tag{4.2}$$

(DSDP) est aussi un programme semidéfini. En effet, on peut le reformuler en substituant à Z son expression $B_0 + \sum_{i=1}^{p} y_i B_i$ dans un repère quelconque $(B_0, B_1, ..., B_p)$ de l'espace affine défini par les égalités $A_i \bullet Z = c_i$ pour tout i dans $\{1, \ldots, m\}$:

$$(DSDP) \Leftrightarrow \begin{cases}\text{Minimiser} & \sum_{i=1}^{m} (-A_0 \bullet B_i)\, y - A_0 \bullet B_0 \\ \text{Sous contrainte} & B_0 + \sum_{i=1}^{p} y_i B_i \succcurlyeq 0\end{cases} \tag{4.3}$$

Les résultats concernant la dualité en programmation semidéfinie sont plus faibles qu'en programmation linéaire en ce sens qu'il peut exister un saut de dualité borné entre un (SDP) et son dual. L'absence de saut de dualité et l'existence de solutions optimales peuvent être obtenus au prix d'une hypothèse supplémentaire : l'existence de points strictement réalisables pour le dual et le primal. Le saut de dualité vaut $c^T x - A_0 \bullet Z = \sum_{i=1}^{m} (A_i \bullet Z)\, x_i - A_0 \bullet Z = F(x) \bullet Z$. Les matrices $F(x)$ et Z étant positives, leur produit scalaire l'est également, et il est nul si et seulement si le produit matriciel $ZF(x)$ est nul. Il s'agit d'une généralisation des *conditions des écarts complémentaires* de la programmation linéaire. En l'absence de saut de dualité, c'est une condition nécessaire et suffisante d'optimalité pour un couple (x, Z) de solutions respectivement admissibles du primal et du dual.

Comparons à présent la programmation semidéfinie avec d'autres cas particuliers de programmes convexes. Un programme linéaire (PL) $\left\{\min_{x\in\Re^m} c^T x \text{ s.c. } Ax \geq b\right\}$, où $A \in \Re^{n\times m}$ et $b \in \Re^n$, est un programme semidéfini particulier où toutes les matrices A_i pour $i \in \{0,\ldots,m\}$ sont diagonales. Il suffit en effet de poser $F(x) = \mathbf{diag}(Ax - b)$. De plus, son dual (DPL) $\left\{\max_{y\in\Re^n} b^T y \text{ s.c. } A^T y = c\,;\, y \geq 0\right\}$ peut être déduit de 4.2 en constatant que dans ce cas $Z = \mathbf{diag}(y)$ est diagonale. La programmation quadratique convexe est également un cas particulier de la programmation semidéfinie. Considérons :

$$(Q)\begin{cases}\text{Minimiser} & f_0(x)\\ \text{Sous contraintes} & f_i(x) \leq 0\ \forall i \in \{1,\ldots,m\}\end{cases}$$

où $x \in \Re^n$, et pour tout $i \in \{0,\ldots,m\}$, $f_i(x)$ est une fonction quadratique convexe. $f_i(x) \leq 0$ peut donc s'écrire $x^T A_i^T A_i x + c_i^T x + d_i \leq 0$. Pour modéliser ces contraintes dans un programme semidéfini, nous allons utiliser le résultat suivant :

Théorème 10 *Si $A \in S_p^{++}$, $C \in S_n$, et B est une matrice réelle $p \times n$, alors* $\begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \succcurlyeq 0$ *est équivalent à $C - B^T A^{-1} B \succcurlyeq 0$.*

Démonstration. Le résultat est immédiat : $\begin{bmatrix} A & B \\ B^T & C \end{bmatrix} = \begin{bmatrix} I & 0 \\ B^T A^{-1} & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & C - B^T A^{-1} B \end{bmatrix} \begin{bmatrix} I & A^{-1}B \\ 0 & I \end{bmatrix}$

On en déduit que $f_i(x) \leq 0$ est équivalent à $\begin{bmatrix} I_n & A_i x \\ x^T A_i^T & -c_i^T x - d_i \end{bmatrix} \succcurlyeq 0$. (Q) peut ainsi être formulé comme un programme semidéfini en introduisant une variable t pour avoir une fonction objectif linéaire :

$$(Q) \Leftrightarrow \begin{cases}\text{Minimiser } t & \\ \text{s.c.} & \begin{bmatrix} I_n & A_0 x \\ x^T A_0^T & -c_0^T x - d_0 + t \end{bmatrix} \succcurlyeq 0 \\ & \begin{bmatrix} I_n & A_i x \\ x^T A_i^T & -c_i^T x - d_i \end{bmatrix} \succcurlyeq 0\ \forall i \in \{1,\ldots,m\}\end{cases}$$

En utilisant encore le Théorème 10 mais en prenant cette fois A=1 et $B = x^T$, on peut également formuler (Q) en introduisant plus de variables mais moins d'inégalités matricielles linéaires :

$$(Q) \Leftrightarrow \begin{cases}\text{Minimiser } t & \\ \text{s.c.} & \begin{bmatrix} d_0 - t & \frac{1}{2}c_0^T \\ \frac{1}{2}c_0 & A_0^T A_0 \end{bmatrix} \bullet \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \leq 0 \\ & \begin{bmatrix} d_i & \frac{1}{2}c_i^T \\ \frac{1}{2}c_i & A_i \end{bmatrix} \bullet \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \leq 0\ \ \forall i \in \{1,\ldots,m\} \\ & \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succcurlyeq 0\end{cases}$$

Cette approche permet également d'écrire un ensemble de contraintes linéaires $Ax-b=0$ sous la forme $(Ax-b)^2 = A^TA \bullet xx^T - 2b^TAx + b^Tb = 0$ puis de constater que $A^TA \bullet X - 2b^TAx + b^Tb = A^TA \bullet \left(X - xx^T\right) + (Ax-b)^2 = 0$ et $X - xx^T \succcurlyeq 0$ implique $Ax - b = 0$. On décrit donc l'ensemble des contraintes linéaires avec la contrainte de positivité et une seule contrainte linéaire. Ces derniers résultats montrent que la programmation semidéfinie représente une large classe de programmes mathématiques convexes, ce qui permet d'expliquer son emploi intensif ces dernières années comme relaxation de programmes quadratiques continus ou à variables entières.

4.2 Relaxation semidéfinie basique des programmes quadratiques en variables bivalentes

Soit un programme quadratique de variable $x \in \{0,1\}^n$. Nous formulons ce programme en utilisant la variable additionnelle $X = xx^T$:

$$(P\{0,1\}) \begin{cases} \text{Minimiser } A_0 \bullet X + b^Tx = x^TA_0x + b^Tx \\ \text{s.c. :} \quad A_i \bullet X + c_i^Tx = (\text{ou} \leq)\, d_i \qquad \forall i \in \{1,\ldots,m\} \\ \qquad\quad X = xx^T \\ \qquad\quad x \in \{0,1\}^n \end{cases}$$

Remarquons que si toutes les matrices A_i de S_n $i \in \{0,...,m\}$ sont nulles, alors $(P\{0,1\})$ est un programme linéaire en $0-1$. Une façon naturelle d'obtenir une relaxation semidéfinie de $(P\{0,1\})$ est de relâcher les contraintes d'intégrité sur x et la contrainte $X = xx^T$ en $X \succcurlyeq xx^T$ [92]. Pour garantir que les solutions obtenues vérifieront $0 \leq x_i \leq 1$, on impose en plus que $\mathbf{d}(X) = x$. En effet, $X \succcurlyeq xx^T$ est équivalent à $\begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succcurlyeq 0$ (Théorème 10), ce qui implique que $\det \begin{bmatrix} 1 & x_i \\ x_i & x_i \end{bmatrix} \geq 0$ pour tout i dans $\{1,\ldots,n\}$, et donc $0 \leq x_i^2 \leq x_i \leq 1$. La relaxation semidéfinie obtenue est :

$$(SDP\{0,1\}) \begin{cases} \text{Minimiser } A_0 \bullet X + b^Tx \\ \text{s.c. :} \quad A_i \bullet X + c_i^Tx = (\text{ou} \leq)\, d_i \qquad \forall i \in \{1,\ldots,m\} \\ \qquad\quad \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succcurlyeq 0 \quad \text{et} \quad \mathbf{d}(X) = x \\ \qquad\quad (X,x) \in S_n \times \Re^n \end{cases}$$

Une autre approche [60] consiste à considérer un modèle quadratique comportant n variables y_i avec $i \in \{1,...,n\}$ à valeurs dans $\{-1,1\}$. On effectue donc le changement de variables $x = \frac{1}{2}(y + e_n)$ dans $(P\{0,1\})$, où toutes les composantes de $e_n \in \Re^n$ valent 1. On rend totalement quadratique le programme en remplaçant chaque terme linéaire y_i par y_0y_i et on impose $y_0 = 1$. Cette égalité s'écrit matriciellement $\begin{pmatrix} 1 \\ x \end{pmatrix} = Q \begin{pmatrix} y_0 = 1 \\ y \end{pmatrix}$ où $Q = \begin{bmatrix} 1 & 0 \\ \frac{1}{2}e & \frac{1}{2}I_n \end{bmatrix}$. Par conséquent

on a $\begin{bmatrix} 1 & x^T \\ x & xx^T \end{bmatrix} = Qyy^TQ^T$. On effectue alors une relaxation en remplaçant chacune des variables y_i pour $i \in \{0,\dots,n\}$ par un vecteur unitaire v_i de dimension $n+1$. Enfin, on associe à $V = [v_0,\dots,v_n]$ sa matrice de Gram $Y = V^TV$. Remarquons que nécessairement $Y \succcurlyeq 0$ et $\mathbf{d}(Y) = e_{n+1}$. Réciproquement, le théorème de Cholesky nous permet d'associer un champ de vecteurs unitaires de dimension $n+1$ à toute matrice Y dans S_{n+1} telle que $Y \succcurlyeq 0$ et $\mathbf{d}(Y) = e_{n+1}$. Par conséquent, la relaxation semidéfinie obtenue est :

$$(SDP\{-1,1\}) \begin{cases} \text{Minimiser } Q^T \begin{bmatrix} 0 & \frac{1}{2}b^T \\ \frac{1}{2}b & A_0 \end{bmatrix} Q \bullet Y \\ \text{s.c. :} \quad Q^T \begin{bmatrix} 0 & \frac{1}{2}c_i^T \\ \frac{1}{2}c_i & A_i \end{bmatrix} Q \bullet Y = (\text{ou} \leq)\, d_i \ \forall i \in \{1,\dots,m\} \\ Y \succcurlyeq 0 \quad \text{et} \quad \mathbf{d}(Y) = e_{n+1} \\ Y \in S_{n+1} \end{cases}$$

$(SDP\{0,1\})$ et $(SDP\{-1,1\})$ sont équivalents. En effet, Q est inversible et $Q^{-1} = \begin{bmatrix} 1 & 0 \\ -e_n & 2I_n \end{bmatrix}$. Effectuons le changement de variable $\begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} = QYQ^T$ dans $(SDP\{0,1\})$. Y est donc bien positive. Pour toute matrice W dans S_{n+1}, on a :

$$\begin{aligned} W \bullet \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} &= Tr\left(Q^TW \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \left(Q^T\right)^{-1}\right) = Tr\left(Q^TWQQ^{-1} \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \left(Q^T\right)^{-1}\right) \\ &= Tr\left(Q^TWQQ^{-1}QYQ^T\left(Q^T\right)^{-1}\right) = Tr\left(Q^TWQY\right) = Q^TWQ \bullet Y \end{aligned}$$

Ce calcul est valable pour les fonctions à optimiser de $(SDP\{0,1\})$ et de $(SDP\{-1,1\})$ et également pour les contraintes linéaires de ces deux programmes.

4.3 Liens entre relaxations semidéfinies, linéaires, et approche lagrangienne

Comparons à présent les relaxations semidéfinies précédentes et la linéarisation classique [42, 54] de $(P\{0,1\})$ écrite sous forme matricielle :

$$(LP\{0,1\}) \begin{cases} \text{Minimiser } A_0 \bullet X + b^Tx \\ \text{s.c. :} \quad A_i \bullet X + c_i^Tx = (\text{ou} \leq)\, d_i \ \forall i \in \{1,\dots,m\} \\ \qquad 0 \leq X_{ij} \leq x_i \qquad \forall i < j \in \{1,\dots,n\} \\ \qquad 0 \leq X_{ij} \leq x_j \qquad \forall i < j \in \{1,\dots,n\} \\ \qquad x_i + x_j \leq 1 + X_{ij} \qquad \forall i < j \in \{1,\dots,n\} \\ \qquad \mathbf{d}(X) = x \\ \qquad (X,x) \in S_n \times [0,1]^n \end{cases}$$

On introduit habituellement uniquement les variables de linéarisation X_{ij} pour $i < j$, mais notre formulation rendra la comparaison avec l'approche semidéfinie plus aisée. Si nous ajoutons les contraintes de linéarisation à $(SDP\{0,1\})$ qui sont non redondantes [79], la seule différence entre ces deux programmes est la contrainte de positivité $X - xx^T \succcurlyeq 0$ (non-linéaire) dans $(SDP\{0,1\})$. En effet, rappelons que les contraintes de bornes $x \in [0,1]^n$ sont impliquées par $\begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succcurlyeq 0$ et $\mathbf{d}(X) = x$. On peut également renforcer $(SDP\{-1,1\})$ de la même manière en lui ajoutant les contraintes de linéarisation exprimées en variables $\{-1,1\}$. Le tableau 4.1 donne la correspondance entre les deux modèles (nous avons également ajouté les inégalités triangulaires générales). L'inégalité d'un modèle est obtenue en utilisant l'inégalité correspondante dans l'autre modèle et la matrice Q.

$SDP\{-1,1\}$	$SDP\{0,1\}$
$Y_{0i} + Y_{0j} + Y_{ij} \geq -1$	$X_{ij} \geq 0$
$Y_{ij} - Y_{i0} - Y_{j0} \geq -1$	$x_i + x_j \leq 1 + X_{ij}$
$Y_{i0} - Y_{j0} - Y_{ij} \geq -1$	$X_{ij} \leq x_i$
$-Y_{i0} + Y_{j0} - Y_{ij} \geq -1$	$X_{ij} \leq x_j$
$Y_{ij} + Y_{ik} + Y_{jk} \geq -1$	$X_{ij} + X_{ik} + X_{jk} - x_i - x_j - x_k \geq -1$
$Y_{ij} - Y_{ik} - Y_{jk} \geq -1$	$x_k + X_{ij} - X_{ik} - X_{jk} \geq 0$

Tab. 4.1. *Equivalence des contraintes. i, j, k sont pris distincts dans $\{1, \ldots, n\}$.*

Les relaxations semidéfinies basiques consistent donc simplement à ajouter une contrainte de positivité sur la matrice X constituée des variables de linéarisation. Nous verrons au chapitre 6 qu'il est possible d'élaborer de meilleures relaxations semidéfinies en considérant d'autres traitements des contraintes linéaires connus en programmation linéaire, mais qui profiteront en plus de l'effet de "levier" de la contrainte de positivité en SDP.

Il existe également une interprétation lagrangienne de ces relaxations semidéfinies. Nous rappelons uniquement ici un résultat qui nous sera utile dans le chapitre 5. Pour une présentation complète du sujet le lecteur est invité à consulter [83]. Remplaçons dans $(P\{0,1\})$ la contrainte d'intégrité $x \in \{0,1\}$ par les contraintes quadratiques équivalentes $x_i^2 = x_i$ pour tout i dans $\{1, \ldots, n\}$. Le programme dual du programme continu obtenu est alors :

$$(\mathrm{DT}) \sup_{\mu,\,\lambda} \Theta(\mu,\lambda) = \inf_{x \in \Re^n} x^T A(\mu,\lambda) x + c(\mu,\lambda)^T x - \lambda^T d$$

où $A(\mu,\lambda) = A_0 + \mathbf{diag}\,(\mu) + \sum_{i=1}^m \lambda_i A_i$ et $c(\mu,\lambda) = b - \mu + \sum_{i=1}^m \lambda_i c_i$. Nous avons omis les contraintes de signe sur les variables duales λ_i associées à des contraintes d'inégalité pour plus de lisibilité. Rappelons qu'une fonction quadratique quelconque $x^T Q x + e^T x + f$ définie sur $\Re^n$ admet un minimum si et seulement si $Q \succcurlyeq 0$ et $e \in Im(Q)$ (voir par exemple [83]). Par conséquent, $\Theta(\mu,\lambda)$, la valeur de la fonction duale de (DT) en (μ,λ), est finie si et seulement si

$A(\mu,\lambda) \succcurlyeq 0$ et $c(\mu,\lambda) \in Im\,(A(\mu,\lambda))$. De tels points existent, puisque, pour tout λ, en prenant μ suffisamment grand, $A(\mu,\lambda)$ sera définie positive (et donc $Im(A(\mu,\lambda)) = \Re^n$). Nous pouvons à présent rappeler le résultat suivant [83, 92, 103] :

Proposition 2 *(DT) peut se formuler comme le programme semidéfini suivant :*

$$(\text{SD}) \begin{cases} \sup\ r - \lambda^T d \\ \text{s.c. } F(r,\mu,\lambda) = \begin{bmatrix} -r & \frac{1}{2}c(\mu,\lambda)^T \\ \frac{1}{2}c(\mu,\lambda) & A(\mu,\lambda) \end{bmatrix} \succcurlyeq 0 \end{cases}$$

Démonstration. (r,μ,λ) est une solution admissible de (SD) si et seulement si pour tout $(\alpha, y) \in \Re \times \Re^n$ la forme quadratique $q(\alpha, y) = -\alpha^2 r + \alpha c(\mu,\lambda)^T y + y^T A(\mu,\lambda) y$ associée à la matrice $F(r,\mu,\lambda)$ est positive. Pour $\alpha = 0$ on retrouve la condition $A(\mu,\lambda) \succcurlyeq 0$. Sinon, en posant $x = \frac{1}{\alpha} y$, on obtient comme condition équivalente $q(1,x) = -r + c(\mu,\lambda)^T x + x^T A(\mu,\lambda) x \geq 0$ pour tout x dans $\Re^n$, ce qui implique $c(\mu,\lambda) \in Im\,(A(\mu,\lambda))$. Sinon on pourrait écrire $c(\mu,\lambda) = c_K(\mu,\lambda) + c_I(\mu,\lambda)$ avec $c_K(\mu,\lambda)$ non nul dans $\ker\,(A(\mu,\lambda))$ (cette matrice étant symétrique réelle), et on aurait $q(1, -t c_K(\mu,\lambda)) = -r - t c_K(\mu,\lambda)^2 < 0$ pour t réel suffisamment grand. Par conséquent (r,μ,λ) est une solution admissible de (SD) si et seulement si $\Theta(\mu,\lambda) \neq -\infty$ et $r \leq c(\mu,\lambda)^T x + x^T A(\mu,\lambda) x$ pour tout $x \in \Re^n$. Ainsi (SD) est bien équivalent à (DT). □

Remarquons que la région admissible de (SD) est d'intérieur non vide. Il suffit de prendre $\lambda = 0$, μ suffisamment grand, et r suffisamment négatif pour obtenir des solutions strictement réalisables. Par contre il n'existe pas forcément de solutions optimales. Cela suffit néanmoins pour garantir l'absence de saut de dualité entre (SD) et son dual semidéfini qui n'est autre que $(SDP\,\{0,1\})$. En effet, en notant $X' = \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix}$, μ est associé à la contrainte $\mathbf{d}(X) = x$, r à $X'_{11} = 1$, et λ aux contraintes $A_i \bullet X + c_i^T x = \begin{bmatrix} 0 & \frac{1}{2}c_i^T \\ \frac{1}{2}c_i & A_i \end{bmatrix} \bullet X' = (\text{ou } \leq)\, d_i$ pour $i \in \{1,...,m\}$. Par conséquent, en bidualisant la formulation en variables continues de $(P\,\{0,1\})$, on retrouve la relaxation semidéfinie $(SDP\,\{0,1\})$. Cette démarche qui fait apparaitre les relaxations semidéfinies basiques comme des instances particulières de la dualité lagrangienne est clairement présentée dans [83]. Dans le prochain chapitre, nous allons généraliser cette approche pour les programmes quadratiques quelconques en considérant une relaxation lagrangienne partielle du problème initial où les contraintes linéaires ne seront pas dualisées.

5

Construire une relaxation semidéfinie en utilisant une approche lagrangienne

5.1 Introduction

Un thème récurrent en optimisation continue et combinatoire est la convexification d'un programme mathématique, autrement dit la recherche d'une relaxation facile à résoudre. L'approche par dualité lagrangienne fournit un cadre général pour cette opération [61, 82]. La question consiste alors non seulement à déterminer quelle formulation du problème choisir, mais également quelles contraintes dualiser pour rendre le programme relâché facile à résoudre tout en obtenant la meilleure borne possible. On peut en effet considérer plusieurs relaxations lagrangiennes partielles obtenues à partir de différentes formulations du problème étudié. C'est pourquoi l'ajout de contraintes redondantes au programme initial et le choix de l'expression de la fonction à optimiser sont déterminants. Une autre question essentielle est la possibilité de formuler explicitement ou non le programme dual obtenu. Ce dernier point a évidemment des conséquences importantes dans la pratique car il a un incidence directe sur les méthodes de résolution numérique envisageables.

Dans ce chapitre, nous allons étudier un programme quadratique général (Pb) qui contient des contraintes linéaires d'égalité. Le programme $(P\{0,1\})$ défini au chapitre 4 est un cas particulier de (Pb), puisque comme nous l'avons déjà rappelé on peut modéliser le cas particulier d'un programme quadratique en variables $0-1$ en utilisant des contraintes $x_i^2 = x_i$ pour tout i dans $\{1,\ldots,n\}$.

$$\text{(Pb)} \quad \begin{cases} \min_{x\in\Re^n} & f(x) = x^T Q x + c^T x \\ \text{s.c.} & x^T B_i x + d_i^T x = (\text{ou } \leq)\, e_i \; i \in I = \{1,\ldots,m\} \\ & Ax = b \end{cases}$$

Les matrices réelles symétriques $n \times n$ Q et B_i pour tout i dans I sont quelconques. La matrice A définissant les contraintes linéaires du programme est de dimension $p \times n$, et est supposée de rang plein (ce qui n'est pas restrictif). Les vecteurs réels c et d_i pour tout i dans I sont de dimension n. Dans le chapitre 4, nous avons rappelé que bidualiser $(P\{0,1\})$ (en ayant

écrit les contraintes d'intégrité sous la forme de contraintes quadratiques) permettait d'obtenir la relaxation semidéfinie basique ($SDP\{0,1\}$). Ici, nous n'avons pas les contraintes $x_i^2 = x_i$ pour tout i dans $\{1,\ldots,n\}$, mais nous allons montrer qu'il est possible d'obtenir une relaxation semidéfinie équivalente à la valeur optimale d'une relaxation lagrangienne partielle de (Pb) où la contrainte $Ax = b$ n'est pas dualisée. De plus, nos résultats permettront d'obtenir une lecture unifiée de plusieurs relaxations proposées dans la littérature et d'en proposer de nouvelles dans la partie III. Dans le cas des programmes quadratiques en variables $0-1$, plusieurs travaux similaires existent [83, 92]. Nous allons donc généraliser ces résultats au cas du problème (Pb).

5.2 Convexifier dans l'approche lagrangienne

Pour μ dans $\Re^m$, définissons $Q(\mu) = Q + \sum_{i\in I}\mu_i B_i$, $c(\mu) = c + \sum_{i\in I}\mu_i d_i$ et $e(\mu) = \sum_{i\in I}\mu_i e_i$. (DT), la relaxation lagrangienne totale de (Pb), et (DP), la relaxation lagrangienne partielle où les contraintes linéaires d'égalité ne sont pas dualisées s'écrivent :

$$\text{(DT)}\ \sup_{\mu,\,\lambda}\inf_{x}\mathfrak{L}_{DT}(x,\mu,\lambda) = x^T Q(\mu)\,x + c^T(\mu)\,x - e(\mu) + \lambda^T(Ax-b)$$

$$\text{(DP)}\ \sup_{\mu}\ \inf_{x\ |\ Ax=b}\mathfrak{L}_{DP}(x,\mu) = x^T Q(\mu)\,x + c^T(\mu)\,x - e(\mu)$$

Pour plus de lisibilité, nous avons omis les contraintes de signe sur les μ_i correspondant à des contraintes d'inégalité dans (Pb). En suivant la démarche rappelée dans le chapitre 4 pour ($P\{0,1\}$), on peut montrer que (DT) est équivalent au dual du programme semidéfini [83] :

$$(SDP)_{DT}\begin{cases}\min\ Q\bullet X + c^T x\\ \text{s.c.}\ \ Ax = b\\ \qquad B_i\bullet X + d_i^T x = (\text{ou}\ \ \leq)\, e_i\ \ i\in I\\ \qquad \begin{bmatrix}1 & x^T\\ x & X\end{bmatrix}\succcurlyeq 0\end{cases}$$

Remarquons que la valeur optimale de (DP) est évidemment supérieure ou égale à celle de (DT). C'est pourquoi il serait intéressant de posséder également une telle formulation par SDP pour (DP). Pour μ fixé, les conditions nécessaires et suffisantes d'optimalité pour x^* dans $\Re^n$ pour le problème $\inf_{x\ |\ Ax=b}\mathfrak{L}_{DP}(x,\mu)$ sont [89] :

(C1) $\exists\lambda\in\Re^n$ tel que $2Q(\mu)x^* + c(\mu) + A^T\lambda = 0$ avec $x^*\in\Omega = \{x\in\Re^n :\ Ax = b\}$
(C2) $y^T Q(\mu)y \geq 0\ \forall y\in L = \ker(A)$

En examinant la condition **(C2)**, on constate qu'ajouter à (Pb) des contraintes quadratiques nulles sur Ω (et donc redondantes) revient à augmenter la dimension de μ et a par conséquent un impact sur (DT). En effet, cela peut rendre convexe la fonction $\mathfrak{L}_{DT}(x,\mu,\lambda)$. Notons que les

conséquences seront nulles sur (DP) puisque la contrainte $Ax = b$ est maintenue. Considérons en effet $\mathfrak{J} = \left\{ f_j(x) = x^T C_j x + q_j^T x + \alpha_j \ : \ j \in J \right\}$ un ensemble de fonctions quadratiques nulles sur Ω, et ajoutons les contraintes redondantes $f_j(x) = 0 \ (\forall j \in J)$ dans (Pb) pour obtenir un problème équivalent $(\mathrm{Pb})_{\mathfrak{J}}$. Les relaxations lagrangiennes totales et partielles de $(\mathrm{Pb})_{\mathfrak{J}}$ sont :

$$(\mathrm{DT})_{\mathfrak{J}} \sup_{\mu,\, \omega,\, \lambda} \inf_{x} x^T Q(\mu)\, x + c^T(\mu)\, x - e(\mu) + \sum_{j \in J} \omega_j f_j(x) + \lambda^T (Ax - b)$$

$$(\mathrm{DP})_{\mathfrak{J}} \sup_{\mu,\, \omega} \inf_{x \ | \ Ax = b} x^T Q(\mu)\, x + c^T(\mu)\, x - e(\mu) + \sum_{j \in J} \omega_j f_j(x)$$

Pour tout ensemble $\mathfrak{J}$, $(\mathrm{DP})_{\mathfrak{J}}$ est (DP) puisque $f_j(x) = 0$ pour $x \in \Omega$ et $\forall j \in J$. La valeur optimale de $(\mathrm{DT})_{\mathfrak{J}}$ est donc inférieure ou égale à celle de (DP) (car dans $(\mathrm{DP})_{\mathfrak{J}}$ les contraintes $Ax = b$ sont maintenues). μ étant fixé, pour obtenir une valeur finie pour $(\mathrm{DP})_{\mathfrak{J}}$, nous devons avoir $Q(\mu) + \sum_{j \in J} \omega_j C_j \succcurlyeq 0$ sur $\ker(A)$. Nous avons montré que :

Proposition 3 [48] *Soit μ^* une solution optimale de (DP). S'il existe ω^* tel que $\sum_{j \in J} \omega_j^* f_j(x)$ convexifie $x^T Q(\mu^*)\, x + c^T(\mu^*)\, x$ sur $\Re^n$, alors la valeur optimale de $(DT)_{\mathfrak{J}}$ est égale à celle de (DP).*

Cela signifie que si nous ajoutons des contraintes quadratiques redondantes dans (Pb) qui peuvent convexifier $\mathfrak{L}_{DP}(x, \mu^*)$ sur $\Re^n$ alors nous pourrons obtenir la valeur optimale de (DP) en résolvant $(\mathrm{DT})_{\mathfrak{J}}$ la relaxation lagrangienne totale de $(\mathrm{Pb})_{\mathfrak{J}}$. De plus, cette relaxation est équivalente au dual du programme semidéfini $(\mathrm{SDP})_{\mathfrak{J}}$ défini ci-dessous qui est simplement $(SDP)_{DT}$ auquel on a ajouté les contraintes quadratiques de $\mathfrak{J}$ linéarisées à l'aide de X :

$$(SDP)_{\mathfrak{J}} \begin{cases} \min Q \bullet X + c^T x \\ \text{s.c.} \ \ Ax = b \\ \qquad B_i \bullet X + d_i^T x = (\text{ou } \ \leq)\, e_i \ \ i \in I \\ \qquad C_j \bullet X + q_j^T x + \alpha_j = 0 \qquad j \in J \\ \qquad \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succcurlyeq 0 \end{cases}$$

Il nous faut à présent trouver un ensemble $\mathfrak{J}$ qui satisfasse à coup sûr la condition de la proposition 3, i.e. qui puisse convexifier sur $\Re^n$ entier toute fonction quadratique convexe sur $\ker(A)$. Dans la section suivante, nous allons tout d'abord caractériser l'ensemble des fonctions quadratiques constantes sur $\Omega = \{x \in \Re^n : \ Ax = b\}$, puis nous en choisirons certaines qui nous permettront d'utiliser la proposition 3.

5.3 Ensemble des fonctions quadratiques constantes sur une variété affine

Soit $L = \{u \in \Re^n : Au = 0\} = \ker(A)$, et $F(x) = x^T Hx + g^T x$ une fonction quadratique constante sur Ω. On a $F(x+\lambda u) = F(x) = F(x) + \lambda^2 u^T Hu + 2\lambda u^T Hx + \lambda g^T u$ pour tout $x \in \Omega$, et $u \in L$. Ainsi, on peut en déduire des conditions nécessaires et suffisantes pour F :

$$\begin{aligned} u^T Hu = 0 \ \forall u \in L \qquad & [A] \\ 2u^T Hx + g^T u = 0 \ \forall u \in L \forall x \in \Omega \quad & [B] \end{aligned}$$

On détermine tout d'abord l'expression des matrices H qui satisfont la condition [A].

Lemme 3 [48] *$q(u) = u^T Hu$ est une forme quadratique nulle sur $L = \{u : Au = 0\}$ si et seulement si $q(u) = u^T \left(A^T W^T + WA\right) u$, où W est une matrice $n \times p$ réelle quelconque.*

L'idée de la preuve consiste à choisir une nouvelle base pour exprimer la forme quadratique. On choisit la matrice de passage $P = \left[\begin{array}{cc} A^T & B \end{array}\right]$, où les $n-p$ colonnes de B forment une base L. Remarquons que les p colonnes de A^T sont une base de $L^\perp$ (elles peuvent être supposées linéairement indépendantes). Voir la preuve complète dans [48] jointe en annexe. Il suffit à présent de remplacer H par son expression dans la condition [B] pour obtenir la formule générale de nos fonctions :

Théorème 11 [48] *$F(x) = x^T Hx + g^T x$ est une fonction quadratique constante sur $\{x : Ax = b\}$ si et seulement si $F(x) = x^T \left(A^T W^T + WA\right) x + \left(A^T \alpha - 2Wb\right)^T x$, où W est une matrice $n \times p$ et α un vecteur réel de dimension p quelconques.*

A l'aide de ce résultat, on peut obtenir facilement les différentes familles de contraintes redondantes utilisées dans la littérature. Il suffit de choisir des valeurs particulières pour W et α.

Pour commencer, prenons $W = \frac{1}{2}E^{ij}$ et $\alpha = 0$, où E^{ij} est la matrice contenant comme seul élément non nul $E^{ij}_{ij} = 1$. On obtient $F(x) = x_i a_j^T x - b_j x_i$ et donc pour tout x dans Ω, $F(x) = 0$. Nous reconnaissons la méthode consistant à multiplier chaque contrainte linéaire par chaque variable (voir par exemple [3, 18, 88]).

A présent, choisissons $W = \frac{1}{2}A^T V$, où V est une matrice réelle symétrique $p \times p$. Nous obtenons $A^T W^T + WA = A^T VA$, et donc $F(x) = x^T A^T VAx + x^T \left(A^T \alpha - A^T Vb\right)$. Pour $V = \frac{1}{2}\left(E^{ij} + E^{ji}\right)$ et $\alpha = Vb$, il vient $F(x) = x^T a_i a_j^T x$. Nous obtenons $x^T a_i a_j^T x = b_i b_j$ pour tout x dans Ω. Nous reconnaissons la méthode consistant à multiplier deux par deux les contraintes linéaires (voir par exemple [70, 100]).

Pour terminer, choisissons $V = I$ et $\alpha = -b$. Il vient $F(x) = x^T A^T Ax - 2x^T A^T b$ et donc $(Ax - b)^2 = 0$ pour tout x dans Ω. Il s'agit d'un terme de pénalité classique utilisé par exemple dans [83, 89, 91].

On peut bien sûr faire varier W and α pour obtenir toutes les autres fonctions constantes sur Ω. Notre objectif étant de satisfaire la condition de la proposition 3, nous allons à présent nous intéresser à un sous-ensemble particulier de ces fonctions.

5.4 Expression de la relaxation lagrangienne partielle comme programme semidéfini

Dans [83], les auteurs montrent que si M est une matrice définie positive sur $L = \ker(A)$, alors il existe une matrice V telle que $M + A^T VA \succcurlyeq 0$. Nous reconnaissons le cas où $W = \frac{1}{2}A^T V$ donné dans la section précédente. Mais le résultat n'est malheureusement plus vrai si M est seulement positive (voir un contre-exemple dans [48]). Nous avons montré que si une matrice M est positive (mais pas forcément définie) sur L, il est possible de trouver W telle que $M + A^T W^T + WA \succcurlyeq 0$ sur $\Re^n$. On a en effet :

Théorème 12 [48] *Soient A et Q des matrices symétriques réelles de tailles respectives $p \times n$ et $n \times n$. Si Q est positive sur $L = \ker(A)$ alors il existe une combinaison linéaire des fonctions quadratiques $q_{ij}(x) = x_i(a_j^T x - b_j)$ pour $i \in \{1, ..., n\}$ et $j \in \{1, ..., p\}$ qui convexifie la forme quadratique $x^T Qx$ sur $\Re^n$.*

Remarquons que les fonctions q_{ij} sont nulles sur $\Omega = \{x \in \Re^n : Ax = b\}$. La preuve complète du théorème 12 est donnée en annexe [48]. L'idée est d'écrire H sous la forme $H = U^T W^T + WU$ avec $UU^T = I$ et $U = S^T A$, où S est une matrice $p \times p$ inversible triangulaire supérieure (on suit donc la procédure d'orthogonalisation de Gram-Schmidt). On écrit H sous la forme $H = \sum_{i=1}^{p} H_i$ avec $H_i = u_i \omega_i^T + \omega_i u_i^T$, où ω_i et u_i pour $i \in \{1, ..., p\}$ sont les vecteurs égaux respectivement à la $i^{\text{ème}}$ colonne de W et à la $i^{\text{ème}}$ ligne de U. Puis on considère les sous-espaces vectoriels

$$L_i = \left\{ y = \sum_{j=i+1}^{p} z_j u_j + Bz \, : \, z_j \in \Re \; \forall j \in \{i+1, ..., p\} \, , \, z \in \Re^{n-p} \right\}$$

où les colonnes de B forment une base orthonormée de $\ker(A)$. En remarquant que $L_p = \ker(A)$ et que $L_0 = \Re^n$, le résultat est alors une conséquence du lemme technique suivant :

Lemme 4 *Soit $1 \leq i \leq p$ et Q une matrice symétrique réelle, si $Q + \sum_{j=i+1}^{p} H_j \succcurlyeq 0$ sur L_i alors il existe ω_i tel que $Q + \sum_{j=i+1}^{p} H_j + u_i \omega_i^T + \omega_i u_i^T \succcurlyeq 0$ sur L_{i-1}.*

Un aspect important dans la pratique de notre preuve est que cette dernière est constructive. Nous donnons en effet les formules pour calculer la matrice W, qui peut être obtenue en $O(pn^2)$.

Un exemple est donné en annexe [48].

L'ensemble $\mathfrak{P} = \left\{x_i a_j^T x - b_j x_i \,:\, \forall i \in \{1,...,n\}\ \forall j \in \{1,...,p\}\right\}$ convient donc pour remplir la condition de la proposition 3, et nous pouvons à présent affirmer que (DP) est équivalent au dual du programme semidéfini suivant :

$$(SDP)_{\mathfrak{P}} \begin{cases} \min Q \bullet X + c^T x \\ \text{s.c. } Ax = b \\ \quad B_i \bullet X + d_i^T x = (\text{ou } \leq)\, e_i \ i \in I \\ \quad \sum_{k=1}^{n} A_{jk} X_{ki} - b_j x_i = 0 \quad i \in \{1,...,n\}\,;\, j \in \{1,...,p\} \\ \quad \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succcurlyeq 0 \end{cases}$$

Dans [83], les auteurs utilisent l'ensemble $\mathfrak{C} = \left\{(Ax-b)^2\right\}$ dans le cas particulier où les variables sont booléennes (i.e. le problème $(P\,\{0,1\})$ du chapitre 4), et démontrent que dans ce cas particulier de (Pb) le supremum du dual lagrangien total $(\text{DT})_{\mathfrak{C}}$ est égal à la valeur de (DP). Dans le cas général, ce résultat ne tient plus : $(\text{DT})_{\mathfrak{C}}$ n'est pas toujours équivalent à (DP), alors que c'est toujours le cas en utilisant $(\text{DT})_{\mathfrak{P}}$ (voir les exemples dans [48], donné dans l'annexe). Le programme semidéfini obtenu dans le cas booléen est $(SDP)_{\mathfrak{C}}$ contenant la contrainte $\mathbf{d}(X) = x$ issue des contraintes quadratiques $x_i^2 = x_i$ pour tout i dans $\{1,\ldots,n\}$.

$$(SDP)_{\mathfrak{C}} \begin{cases} \min Q \bullet X + c^T x \\ \text{s.c. } (Ax = b) \\ \quad B_i \bullet X + d_i^T x = (\text{ou } \leq)\, e_i \quad i \in I \\ \quad A^T A \bullet X - 2b^T Ax + b^2 = 0 \\ \quad \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succcurlyeq 0 \end{cases}$$

Nous avons mis la contrainte $Ax = b$ entre parenthèses dans $(\text{SDP})_{\mathfrak{C}}$ car elle est redondante. En effet, $A^T A \succcurlyeq 0$ et $X - xx^T \succcurlyeq 0$ impliquent $A^T A \bullet \left(X - xx^T\right) \geq 0$, et donc $A^T A \bullet \left(X - xx^T\right) + (Ax-b)^2 = 0$ implique $Ax = b$.

5.5 Conclusion

En fait, nous allons démontrer dans le chapitre suivant que les programmes $(\text{SDP})_{\mathfrak{C}}$ et $(\text{SDP})_{\mathfrak{P}}$ sont équivalents. Cependant, puisque leurs programmes duaux ne le sont pas toujours (saut de dualité avec leur primal ou absence de solutions optimales), cela peut avoir de lourdes conséquences dans la pratique lors de l'utilisation de certains logiciels de résolution de SDP. La figure 5.1 résume les résultats obtenus.

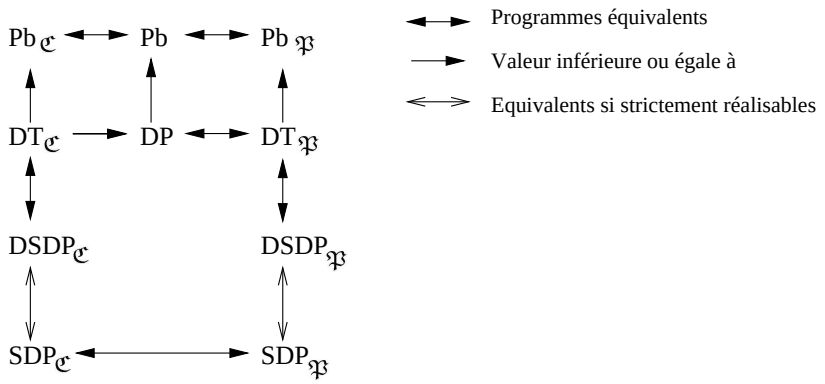

Fig. 5.1. *Relations entre les différentes formulations et relaxations de (Pb)*

Nous disposons à présent d'un cadre général permettant de formuler et de comparer facilement des relaxations semidéfinies. De plus, nous avons établi que la relaxation lagrangienne partielle de la formulation en variables continues de (Pb) où les contraintes linéaires $Ax = b$ ne sont pas dualisées (i.e. relâchées) constitue une limite pour les relaxations semidéfinies construites à l'aide de contraintes redondantes fondées sur $Ax = b$. Cela nous guidera dans la partie III de ce document lors de l'élaboration de nos relaxations semidéfinies.

6

Construire une relaxation semidéfinie en utilisant une relaxation linéaire existante

6.1 Introduction

Nous avons rappelé dans la section 4.1 les deux relaxations semidéfinies basiques pour un problème combinatoire pouvant se formuler comme un programme quadratique en variables bivalentes. Nous avions alors remarqué qu'étant donnée une relaxation linéaire standard, il est toujours possible de renforcer cette dernière en ajoutant la contrainte non-linéaire de positivité sur la matrice constituée des variables initiales et de linéarisation pour obtenir une relaxation semidéfinie. Nous allons voir à présent qu'il est possible de procéder à des traitements encore plus efficaces des contraintes linéaires, afin d'obtenir de meilleures relaxations. L'idée est de profiter de l'effet de "levier" de la contrainte de positivité. Précisons qu'ici notre propos n'est pas d'étudier une hiérarchie de relaxations de l'enveloppe convexe de la région admissible de notre programme (voir [78] pour une présentation et une comparaison des hiérarchies existantes), mais de construire mécaniquement des relaxations SDP utilisables en pratique en profitant de relaxations par programmation linéaire déjà proposées dans la littérature et ayant démontré leur efficacité (qualité de la borne et temps de résolution). L'application de cette démarche conduira donc éventuellement à des relaxations semidéfinies différentes puisqu'elles dépendent bien sûr du problème considéré mais aussi de la relaxation linéaire choisie.

Dans ce qui suit, nous allons considérer de nouveau le programme $(P\{0,1\})$ du chapitre 4 où nous avons mis à part les contraintes linéaires où X n'apparaît pas.

$$(P\{0,1\}) \begin{cases} \min & A_0 \bullet X + b^T x & & \\ \text{s.c. :} & A_i \bullet X + c_i^T x = (\text{ou} \leq)\, d_i & i \in \{1,...,m_1\} \\ & c_i^T x = d_i & i \in \{m_1+1,...,m_2\} \\ & d_i \leq c_i^T x \leq d_i' & i \in \{m_2+1,...,m\} \\ & X = xx^T & \\ & x \in \{0,1\}^n & \end{cases}$$

où $1 \leq m_1 \leq m_2 \leq m$. Rappelons que $(P\{0,1\})$ contient le cas particulier des programmes linéaires en $0-1$. Si certains d_i ou d'_i sont absents, nous pouvons utiliser $\sum_{i \text{ tel que } c_i>0} c_i$ pour d'_i et $\sum_{i \text{ tel que } c_i<0} c_i$ pour d_i (puisque $x \in \{0,1\}^n$). Nous avons rappelé dans le chapitre 4 que les régions admissibles de $(LP\{0,1\})$, la relaxation linéaire classique [42, 54] de $(P\{0,1\})$, et celle de (SDP)$\{0,1\}$ n'étaient pas comparables. D'où l'idée de construire une nouvelle relaxation SDP en partant d'une relaxation linéaire existante. Nous allons donc considérer chaque type de contrainte présente dans une relaxation linéaire donnée (P_L), et lui associer une ou plusieurs contraintes plus fortes ou équivalentes. Ainsi, nous obtiendrons un programme semidéfini qui sera par construction une meilleure relaxation que (P_L). Le principe est simplement de tenir compte des relaxations existantes en les améliorant de façon certaine. Dans la section suivante, nous allons étudier l'impact de l'ajout de la contrainte $X - xx^T \succcurlyeq 0$ et comparer dans le cadre de la programmation semidéfinie les traitements possibles des contraintes linéaires simples présentes dans (P_L) (i.e. où X n'intervient pas).

6.2 Contraintes linéaires d'égalité

Considérons une contrainte se présentant dans (P_L) (notre relaxation linéaire de départ) comme une égalité linéaire $c^T x = d$. Nous pouvons supposer que $d \geq 0$ (sinon on change les signes de chacun des membres). Dans $(SDP\{0,1\})$ (voir chapitre 4) nous avions simplement gardé la contrainte $c^T x = d$, mais nous pouvons également étudier les conséquences de l'ajout de différentes contraintes redondantes en $0-1$ utilisées en programmation linéaire en conjonction avec la contrainte $X - xx^T \succcurlyeq 0$.

Proposition 6.1. [100] *$cc^T \bullet xx^T = d^2$ est une contrainte valide pour $(P\{0,1\})$. Si $(X,x) \in S_n \times \Re^n$ est tel que $cc^T \bullet X = d^2$, $c^T x = d$ et $X - xx^T \succcurlyeq 0$ alors on a $cc^T \left(X - xx^T\right) = 0$ (contrainte non linéaire).*

Si $x \in \{0,1\}^n$ est tel que $\mathbf{d}(X) = x$ et $X - xx^T \succcurlyeq 0$ alors $X = xx^T$. Donc dans ce cas, $cc^T \bullet X = d^2$ est équivalent à $c^T x = d$ (puisque $d \geq 0$). D'autre part, la contrainte quadratique $cc^T \left(X - xx^T\right) = 0$ n'est pas satisfaite par toutes les solutions admissibles et non-entières de $(LP\{0,1\})$ défini dans le chapitre 4. Remarquons à présent que la contrainte $(c^T x - d)^2 = 0$ linéarisée en $cc^T \bullet X - 2dc^T x + d^2 = 0$ est un traitement équivalent puisque :

Proposition 6.2. [100] *On a $\left\{(X,x) \in S_n \times \Re^n \,;\, X - xx^T \succcurlyeq 0,\, cc^T \bullet X = d^2,\, c^T x = d\right\} = \left\{(X,x) \in S_n \times \Re^n \,;\, X - xx^T \succcurlyeq 0,\, cc^T \bullet X - 2dc^T x + d^2 = 0\right\}$.*

Nous pouvons également multiplier l'égalité $c^T x = d$ par x_i pour tout i dans $\{1,...,n\}$ pour obtenir les n contraintes "*produit*" $\sum_{j=1}^n c_j X_{ij} = dx_i$ déjà utilisées au chapitre 5, et introduites pour la programmation linéaire dans [3].

Proposition 6.3. [100] *Les contraintes $\sum_{j=1}^n c_j X_{ij} = dx_i \, \forall i \in \{1,...,n\}$ sont des inégalités valides pour $(P\{0,1\})$. Si $(X,x) \in S_n \times \Re^n$ est tel que $\sum_{j=1}^n c_j X_{ij} = dx_i \, \forall i \in \{1,...,n\}$ et $c^T x = d$ alors on a $cc^T \bullet X = d^2$.*

La proposition suivante montre que la réciproque est vraie en programmation semidéfinie grâce à la contrainte $X \succcurlyeq xx^T$, alors qu'elle est fausse en général en programmation linéaire. Ceci est moins évident intuitivement puisque le nombre de contraintes linéaires vérifiées passe de 2 à $n+1$.

Proposition 6.4. *Soit $(X,x) \in S_n \times \Re^n$ vérifiant $cc^T \bullet X = d^2$, $c^T x = d$ et $X - xx^T \succcurlyeq 0$, alors (X,x) vérifie $\sum_{j=1}^n c_j X_{ij} = dx_i \, \forall i \in \{1,...,n\}$*

Démonstration. On a $cc^T \bullet X = d^2$, donc $cc^T \bullet \left(X - xx^T\right) + \left(c^T x\right)^2 = d^2$ ce qui implique $cc^T \bullet \left(X - xx^T\right) = 0$ (puisque $c^T x = d$). Or $cc^T \succcurlyeq 0$ et $X - xx^T \succcurlyeq 0$, donc leur produit scalaire est nul si est seulement si leur produit (matriciel) est nul, ce qui s'écrit :

$$\begin{bmatrix} c_1^2 & \cdots & c_1c_j & \cdots & c_1c_n \\ \vdots & \ddots & & & \vdots \\ c_kc_1 & & c_j^2 & & c_kc_n \\ \vdots & & & \ddots & \vdots \\ c_nc_1 & \cdots & c_nc_j & \cdots & c_n^2 \end{bmatrix} \begin{bmatrix} X_{11} - x_1^2 & \cdots & X_{1j} - x_1x_j & \cdots & X_{n1} - x_1x_n \\ \vdots & \ddots & & & \vdots \\ X_{1k} - x_1x_k & & X_{jj} - x_j^2 & & X_{nk} - x_kx_n \\ \vdots & & & \ddots & \vdots \\ X_{1n} - x_nx_1 & \cdots & X_{nj} - x_nx_j & \cdots & X_{nn} - x_n^2 \end{bmatrix} = 0$$

On a donc pour tout k et j dans $\{1,...,n\}$: $c_k \left(\sum_{i=1}^n c_i \left(X_{ij} - x_ix_j\right)\right) = 0$. Si le vecteur c est nul, le résultat est trivial, on peut donc supposer qu'il existe k_0 dans $\{1,...,n\}$ tel que $c_{k0} \neq 0$. Ce qui nous donne pour tout j dans $\{1,...,n\}$ $\sum_{i=1}^n c_i X_{ij} = x_j \sum_{i=1}^n c_i x_i = dx_j$ puisque $c^T x = d$.

En fait, nous avons montré dans [48] (fourni dans l'annexe) que dans le cas de multiples égalités (écrites sous la forme d'un système linéaire de p lignes $Ax = b$), on peut même "agréger" le tout en une seule contrainte $A^T A \bullet X - 2b^T Ax + b^2 = 0$ dans le programme semidéfini (approche proposée en particulier dans [83]) et conserver la même région admissible. En effet, on a :

Proposition 4 [48] *Les régions $R_1 = \left\{(X,x) : X - xx^T \succcurlyeq 0,\ A^t A \bullet X - 2b^T Ax + b^2 = 0\right\}$ et $R_2 = \left\{(X,x) : Ax = b,\ X - xx^T \succcurlyeq 0,\ \sum_{k=1}^n A_{jk} X_{ki} - b_j x_i = 0 \forall i \in \{1,...,n\} \, \forall j \in \{1,...,p\}\right\}$ sont égales.*

Une différence majeure entre la programmation linéaire et la programmation semidéfinie est la présence de l'ensemble des variables de linéarisation dans la matrice X en SDP. A première vue, cela peut sembler être un inconvénient puisqu'en programmation linéaire on réduit souvent la taille du programme en faisant l'économie d'un sous-ensemble $\{X_{ij}\}_{(i,j)\in J}$ avec $J \subset (1,\ldots,n)^2$ de ces variables parce qu'elles n'interviennent ni dans la fonction économique ni dans les contraintes. En SDP, la contrainte $X - xx^T \succcurlyeq 0$ (liant toutes les variables X_{ij} aux produits x_ix_j) permet de redonner un intérêt à ces variables absentes car muettes dans l'approche par programmation linéaire.

Nous pouvons relire une partie des résultats de cette section à la lumière de ceux présentés au chapitre 5. En effet, nous savons déjà que les contraintes "produits" ajoutées à $P\{0,1\}$

(P_L)	$(SDP\{0,1\})$
	Règle **LE1**
$c^T x = d$	$c^T x = d\,;\, cc^T \bullet X = d^2$
	Règle **LE2**
	$cc^T \bullet X - 2dc^T x + d^2 = 0$
	Règle **LE3**
	$\sum_{j=1}^{n} c_j X_{ij} = dx_i \,\forall i \in \{1,\ldots,n\}$
	$c^T x = d$
	Règle **LE4**
$Ax = b$	$A^T A \bullet X - 2b^T Ax + b^2 = 0$

Tab. 6.1. *Règles équivalentes **LE** pour les égalités linéaires*

conduisent à un programme semidéfini ($(\text{SDP})_{\mathfrak{P}}$ contenant la contrainte $\mathbf{d}(X) = x$) équivalent à la relaxation lagrangienne partielle de $P\{0,1\}$ où les contraintes linéaires simples ne sont pas dualisées. D'un autre côté, un résultat semblable [83] concerne la relaxation semidéfinie $(\text{SDP})_{\mathfrak{C}}$ contenant $\mathbf{d}(X) = x$. Ainsi les propositions de cette section constituent des preuves directes de l'équivalence de ces relaxations semidéfinies (sans passer par la dualité lagrangienne). Nous constatons que l'ensemble des traitements usuels des contraintes linéaires d'égalité conduisent finalement tous à des relaxations semidéfinies équivalentes de $(P\{0,1\})$. Mais il s'agit là d'une vision uniquement du côté "primal", alors qu'en pratique les programmes semidéfinis duaux jouent un rôle tout aussi important dans les logiciels de résolution numérique. Nous illustrerons ce point dans la partie III de ce document pour le problème de l'affectation quadratique (QAP).

6.3 Contraintes linéaires d'inégalité

A présent, supposons avoir à traiter dans notre relaxation linéaire de départ une contrainte de la forme $d' \leq c^T x \leq d$. Rappelons que si d ou d' sont absents nous pouvons utiliser respectivement à leur place $\sum_{i \text{ tel que } c_i > 0} c_i$ pour d et $\sum_{i \text{ tel que } c_i < 0} c_i$ pour d'. Le premier traitement examiné est le suivant :

Proposition 6.5. [100] *Si (X,x) est admissible pour $(P\{0,1\})$ alors $cc^T \bullet X - (d+d')\,c^T x + dd' \leq 0$ est équivalent à $d' \leq c^T x \leq d$. Si $(X,x) \in S_n \times \Re^n$ n'est pas entier et est tel que $cc^T \bullet X - (d+d')\,c^T x + dd' \leq 0$ et $X - xx^T \succcurlyeq 0$, alors soit $cc^T \left(X - xx^T\right) = 0$ ou $d' < c^T x < d$ ou $c^T x \notin [d', d]$.*

Remarquons qu'ajouter la contrainte $cc^T \bullet X - (d+d')\,c^T x + dd' \leq 0$ dans une relaxation linéaire n'aurait pas le même effet. Car c'est la contrainte $X - xx^T \succcurlyeq 0$ qui permet de pénaliser ou de rendre non admissibles certains points non entiers, et d'obtenir pour d'autres l'ensemble des contraintes quadratiques décrit par le produit matriciel nul $cc^T \left(X - xx^T\right) = 0$.

Comme proposé dans [88], multiplions à présent les inégalités $d' \leq c^T x \leq d$ par x_i ou par $(1 - x_i)$ pour chaque i dans $\{1, ..., n\}$. Les deux propositions suivantes montrent que, sous certaines hypothèses, cette technique permet d'obtenir de meilleurs bornes.

Proposition 6.6. [100] *Si (X, x) est admissible pour $(P\{0,1\})$ alors pour tout i dans $\{1, ..., n\}$ les contraintes $d'x_i \leq \sum_{j=1}^n c_j X_{ij} \leq dx_i$, et $d'(1 - x_i) \leq \sum_{j=1}^n c_j (x_j - X_{ij}) \leq d(1 - x_i)$ sont valides pour $(P\{0,1\})$. Si $(X, x) \in S_n \times \Re^n$ est tel que $X - xx^T \succcurlyeq 0$ et s'il existe i_0 dans $\{1, ..., n\}$ tel que $d'x_{i_0} \leq \sum_{j=1}^n c_j X_{i_0 j} \leq dx_{i_0}$ et $d'(1 - x_{i_0}) \leq \sum_{j=1}^n c_j (x_j - X_{i_0 j}) \leq d(1 - x_{i_0})$ alors $d' \leq c^T x \leq d$.*

Proposition 6.7. [100] *Si $c_j \geq 0$ pour tout $j \in \{1, ..., n\}$, $d' \leq 0$, et $(X, x) \in S_n \times \Re^n$ est tel que $X - xx^T \succcurlyeq 0$, alors : $\sum_{j=1}^n c_j X_{ij} \leq dx_i$ pour tout i dans $\{1, \ldots, n\}$ et $\sum_{j=1}^n c_j (x_j - X_{i_0 j}) \leq d(1 - x_{i_0})$ pour au moins un indice i_0 dans $\{1, \ldots, n\}$ impliquent $cc^T \bullet X - (d' + d) c^T x + dd' \leq 0$.*

Remarquons que l'hypothèse "il existe i_0 dans $\{1, \ldots, n\}$ tel que $\sum_{j=1}^n c_j (x_j - X_{i_0 j}) \leq d(1 - x_{i_0})$" peut être remplacée par $c^T x \leq d$.

Une autre idée naturelle est de se ramener au cas d'égalité (de la section précédente) en ajoutant une variable d'écart x_0 au problème. Imposons donc que $c^T x + x_0 = d$ avec $x_0 \geq 0$, et appliquons la règle LE2 du tableau 6.1 (elles sont toutes équivalentes). Notons $c' = \begin{pmatrix} c \\ 1 \end{pmatrix}$, $x' = \begin{pmatrix} x \\ x_0 \end{pmatrix}$ et imposons $X' - x'x'^T \succcurlyeq 0$. Un calcul identique à celui mené dans la proposition 6.2 donne $c'c'^T \bullet \left(X' - x'x'^T\right) + \left(c^T x + x_0 - d\right)^2 = 0$. On obtient donc également que $c^T x \leq d$, et aussi $c'c'^T \left(X' - x'x'^T\right) = 0$. Donc en suivant la preuve de la proposition 6.4, on obtient pour tout j dans $\{1, \ldots, n\}$ $\sum_{i=1}^n c_i (X_{ij} - x_j x_i) + X'_{j0} - x_j x_0 = 0$. Puisque $c^T x + x_0 = d$, il vient : $\sum_{i=1}^n c_i X_{ij} = dx_j - X'_{j0}$. En imposant à présent $X'_{j0} \geq 0$ pour tout j dans $\{1, \ldots, n\}$, on retrouve donc les n inégalités déjà étudiées $\sum_{i=1}^n c_i X_{ij} \leq dx_j$. Le tableau 6.2 résume les traitements étudiés et les règles associées.

(P_L)	$(SDP\{0,1\})$
	Règle **LI1**
$d' \leq c^T x \leq d$	$cc^T \bullet X - (d + d') c^T x + dd' \leq 0$
	Règle **LI2**
	$\sum_{j=1}^n c_j X_{ij} \leq dx_i \quad i = 1, ..., n$
	$d'x_i \leq \sum_{j=1}^n c_j X_{ij} \quad i = 1, ..., n$
	$d'(1 - x_i) \leq \sum_{j=1}^n c_j (x_j - X_{ij}) \quad i = 1, ..., n$
	$\sum_{j=1}^n c_j (x_j - X_{ij}) \leq d(1 - x_i) \quad i = 1, ..., n$

Tab. 6.2. *Règles **LI1** et **LI2** pour les inégalités linéaires*

6.4 Algorithme de construction d'une relaxation semidéfinie à partir d'une relaxation linéaire

En utilisant les règles et résultats précédents, on peut proposer l'algorithme suivant pour construire de manière systématique une relaxation SDP à partir d'une relaxation linéaire (P_L) donnée pour $P(\{0,1\})$:

1. Choisir une relaxation linéaire (P_L) pour le problème combinatoire $(P\{0,1\})$. Rappelons que X contient les variables de linéarisation présentes dans (P_L).
2. Construire la relaxation semidéfinie comme suit :
   a) Copier la fonction à optimiser.
   b) Remplacer les contraintes $x \in [0,1]^n$ par $X - xx^T \succcurlyeq 0$ (on a $\mathbf{d}(X) = x$).
   c) Copier les contraintes contenant des variables de linéarisation (*i.e.* X_{ij} $i,j \in \{1,\ldots,n\}$, $i \neq j$). On conserve donc en particulier toutes les inégalités efficaces utilisées dans la relaxation (P_L), spécifiques au problème considéré.
   d) Appliquer une des règles **LE1**, **LE2**, **LE3**, ou **LE4** aux contraintes d'égalité contenant uniquement x.
   e) Appliquer la règle **LI1** ou la règle **LI2** aux contraintes d'inégalité contenant uniquement x.
   f) Retirer éventuellement les contraintes redondantes.

Les propositions de ce chapitre et cet algorithme ont permis le développement du logiciel libre SDP_S [44] lors de plusieurs stages d'élèves-ingénieurs de l'Institut d'Informatique d'Entreprise que j'ai encadrés. SDP_S formule automatiquement des relaxations semidéfinies pour les programmes quadratiques en variables bivalentes sans que l'utilisateur n'ait besoin d'être un spécialiste de la SDP. Son intérêt principal est de permettre d'élaborer puis de tester rapidement de multiples relaxations semidéfinies pour un problème donné à partir de relaxations linéaires connues. La documentation du logiciel est disponible dans l'annexe [44]. Nous avons bien sûr utilisé de manière intensive cet outil pour toutes nos études expérimentales présentées dans la partie III de ce document. Une version graphique (SDP_SX) [9] a été développée afin de rendre encore plus aisée l'utilisation de SDP_S. Elle permet en particulier de tracer les courbes de convergence du logiciel gratuit SB [67] dont les sources ont été utilisées pour la résolution numérique des programmes semidéfinis. Une illustration est donnée dans la figure 6.1.

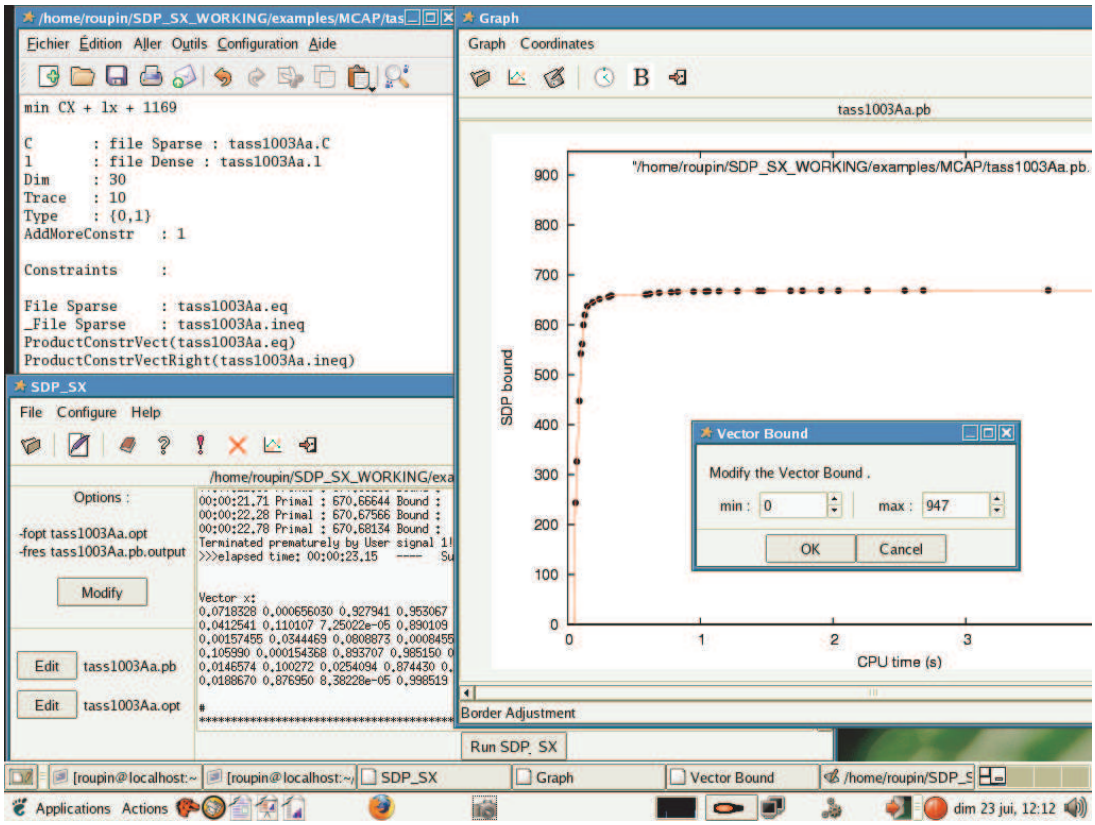

Fig. 6.1. *SDP_SX : l'interface graphique de SDP_S*

6.5 Conclusion et perspectives

Une extension naturelle de notre approche est son application au cas des programmes en variables mixtes. Un travail préliminaire à ce sujet a été effectué dans [84]. Il concerne le cas particulier des programmes pouvant s'exprimer comme :

$$
(P)\begin{cases}
\text{Min} \quad A_0 \bullet X + B_0 \bullet Y + C_0 \bullet Z + b^T x + c^T y & \\
\text{s.c. :} & \\
(Q) \quad A_i \bullet X + B_i \bullet Y + C_i \bullet Z + c_i^T x + d_i^T y = \ (\text{ou} \leq)\, e_i & i \in \{1, ..., t_1\} \\
(LE)\ c_i^T x + d_i^T y = e_i & i \in \{t_1 + 1, ..., t_2\} \\
(LI) \quad e_i \leq c_i^T x + d_i^T y \leq e_i' & i \in \{t_2 + 1, ..., t\} \\
\qquad X = xx^T \ Y = yy^T \ Z = xy^T & \\
\qquad x \in \{0,1\}^n \ \ y \in [0,1]^m &
\end{cases}
$$

où $1 \leq t_1 \leq t_2 \leq t$. X est donc associée à la partie quadratique bivalente, Y à la partie quadratique réelle, et Z à la partie quadratique mixte. Les matrices A_i sont éléments de S_n, B_i de S_m, et C_i sont des matrices réelles $n \times m$. Si pour tout i dans $\{1, \ldots, m\}$, on se donne $\overline{y_i}$ une borne supérieure de chaque y_i (remarquons qu'il est toujours possible de prendre 1 ici). Une relaxation linéaire possible du problème (P) est :

$$
(PL)\begin{cases}
\text{Min} & A_0 \bullet X + B_0 \bullet Y + C_0 \bullet Z + c^T x + d^T y & \\
\text{s.c. :} & & \\
(Q') & A_i \bullet X + B_i \bullet Y + C_i \bullet Z + c_i^T x + d_i^T y = (\text{or} \leq)\, e_i & i \in \{1, ..., t_1\} \\
(LE) & c_i^T x + d_i^T y = e_i & i \in \{t_1 + 1, ..., t_2\} \\
(LI) & e_i \leq c_i^T x + d_i^T y \leq e'_i & i \in \{t_2 + 1, ..., t\} \\
 & 0 \leq x_i \leq 1 & i \in \{1, ..., n\} \\
 & 0 \leq y_j \leq 1 & j \in \{1, ..., m\} \\
 & \text{Dont les contraintes de linéarisation de type :} & \\
(1) & X_{ij} \geq 0 & i < j \in \{1, ..., n\} \\
(1') & Y_{ij} \geq 0 & i < j \in \{1, ..., m\} \\
(1'') & Z_{ij} \geq 0 & i < j \in \{1, ..., n\} \times \{1, ...m\} \\
(2) & X_{ij} \leq x_i\,;\, X_{ij} \leq x_j & i < j \in \{1, ..., n\} \\
(2') & Y_{ij} \leq \overline{y_j} y_i\,;\, Y_{ij} \leq \overline{y_i} y_j & i < j \in \{1, ..., m\} \\
(2'') & Z_{ij} \leq \overline{y_j} x_i\,;\, Z_{ij} \leq y_j & i < j \in \{1, ..., n\} \times \{1, ...m\} \\
(3) & x_i + x_j - 1 \leq X_{ij} & i < j \in \{1, ..., n\} \\
(3') & \overline{y_j} y_i + \overline{y_i} y_j - \overline{y_i y_j} \leq Y_{ij} & i < j \in \{1, ..., m\} \\
(3'') & \overline{y_j} x_i + y_j - \overline{y_i} \leq Z_{ij} & i < j \in \{1, ..., n\} \times \{1, ...m\} \\
 & X \in S_n; & \\
 & Y \in S_m; &
\end{cases}
$$

On considère le vecteur $u = \begin{pmatrix} x \\ y \end{pmatrix}$ et la matrice $U = uu^T$. Cette contrainte est relâchée en $U - uu^T \succcurlyeq 0$, i.e. $\begin{bmatrix} 1 & u^T \\ u & U \end{bmatrix} = \begin{bmatrix} 1 & x & y^T \\ x & X & Z \\ y & Z^T & Y \end{bmatrix} \succcurlyeq 0$. Les propositions de ce chapitre ainsi que l'algorithme présenté dans la section précédente se généralisent alors facilement [84]. Puisqu'à l'exception du traitement des contraintes d'intégrité, on retrouve les mêmes types de contraintes à traiter. Il suffit ici d'imposer $\mathbf{d}(X) = x$ et $\mathbf{d}(Y) \leq \hat{y}$, où $\hat{y}_i = y_i \overline{y}_i$ pour tout i dans $\{1, \ldots, m\}$. Ces inégalités linéaires garantiront que $y \in [0, 1]$. Si aucune borne meilleure que 1 n'est connue a priori pour les composantes de y, on obtient alors $\mathbf{d}(Y) \leq y$. Par exemple, en généralisant les règles LE1 et LI1, on obtient alors le programme semidéfini suivant à partir de (PL) :

$$(SDP)\begin{cases} \text{Min} & A_0 \bullet X + B_0 \bullet Y + C_0 \bullet Z + b^T x + c^T y \\ \text{s.c. :} & \\ (Q') & A_i \bullet X + B_i \bullet Y + C_i \bullet Z + c_i^T x + d_i^T y = (\text{ou} \leq)\, e_i & i \in \{1, ..., t_1\} \\ (LE') & c_i c_i^T \bullet X + d_i d_i^T \bullet Y - 2e_i \left(c_i^T x + d_i^T y\right) - 2c_i d_i^T \bullet Z = -e_i^2 & i \in \{t_1 + 1, ..., t_2\} \\ (LI') & c_i c_i^T \bullet X + d_i d_i^T \bullet Y - (e_i + e'_i) \left(c_i^T x + d_i^T y\right) - 2c_i d_i^T \bullet Z \leq -e'_i e_i & i \in \{t_2 + 1, ..., t\} \\ & \begin{bmatrix} 1 & x^T & y^T \\ x & X & Z \\ y & Z^T & Y \end{bmatrix} \succcurlyeq 0 & \\ & \mathbf{d}(X) = x,\, \mathbf{d}(Y) \leq \hat{y} & \end{cases}$$

Notons cependant que certaines linéarisations de (P) sont compactes, i.e. emploient un nombre réduit de variables. Cette approche a été présentée dans [58] et sera par ailleurs utilisée dans le chapitre 8 de ce document pour un problème de placement de tâches dans un système distribué. En particulier, la contrainte de borne sur y n'est généralement pas 1, ce qui ne pose pas de problème pour la généralisation de notre approche. Par contre, nous perdons le caractère "compact" de cette approche en conservant les matrices X, Y et Z qui ne sont pas présentes sous cette forme dans cette approche. C'est pourquoi un champ de recherche est encore ouvert pour traiter efficacement ces linéarisations par programmation semidéfinie (voir la conclusion générale au chapitre 9).

A présent, nous allons pouvoir appliquer notre algorithme à plusieurs relaxations linéaires connues pour leur efficacité dans la partie III de ce document, et nous obtiendrons ainsi directement et facilement des relaxations semidéfinies utilisables en pratique pour les problèmes combinatoires considérés.

Troisième partie

Résolution de problèmes combinatoires par Programmation Semidéfinie

7

Utilisation de la SDP dans le cadre d'une résolution exacte

7.1 Introduction

L'objectif de ce chapitre est de montrer que la programmation semidéfinie peut être utilisée efficacement comme évaluation dans une méthode de séparation/évaluation pour résoudre exactement certains problèmes combinatoires (comme l'illustrent les travaux expérimentaux présentés dans [68] pour les programmes quadratiques en 0-1). L'excellence des bornes obtenues par cette approche peut en effet compenser le temps nécessaire à la résolution numérique des SDP. Cependant, nous verrons que la mise en place d'un tel algorithme soulève des problèmes spécifiques, et en particulier la nécessité d'obtenir très tôt des solutions admissibles par l'utilisation de méthodes issues de l'approximation polynomiale avec garanties de performance au pire cas.

7.2 Les problèmes Vertex-Cover et Max-Stable

Il existe de nombreux résultats théoriques concernant l'approximation des problèmes Vertex-Cover et Max-Stable, et en particulier en utilisant la programmation semidéfinie [87]. Cependant, peu d'expérimentations ont été menées avec cette approche dans le cadre d'une résolution exacte. Dans les années 1990, cela était dû en grande partie à l'absence d'outils numériques efficaces pour résoudre les SDP, ce qui rendait cette approche non compétitive. Cependant, en considérant la très grande qualité des bornes obtenues par SDP pour ces problèmes, Van-Dat Cung (actuellement professeur à l'ENSGI-INPG, Grenoble) et moi-même avons proposé [41] un algorithme de séparation/évaluation pour le problème Vertex-Cover fondé sur CSDP2.3 [29] (logiciel de résolution numérique de programmes semidéfinis fondé sur une méthode de point intérieur) et la bibliothèque BOB-PM2 [10]. Ce premier travail a été complété par W.J. Van Hoeve lors de son stage de fin d'études, encadré par V.D. Cung et moi-même, pour le problème (équivalent) Max-Stable (voir [71] fourni en annexe).

Considérons $G = (V, E)$, un graphe non-orienté où un entier positif w_i est associé à chaque sommet v_i $i \in \{1, \ldots, n\}$. Une couverture par les sommets des arêtes (ou vertex-cover) de G est

un ensemble $S \subset V$ tel que chaque arête $e \in E$ possède une extrémité au moins dans S. L'objectif est alors de déterminer un vertex-cover de poids total minimum. La formulation linéaire standard de ce problème en variables 0-1 est :

$$(LP)_{VC} \begin{cases} \min \sum_{i=1}^{n} w_i x_i \\ \text{s.c. : } x_i + x_j \geq 1 \ (i,j) \in E \\ \qquad x_i \in \{0,1\} \ \ i = 1, \ldots, n \end{cases} \tag{7.1}$$

Une solution optimale $\overline{x}$ de $(\underline{LP})_{VC}$, la relaxation continue de $(LP)_{VC}$, permet d'obtenir une solution $\widehat{x}$ $\frac{1}{2}$-approchée très simplement en choisissant $\widehat{x}_i = 1$ si et seulement si $\overline{x}_i \geq \frac{1}{2}$ pour tout i dans $\{1, \ldots, n\}$. Kleinberg et Goemans ont proposé et étudié dans [59] la relaxation semidéfinie $(\text{SDP})_{VC}$, obtenue à partir d'un modèle en variables à valeurs dans $\{-1, 1\}$.

$$(SDP)_{VC} \begin{cases} \min \frac{1}{2} \sum_{i=1}^{n} w_i \, (1 + Y_{0i}) \\ \text{s.c. :} \\ \qquad Y_{ij} - Y_{0i} - Y_{0j} = -1 \, ; \, \forall (i,j) \in E \\ \qquad \mathbf{d}(Y) = e_{n+1} \\ \qquad Y \in S_{n+1} \, ; \, Y \succcurlyeq 0 \end{cases} \tag{7.2}$$

La valeur optimale de $(SDP)_{VC}$ vaut $\sum_{i=1}^{n} w_i - \vartheta(G)$ [59], où $\vartheta(G)$ est le nombre de Lovász de G [87]. Cette célèbre borne supérieure de $\alpha(G)$, le nombre de stabilité de G, permet de calculer ce dernier en temps polynomial dans les graphes parfaits grâce à la relation $\alpha(G) \leq \vartheta(G) \leq \xi(\overline{G})$, où $\xi(\overline{G})$ est le nombre chromatique du graphe complémentaire de G. $\vartheta(G)$ peut être défini, entre autres, comme la valeur optimale des deux programmes semidéfinis équivalents suivants [80] :

$$(\Theta_1) \begin{cases} \max e_n e_n^T \bullet Z \\ \text{s.c. :} \\ \qquad Z_{ij} = 0 \, ; \, \forall (i,j) \in E \\ \qquad I \bullet Z = 1 \\ \qquad Z \in S_n \, ; \, Z \succcurlyeq 0 \end{cases} \Leftrightarrow (\Theta_2) \begin{cases} \max \sum_{i=1}^{n} w_i x_i \\ \text{s.c. :} \\ \qquad X_{ij} = 0 \, ; \, \forall (i,j) \in E \\ \qquad \mathbf{d}(X) = x \\ \qquad X \in S_n \, ; \, X - xx^T \succcurlyeq 0 \end{cases} \tag{7.3}$$

On retrouve donc la même relation entre ces deux bornes que celle qui lie les problèmes équivalents Vertex-Cover et Max-Stable. Remarquons que nous retrouvons facilement $(SDP)_{VC}$ en formulant (Θ_2) dans le modèle $\{-1, 1\}$. La relaxation $(SDP)_{VC}$ est améliorée dans [59] en ajoutant les inégalités valides $Y_{ij} - Y_{0i} - Y_{0j} \geq -1 \ \forall (i,j) \in V^2 \backslash E$. Il s'agit en fait des inégalités de linéarisation $x_i + x_j \leq 1 + X_{ij}$ écrites dans le modèle $\{-1, 1\}$ (voir tableau 4.1, chapitre 4). On obtient alors une relaxation semidéfinie de valeur $\sum_{i=1}^{n} w_i - \vartheta'(G)$ (voir la preuve dans [59]), où $\vartheta'(G)$ est la valeur du programme (Θ_1) auquel on a ajouté les contraintes $Z_{ij} \geq 0$ pour tout $i < j$ dans $\{0, \ldots, n\}$ (borne proposée dans [102]). En appliquant notre algorithme de la section 6.4 du chapitre 6, nous obtenons directement (Θ_2) (et donc $(SDP)_{VC}$) en partant de la relaxation continue de $(LP)_{VC}$.

Les conditions sont ici idéales pour utiliser la programmation semidéfinie : l'approche par programmation linéaire fournit des bornes assez pauvres (calculées à l'aide du logiciel PCx dans le tableau 7.1), alors que celles obtenues par SDP sont excellentes. En particulier, la relaxation $(SDP)_{VC}$ fournit des solutions entières (et donc optimales) pour la plupart des graphes de moins de 75 sommets. Cependant, afin de rentabiliser le temps passé à la résolution des programmes semidéfinis, nous avons choisi de construire à chaque sommet de l'arbre de recherche une solution entière admissible à partir de la solution optimale de la relaxation SDP correspondante.

Type	taille	d	PL	SDP	Opt.	Err PL	Err SDP	PL Temps CPU	SDP Temps CPU
pondéré	50	0,1	669	746	752	11,03%	0,80%	0,03s	0,2s (5)
pondéré	50	0,1	606	616	616	1,62%	0.00%	0,02s	0,2s (1)
pondéré	75	0,1	945	1129	1139	17,03%	0,88%	0,15s	5,7s (7)
pondéré	100	0,1	1344	1709	1745	22.9%	2.06%	1,47s	30,4s (75)
pondéré	100	0,3	1374	1827	1835	25,12%	0,44%	49,3s	246,0s (17)
pondéré	150	0,05	2027	2418	2493	18,69%	3,00%	1,05s	48,1s (231)
pondéré	200	0,03	2489	2697	2697	7,71%	0,00%	1,17s	25,5s (1)
pondéré	200	0,03	2566	2765	2772	7,43%	0,25%	1,81s	116,2s (5)
non pondéré	250	0,02	123	133	134	8,21%	0,75%	1,91s	165,4s (9)

Tab. 7.1. *Evaluation de la racine de l'arbre de recherche . PL correspond à $(\underline{LP})_{VC}$ et SDP à $(SDP)_{VC}$. La valeur indiquée entre parenthèses dans la dernière colonne est le nombre d'évaluations par SDP nécessaires à la résolution exacte de l'instance par notre méthode d'évaluation/séparation.*

Cela permet un élagage plus efficace de notre arbre de recherche (éventuellement dès la racine). Pour cela, pour chaque i dans $\{0,\ldots,n\}$ on considère le vecteur réel $(Y_{i1},\ldots,Y_{in})$ de la solution optimale du SDP. Puis, pour chaque j dans $\{1,...,n\}$ si $Y_{ij} > 0$ alors on pose $y_i = 1$ sinon $y_i = -1$. Tant qu'une arête n'est pas couverte, on applique l'heuristique gourmande suivante : poser $y_{i_0} = 1$ où i_0 est tel que $\frac{w_{i_0}}{d_{i_0}} = \min_{i/\exists(i,j)\in E \text{ non couverte}} \left\{\frac{w_i}{d_i}\right\}$ où d_i est le degré du sommet v_i dans le graphe partiel formé des arêtes non couvertes. On retient bien sûr la meilleure solution entière trouvée pendant l'algorithme. La stratégie de parcours de l'arbre de recherche est "le meilleur d'abord" et l'algorithme de séparation/évaluation est parallélisé grâce à la bibliothèque BOB-PM2 [10]. Nous avons utilisé un SMP 4xPentium II 400Mhz sous Linux pour l'ensemble des tests. Les résultats obtenus sont concurrentiels avec l'état de l'art (voir par exemple [90]). Remarquons dans le tableau 7.2 que nous obtenons parfois des valeurs du "Speed-Up" (égal au rapport du temps d'exécution de l'algorithme par un seul processeur par le temps d'exécution de l'algorithme parallélisé) supérieures au nombre de processeurs utilisés. Cela démontre l'intérêt de la parallélisation de l'algorithme de séparation/évaluation. Le nombre de contraintes du programme $(SDP)_{VC}$ dépendant de la densité du graphe, notre approche est performante pour des graphes de faible ou moyenne densité. D'autres résultats numériques pour le problème Max-Stable sont présentés dans [71] fourni dans l'annexe.

Taille	d	noeuds évalués	Temps CPU	# processeurs	Speed Up
100	0.1	7+2	179 s	2	1.25
100	0.1	11+0	333 s	2	1.01
100	0.1	23+20	1460 s	2	2.29
100	0.3	3+0	829 s	2	0.99
150	0.05	67+58+58	5098s	3	3.56
150	0.1	13+9	2775 s	2	1.93
250	0.02	7+2	965 s	2	1.72

Tab. 7.2. Vertex-Cover *: Résolution exacte par un algorithme de séparation/évaluation parallèlisé.*

7.3 Le problème Max-2Sat

Soit un ensemble de variables booléennes $X = \{x_1, ..., x_n\}$ et un ensemble de clauses $X = \{C_1, ..., C_m\}$ comportant au plus deux littéraux. Dans le problème Max-2Sat, on cherche à fixer les variables à "vrai" ou "faux" de façon à maximiser le nombre de clauses satisfaites. Ici, afin d'obtenir des relaxations semidéfinies, on formule Max-2Sat comme un programme quadratique à variables bivalentes. En notant $v(C_i)$ la valuation de la i^{me} clause C_i, on remarque que $v(x_i \vee x_j) = 1 - v(\overline{x}_i \wedge \overline{x}_j) = 1 - v(\overline{x}_i)\, v(\overline{x}_j) = 1 - (1 - x_i)(1 - x_j) = x_i + x_j - x_i x_j$. Pour les autres types de clauses on fait le même type de calcul en remplaçant simplement x_i par $-x_i$ si le littéral apparaît complémenté. Cela permet formuler notre problème comme suit :

$$(\text{Max-2Sat}) \begin{cases} \max \sum_{i=1}^{m} v(C_i) = x^T A x + b^T x = A \bullet X + b^T x \\ \text{s.c. : } X = xx^T \,,\, x \in \{0,1\}^n \end{cases}$$

Il suffit ensuite de relâcher la contrainte $X = xx^T$ en $X \succcurlyeq xx^T$ pour obtenir une relaxation semidéfinie. Cette dernière sera bien entendu équivalente à celle obtenue à partir d'un modèle $\{-1, 1\}$ (voir chapitre 4). En effet, en travaillant à présent avec $n + 1$ variables y_i à valeurs dans $\{-1, 1\}$, on a $v(x_i \vee x_j) = 1 - v(\overline{x}_i \wedge \overline{x}_j) = 1 - v(\overline{x}_i)\, v(\overline{x}_j) = 1 - \frac{1-y_0 y_i}{2} \frac{1-y_0 y_j}{2} = \frac{1}{4}(3 + y_0 y_i + y_0 y_j - y_i y_j)$. Cela conduit à une relaxation semidéfinie de la forme [60] :

$$(SDP_{M2S}) \begin{cases} \max C \bullet Y \\ \text{s.c. : } \mathbf{d}(Y) = e_{n+1} \,;\, Y \succcurlyeq 0 \,;\, Y \in S_{n+1} \end{cases}$$

Nous avons proposé dans [99] un algorithme de séparation/évaluation fondé sur (SDP_{M2S}). Afin d'élaguer au plus tôt l'arbre de recherche, nous avons utilisé les remarquables résultats en approximation polynomiale développés dans [51, 60] pour construire à chaque sommet de l'arbre de recherche une solution admissible très proche de l'évaluation. Le principe est de décomposer $Y \succcurlyeq 0$, la solution optimale du SDP, en $Y = VV^T$, et d'utiliser le champ de vecteurs unitaires donné par les lignes v_i de V pour constuire une solution admissible. Trois approches sont décrites dans [51, 60] :

1. Prendre un vecteur e_0 uniformément distribué sur la sphère unité de dimension n, puis chaque i dans $\{1,\ldots,n\}$, si $v_i^T e_0 \geq 0$ alors on pose $y_i = 1$ et sinon $y_i = -1$. On peut montrer [60] que cela conduit à un algorithme (aléatoire) $0,878$-approché pour MAX-2SAT.
2. On peut améliorer ce résultat en étudiant les pires cas de la méthode précédente. Par exemple, pour la clause $\overline{x}_i \vee \overline{x}_j$, la contribution dans la fonction objectif est $\frac{1}{4}(3 - Y_{0i} - Y_{0j} - Y_{ij})$ et le pire cas est celui où deux des variables de l'expression précédente valent $-0,689$ et le troisième 1. L'idée est alors pour chaque i dans $\{1,\ldots,n\}$ de fixer $y_i = 1$ avec une probabilité δ si $v_i^T v_0 \geq 0$ (sinon $y_i = -1$), et dans l'autre cas d'appliquer l'approche 1 avec la probabilité $1-\delta$. Ainsi, on améliore le pire cas si δ est choisi suffisamment petit. Mais malheureusement, pour obtenir un ratio meilleur $0,888$ il est nécessaire d'ajouter toutes les inégalités de linéarisation (voir tableau 4.1, chapitre 4) : $Y_{0i} + Y_{0j} + Y_{ij} \geq -1$, $-Y_{0i} - Y_{0j} + Y_{ij} \geq -1$, et $-Y_{0i} + Y_{0j} - Y_{ij} \geq -1$. Ce résultat rend inexploitable cette approche pour notre usage, car le programme semidéfini obtenu perd alors son caractère creux et est beaucoup plus long à résoudre.
3. Une troisième technique consiste à associer préalablement un nouveau champ de vecteurs W au champ V avant d'appliquer l'approche 1 sur W.

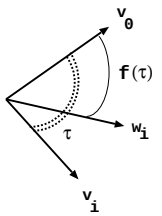

FIG. 7.1. *Transformation du champ de vecteurs*

Pour cela on choisit w_i dans le plan défini par (v_0, v_i) du même coté de v_0 et v_i (pour obtenir deux angles inférieurs ou égaux à $\frac{\pi}{2}$), et tel que $\arccos\left(w_i^T v_0\right) = f\left(\arccos\left(v_i^T v_0\right)\right)$, où f est une fonction vérifiant $f(\pi - \tau) = \pi - f(\tau)$ de manière à traiter de la même façon les littéraux positifs et négatifs. Prendre $f(\tau) = \tau$ revient donc à utiliser la méthode 1. Il a été montré numériquement en discrétisant les valeurs des angles [51] (il semble insurmontable de mener la preuve de façon analytique) que la fonction $f(\tau) = \tau + 0.806765\left[\frac{\pi}{2}(1 - cos(\tau) - \tau)\right]$ conduit à une solution $0,931$-approchée si les inégalités de linéarisation sont ajoutées à (SDP_{M2S}). Remarquons que cette approche a été généralisée récemment dans [53].

Nos tests numériques préliminaires ont montré que la première technique conduit à des solutions en moyenne à 3% à la racine de l'arbre de recherche, alors que la troisième approche permet d'obtenir des solutions à moins de 1%, et ceci sans utiliser les contraintes de linéarisation. Le caractère creux de la relaxation SDP_{M2S} étant préservé, une méthode de point-intérieur pour la résolution des programmes semidéfinis telle que CSDP [29] est donc adaptée (la convergence est ici très rapide : en moyenne 7 itérations). Nous disposons à chaque itération de CSDP d'une

# variables	# clauses	B&C	B&B SDP	# évaluations SDP
50	100	3,32 s	52,2 s	111
50	500	26,94 s	6,5 s	13
50	1000	64,76 s	33, 1s	67
50	1500	1933,55 s	69,7 s	97
50	2500	3378,69 s	173,7 s	331
50	3500	10142,31 s	277,1 s	495
50	5000	37241,84 s	50,59 s	75
100	1000	22915,42 s	1630,78 s	1273
100	2500	non résolu	2834,22 s	2109

Tab. 7.3. *Temps CPU de résolution exacte pour* Max-2Sat.

borne par le programme dual ce qui permet dans certains cas d'élaguer l'arbre de recherche avant même que le SDP ne soit totalement résolu. L'exploration est effectuée en profondeur d'abord et la séparation est effectuée sur la variable d'indice i_0 tel que $Y_{i_0 n+1} = \min_i |Y_{in+1}|$. Dans le tableau 7.3, B&C correspond aux résultats obtenus sur un IBM RS6000/390 par l'algorithme de Branch&Cut proposé par Joy, Mitchell, et Borchers dans [73], et B&B SDP à notre approche fondée sur (SDP_{M2S}). Nous avons utilisé un Pentium II 400Mhz sous Linux dont la puissance de calcul est comparable (environ 1,4 fois plus importante, voir [45]). Les instances utilisées sont disponibles à l'adresse *http ://infohost.nmt.edu/˜borchers/*. Récemment, une nouvelle approche particulièrement efficace pour la résolution exacte de Max-2Sat utilisant des méthodes de séparation fondées sur la programmation disjonctive a été proposée dans [28]. Il serait intéressant d'effectuer une comparaison avec l'approche semidéfinie en utilisant les derniers outils de résolution numérique des SDP (la version de CSDP utilisée dans nos tests de 2001 [99] est la 2.2).

7.4 Conclusion

Nous avons pu constater dans ce chapitre que la programmation semidéfinie peut être une approche efficace pour résoudre exactement certains problèmes combinatoires par une méthode de séparation/évaluation. Cependant, cela n'est possible que dans certaines conditions. L'existence d'un écart important entre la qualité des bornes fournies par l'approche par programmation linéaire et celle par programmation semidéfinie est un critère pertinent (comme pour le problème Max-Stable). D'autre part, il est nécessaire d'obtenir très tôt dans l'arbre de recherche une solution optimale du problème (comme pour Max-2Sat). En effet, très peu d'évaluations doivent être effectuées compte tenu du temps nécessaire à la résolution d'un programme semidéfini. Il est donc indispensable d'utiliser non seulement les bornes mais également les solutions optimales des relaxations semidéfinies pour construire à chaque sommet de l'arbre de recherche des solutions admissibles du problème combinatoire. La preuve de l'optimalité est alors obtenue rapidement compte tenu de l'élagage très important.

8

Utilisation de la SDP pour la résolution approchée

8.1 Introduction

Ce chapitre présente une synthèse de résultats concernant la résolution approchée de plusieurs problèmes combinatoires particulièrement difficiles. Pour ces problèmes, seules des instances de petite taille peuvent être résolues exactement, c'est pourquoi l'obtention de bornes de bonne qualité constitue déjà une tâche ardue. L'utilisation de la programmation semidéfinie est donc ici pertinente. Nous allons bien sûr appliquer les résultats théoriques et les méthodes systématiques présentés dans les chapitres 5 et 6 pour élaborer nos relaxations semidéfinies, et nous mettrons l'accent sur les problèmes pratiques de résolution numérique consécutifs au choix des relaxations choisies.

8.2 Le problème K-CLUSTER

Dans cette section nous allons présenter uniquement les résultats concernant l'approche semidéfinie pour ce problème qui a déjà été défini dans le chapitre 2. Considérons les modèles quadratiques en variables $\{0,1\}$ et $\{-1,1\}$ (équivalents) de K-CLUSTER :

$$(KC)_{\{0,1\}} \begin{cases} \max \frac{1}{2} W \bullet X \\ \text{s.c.} \quad \sum_{i=1}^{n} x_i = k \\ \qquad x \in \{0,1\}^n \\ \qquad X = xx^T \end{cases} \qquad (KC)_{\{-1,1\}} \begin{cases} \max \frac{1}{8} \sum_{i=1}^{n} \sum_{j \neq i} W_{ij} \left(1 + y_i y_j + y_i + y_j\right) \\ \text{s.c.} \quad \sum_{i=1}^{n} y_i = 2k - n \\ \qquad y \in \{-1,1\}^n \end{cases}$$

En utilisant les résultats du chapitre 4, on obtient les relaxations semidéfinies basiques (équivalentes) de ces deux programmes :

$$(KC\,SDP)_{\{0,1\}} \begin{cases} \max \frac{1}{2} W \bullet X \\ \text{s.c.} \quad \sum_{i=1}^{n} x_i = k \\ \qquad \mathbf{d}(X) = x \\ \qquad \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succeq 0 \end{cases} \qquad (KC\,SDP)_{\{-1,1\}} \begin{cases} \max \frac{1}{8} \sum_{i=1}^{n} \sum_{j \neq i} W_{ij} \left(1 + Y_{ij} + Y_{0i} + Y_{0j}\right) \\ \text{s.c.} \quad \sum_{i=1}^{n} Y_{0i} = 2k - n \\ \qquad \mathbf{d}(Y) = e_{n+1} \\ \qquad Y \succeq 0 \end{cases}$$

Plusieurs auteurs ont utilisé la programmation semidéfinie pour construire des algorithmes (aléatoires) approchés avec garantie de performance pour K-CLUSTER. L'objectif était de dépasser le ratio d'approximation $\frac{k}{n}$ qui semble hors d'atteinte par d'autres approches (par programmation linéaire en particulier). En 1998, en utilisant $(KC\,SDP)_{\{-1,1\}}$ un premier résultat de ce type était présenté [105]. Malheureusement, la démonstration (très technique) était fausse [106], un contre-exemple étant donné dans [52]. Parallèlement à ces premiers essais infructueux, d'autres auteurs eurent l'idée de renforcer $(KC\,SDP)_{\{-1,1\}}$ en lui ajoutant des égalités valides afin de rendre non admissibles les solutions des contre-exemples précédents. Ainsi, dans [63, 64], les auteurs ajoutent la contrainte $\sum_{i=1}^{n}\sum_{j=1}^{n} Y_{ij} = (2k-n)^2$, alors que dans [52], c'est l'ensemble des contraintes "produits" $\sum_{i=1}^{n}\sum_{j=1}^{n} Y_{ij} = Y_{0i}\,(2k-n)$ pour tout i dans $\{1,\ldots,n\}$ qui est considéré. Nous savons grâce aux résultats du chapitre 5 que ces contraintes conduisent en fait à des relaxations semidéfinies équivalentes.

Plus récemment, dans [72], les auteurs renforcent la relaxation proposée dans [64] en ajoutant l'ensemble des inégalités triangulaires et donc les inégalités de linéarisation (voir le tableau 4.1 du chapitre 4), pour obtenir finalement la relaxation semidéfinie $(KC\,SDP)_{SJ}$. Même si elle présente un intérêt théorique certain dans le domaine de l'approximation polynomiale, cette dernière relaxation est moins intéressante dans la pratique puisqu'elle contient $O\left(n^3\right)$ contraintes, et fournit des bornes de qualité comparable à celles que nous allons obtenir avec une relaxation contenant seulement $O\left(n^2\right)$ contraintes.

$$(KC\,SDP)_{SJ}\begin{cases}
\max \frac{1}{8}\sum_{i=1}^{n}\sum_{j\neq i} W_{ij}\,(1+Y_{ij}+Y_{0i}+Y_{0j}) & \\
\text{s.c.} \quad \sum_{i=1}^{n} Y_{0i} = 2k-n & \\
\qquad \sum_{i=1}^{n}\sum_{j=1}^{n} Y_{ij} = Y_{0i}\,(2k-n) & i\in\{1,\ldots,n\}\\
\qquad Y_{ij}+Y_{jk}+Y_{ik} \geq -1 & i,j,k \text{ distincts dans } \{0,\ldots,n\}\\
\qquad Y_{ij}-Y_{jk}-Y_{ik} \geq -1 & i,j,k \text{ distincts dans } \{0,\ldots,n\}\\
\qquad \mathbf{d}(Y) = e_{n+1} & \\
\qquad Y \succeq 0 &
\end{cases}$$

Dans [100], nous avons appliqué la démarche présentée dans le chapitre 6 en nous fondant sur les résultats existants en programmation linéaire. Au lieu de partir du modèle $(KC)_{\{-1,1\}}$ et d'enrichir la relaxation semidéfinie basique $(KC\,SDP)_{\{-1,1\}}$, nous avons profité du travail très complet de [18] sur l'approche par programmation linéaire pour le problème K-CLUSTER en considérant les relaxations linéaires issues de $(KC)_{\{0,1\}}$. En examinant les résultats théoriques et expérimentaux présentés dans [18], nous avons choisi d'appliquer l'algorithme présenté dans la section 6.4 du chapitre 6 au programme $(PL_3)_{KC}$, car cette relaxation linéaire donne globalement les meilleurs résultats expérimentaux (bornes de bonne qualité et temps de calcul raisonnable).

$$(PL_3)_{KC} \begin{cases} \max \ \frac{1}{2} W \bullet X \\ \text{s.c. :} \\ (a) \quad \sum_{i=1}^{n} x_i = e_n^T x = k \\ (b) \quad \sum_{i<j} X_{ij} + \sum_{j>i} X_{ij} = (k-1) x_i & i \in \{1, ..., n\} \\ (c) \quad X_{ij} \geq 0 & i < j \notin E \\ (d) \quad X_{ij} \leq x_i \, ; \, X_{ij} \leq x_j & i < j \notin E \\ \quad 0 \leq x_i \leq 1 & i \in \{1, ..., n\} \end{cases}$$

Nous appliquons donc notre algorithme en choisissant la règle LE1 pour la contrainte (a) pour obtenir (a') (voir la table 6.1 du chapitre 6). Ici utiliser la règle LE2 conduit à retrouver les contraintes (b') qui sont simplement copiées dans notre SDP. La contrainte $e_n e_n^T \bullet X = k^2$ est redondante (voir la proposition 6.3 du chapitre 6) mais elle permet d'accélérer la convergence de SB [67], le logiciel de résolution numérique des SDP que nous avons choisi d'utiliser.

$$(SDP_3)_{KC} \begin{cases} \max \ \frac{1}{2} W \bullet X \\ \text{s.c. :} \\ (a') \quad e_n^T x = k \, , \, e_n e_n^T \bullet X = k^2 \\ (b') \quad \sum_{i=1}^{n} X_{ij} = k x_j & \forall j \in \{1, ..., n\} \\ (c') \quad X_{ij} \geq 0 & i < j \in \{1, ..., n\} \\ (d') \quad X_{ij} \leq x_i \, ; \, X_{ij} \leq x_j & i < j \in \{1, ..., n\} \\ \quad \mathbf{d}(X) = x \, ; \, X - x x^T \succeq 0 \end{cases}$$

Les résultats présentés dans [100] pages 479-480 (voir l'annexe) montrent la pertinence de notre démarche : $(SDP_3)_{KC}$ est une meilleure relaxation que celles présentées dans [64] et [52], et elle fournit d'excellents résultats expérimentaux en terme de qualité des bornes obtenues et des temps de calcul. Il est en particulier remarquable que cette relaxation semidéfinie permette d'atteindre en un temps plus court les bornes obtenues par programmation linéaire. Ce phénomène est illustré dans la figure 1 page 481 de [100] (voir l'annexe).

8.3 CMAP : un problème d'allocation de tâches avec contraintes de ressources

Les problèmes d'allocation de tâches dans les systèmes distribués ont de nombreuses variantes et ont été beaucoup étudiés [19, 20, 21, 22, 33, 43, 62]. En particulier nous avons proposé dans [97] (fournie en annexe) une heuristique rapide fondée sur une méthode de décomposition et la programmation linéaire, et appliqué la méthode VNS (Variable Neighborhood Search) dans [36] pour une variante sans contraintes de ressources. Celle qui nous intéresse ici, le problème CMAP, consiste à affecter N tâches communicantes à P processeurs dans un système distribué,

de façon à minimiser la somme totale des coûts d'exécution et de communication. Une hypothèse particulière est faite sur les coûts de communication qui sont dits "uniformes" car ils ne dépendent que des tâches. Plus précisément $C_{tt'}$ est le coût de communication entre les tâches t et t', si ces dernières sont affectées à des processeurs différents (C_{tt} est nul pour tout $t \in \{1, ..., N\}$). Q_{tp} représente le coût d'exécution de la tâche t lorsqu'elle est affectée au processeur p. Enfin, chaque processeur possède une mémoire limitée n_p, et chaque tâche t requiert une quantité s_t de mémoire pour être exécutée. Pour modéliser notre problème comme un programme quadratique en 0-1 soumis à des contraintes linéaire, on utilise des variables de décision x_{tp} qui valent 1 si et seulement si la tâche t est affectée au processeur p. Les contraintes linéaires du programme traduisent le fait que chaque tâche doit être affectée à exactement un processeur et les contraintes de capacité des processeurs.

$$(CMAP) \begin{cases} \text{Min } \sum_{t=1}^{N} \sum_{p=1}^{P} Q_{tp} x_{tp} + \sum_{t=1}^{N} \sum_{t'<t} \sum_{p=1}^{P} C_{tt'} x_{tp} (1 - x_{t'p}) \\ \text{s.c. } \sum_{p=1}^{P} x_{tp} = 1 \qquad t = 1, \ldots, N \\ \quad \sum_{t=1}^{N} s_t x_{tp} \leq n_p \quad p = 1, \ldots, P \\ \quad x \in \{0,1\}^{NP} \end{cases}$$

CMAP est un problème $\mathcal{NP}$-difficile, et nous avons montré que sa résolution approchée avec garantie de performance fixée (algorithme ε-approché) même dans des cas très particuliers est un problème $\mathcal{NP}$-difficile (voir [98], fourni dans l'annexe). Des travaux portant sur différentes relaxations linéaires et une méthode de décomposition lagrangienne ont été menés par Sourour Elloumi et Eric Soutif du CEDRIC-CNAM dans [47]. Nous avons complété ces travaux avec ces auteurs en étudiant l'apport de la programmation semidéfinie à l'obtention de bornes pour CMAP [46, 100]. Au total, sept bornes différentes ont été proposées, et un travail comparaison théorique et d'expérimentation a permis de déterminer les avantages et inconvénients de chaque approche en termes de qualité et de temps de calcul. Pour une présentation exhaustive sur les approches par programmation linéaire et la méthode de décomposition lagrangienne voir [46] dans l'annexe.

Rappelons à présent les trois relaxations par programmation linéaire proposées dans [47]. Ces dernières nous seront utiles pour construire nos relaxations semidéfinies à l'aide de l'algorithme présenté dans le chapitre 6. La première $(LP)_1$, est obtenue en remplaçant chaque terme quadratique par une variable et les contraintes classiques de linéarisation. On note $L1$ la valeur optimale de $(LP)_1$.

$$(LP)_1 \begin{cases} \min & \sum_{t=1}^{N}\sum_{p=1}^{P} Q_{tp}x_{tp} + \sum_{t=2}^{N}\sum_{t'<t} C_{tt'} \\ & - \sum_{t=1}^{N}\sum_{t'=t+1}^{N}\sum_{p=1}^{P} C_{tt'}X_{tpt'p} \\ \text{s.c.} & \\ (Aff) & \sum_{p=1}^{P} x_{tp} = 1 & t = 1,\dots,N \\ (Cap) & \sum_{t=1}^{N} s_t x_{tp} \le n_p & p = 1,\dots,P \\ (L_1) & X_{tpt'p} \le x_{tp} & 1 \le t < t' \le N \quad p = 1,\dots,P \\ (L_2) & X_{tpt'p} \le x_{t'p} & 1 \le t < t' \le N \quad p = 1,\dots,P \\ (Pos) & X_{tpt'p} \ge 0\,;\, x_{tp} \ge 0 & p = 1,\dots,P \\ & & 1 \le t < t' \le N \end{cases}$$

Nous aurions pu noter $X_{tpt'p}$ simplement $X_{tpt'}$, mais cette notation facilitera par la suite notre comparaison avec les programmes semidéfinis. Dans [20], il est démontré que $L1$ est égale à la valeur optimale du dual lagrangien de (CMAP) où les contraintes (Aff) et (Cap) sont dualisées. Cette relaxation est très mauvaise si tous les coûts d'exécution sont nuls mais excellente si les contraintes de capacité (Cap) sont supprimées de (CMAP) [19].

$$(LP)_2 \begin{cases} \min & \sum_{t=1}^{N}\sum_{p=1}^{P} Q_{tp}x_{tp} + \sum_{t=2}^{N}\sum_{t'<t} C_{tt'} \\ & - \sum_{t=1}^{N}\sum_{t'=t+1}^{N}\sum_{p=1}^{P} C_{tt'}X_{tpt'p} \\ \text{s.c.} & \\ (Aff) & \sum_{p=1}^{P} x_{tp} = 1 & t = 1,...,N \\ (Prod) & \sum_{p=1}^{P} X_{tpt'p'} = x_{t'p'} & 1 \le t < t' \le N \\ & & p' = 1,\dots,P \\ (Cap) & \sum_{t=1}^{N} s_t x_{tp} \le n_p & p = 1,\dots,P \\ (LS_1) & \sum_{t=1}^{t'-1} s_t X_{tpt'p} + \sum_{t=t'+1}^{N} s_t X_{t'ptp} \le (n_p - s_{t'})x_{t'p} & t' = 1,...,N \\ & & p = 1,\dots,P \\ (LS_2) & \sum_{t=1}^{t'-1} s_t X_{tpt'p'} + \sum_{t=t'+1}^{N} s_t X_{t'p'tp} \le n_p x_{t'p'} & t' = 1,\dots,N \\ & & p,p' = 1,\dots,P, p' \ne p \\ & X_{tpt'p'} \ge 0\,;\, x_{tp} \ge 0 & p,p' = 1,\dots,P \\ & & 1 \le t < t' \le N \end{cases}$$

La seconde relaxation linéaire $(LP)_2$ est obtenue en suivant l'approche proposée dans [3]. On construit les contraintes linéaires $(Prod)$, (LS_1) et (LS_2) en multipliant les contraintes (Aff) et (Cap) par x_{tp}, puis en linéarisant à l'aide de X. Pour (Cap), il est inutile de considérer les contraintes obtenues en multipliant par $(1 - x_{tp})$ car ces dernières sont redondantes grâce aux contraintes (Aff). On peut démontrer que $L2$, la valeur optimale de (LP_2), est supérieure ou égale à $L1$ [47]. La troisième relaxation linéaire est obtenue en suivant la méthode proposée dans [58]. Pour tout (t,p) dans $\{1,\dots,N\} \times \{1,\dots,P\}$, on remplace l'expression

$x_{tp}\left(\sum_{t'=1,\ t'\neq t}^{N}\frac{1}{2}C_{tt'}x_{t'p}\right)$ par une nouvelle variable h_{tp}, et pour $\bar{\alpha}$ donné dans $\Re^{N\times P}$, on définit le programme linéaire mixte suivant :

$$(L_{\bar{\alpha}})\begin{cases}\min & \sum_{t=1}^{N}\sum_{p=1}^{P}Q_{tp}x_{tp}+\sum_{t=2}^{N}\sum_{t'<t}C_{tt'}-\sum_{t=1}^{N}\sum_{p=1}^{P}h_{tp} & & \\ \text{s.c.} & & & \\ (Aff) & \sum_{p=1}^{P}x_{tp}=1 & t=1,\ldots,N & \\ (Cap) & \sum_{t=1}^{N}s_t x_{tp}\leq n_p & p=1,\ldots,P & \\ (L_1) & h_{tp}\leq \sum_{t'=1,\ t'\neq t}^{N}\frac{1}{2}C_{tt'}x_{t'p} & t=1,\ldots,N & p=1,\ldots,P \\ (L_2) & h_{tp}\leq \bar{\alpha}_{tp}x_{tp} & t=1,\ldots,N & p=1,\ldots,P \\ & h_{tp}\geq 0\,;\ x\in\{0,1\}^{N\times P} & t=1,\ldots,N & p=1,\ldots,P\end{cases}$$

Sous certaines hypothèses sur $\bar{\alpha}$, la relaxation continue de $L_{\bar{\alpha}}$ fournit une borne inférieure de (CMAP). L'avantage évident de cette linéarisation "compacte" est le faible nombre de variables. Dans [47], on choisit chaque α^*_{tp} comme la solution d'un problème d'affectation particulier. Il est alors démontré que les programmes $(L_{\bar{\alpha}^*})$ et (CMAP) sont équivalents (voir [46] dans l'annexe pour les détails). On note L3 la valeur de $(\underline{L}_{\bar{\alpha}^*})$, la relaxation continue de $(L_{\bar{\alpha}^*})$. Comme indiqué dans la conclusion du chapitre 6 nous ne disposons pas de méthode satisfaisante pour l'instant pour élaborer une relaxation semidéfinie efficace et de taille comparable à celle de $(L_{\bar{\alpha}^*})$ en partant de ce programme linéaire.

La relaxation semidéfinie basique du programme $(CMAP)$ est particulièrement mauvaise ici, c'est pourquoi nous l'avons renforcée avec les contraintes de positivité sur tous les éléments de la matrice X pour obtenir (SDP_0). Ce programme peut être vu comme une relaxation semidéfinie "naive" directement élaborée à partir de $(CMAP)$.

$$(SDP_0)\begin{cases}\min & \sum_{t=1}^{N}\sum_{p=1}^{P}Q_{tp}x_{tp} & & \\ & -\sum_{t=1}^{N}\sum_{t'=t+1}^{N}\sum_{p=1}^{P}C_{tt'}X_{tpt'p}+\sum_{t=2}^{N}\sum_{t'<t}C_{tt'} & & \\ \text{s.c. :} & & & \\ (Aff) & \sum_{p=1}^{P}x_{tp}=1 & t=1,\ldots,N & \\ (Cap) & \sum_{t=1}^{N}s_t x_{tp}\leq n_p & p=1,\ldots,P & \\ (Sdp) & \mathbf{d}(X)=x\ ;\ \begin{bmatrix}1 & x^T\\ x & X\end{bmatrix}\succcurlyeq 0 & & \\ (Pos) & X_{tpt'p'}\geq 0 & p,p'=1,\ldots,P & \\ & & 1\leq t\leq N; 1\leq t'\leq N & \end{cases}$$

En revanche, puisque nous disposons de relaxations linéaires pour $(CMAP)$, nous pouvons appliquer l'algorithme décrit dans la section 6.4 du chapitre 6. Construisons pour commencer une relaxation semidéfinie (SDP_1) en partant de (LP_1). Nous utilisons la règle LE1 sur les contraintes (Aff) pour obtenir en plus (Car_1), et la règle LI1 sur (Cap) qui donne les contraintes (Car_2) puisque nous avons ici $\sum_{t=1}^{N} s_t x_{tp} \geq 0$ pour tout p dans $\{1,\ldots,P\}$. Rappelons que (Car_2) et $X - xx^T \succcurlyeq 0$ impliquent (Cap) (voir proposition 6.5 du chapitre 6). Enfin, les contraintes où apparaît X sont conservées et même étendues pour $p' \neq p$ (puisque toutes les variables de (X,x) sont liées par $X - xx^T \succcurlyeq 0$).

$$(SDP_1)\begin{cases} \min & \sum_{t=1}^{N}\sum_{p=1}^{P} Q_{tp}x_{tp} \\ & -\sum_{t=1}^{N}\sum_{t'=t+1}^{N}\sum_{p=1}^{P} C_{tt'}X_{tpt'p} + \sum_{t=2}^{N}\sum_{t'<t} C_{tt'} \\ \text{s.c. :} & \\ (Aff) & \sum_{p=1}^{P} x_{tp} = 1 & t = 1,\ldots,N \\ (Car_1) & \sum_{p=1}^{P}\sum_{p'=1}^{P} X_{tptp'} = 1 & t = 1,\ldots,N \\ (Car_2) & \sum_{t=1}^{N}\sum_{t'=1}^{N} s_t s_{t'} X_{tpt'p} - n_p \sum_{t=1}^{N} s_t x_{tp} \leq 0 & p = 1,\ldots,P \\ (L) & X_{tpt'p'} \leq x_{tp} & t,t' = 1,\ldots,N \text{ et } t \neq t' \\ & & p',p = 1,\ldots,P \\ (Sdp) & \mathbf{d}(X) = x \ ; \ \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succcurlyeq 0 & \\ (Pos') & X_{tpt'p'} \geq 0 & p,p' = 1,\ldots,P \\ & & 1 \leq t \leq N; 1 \leq t' \leq N \end{cases}$$

Construisons à présent (SDP_2) en partant de (LP_2).

$$(SDP_2)\begin{cases} \min & \sum_{t=1}^{N}\sum_{p=1}^{P} Q_{tp}x_{tp} \\ & -\sum_{t=1}^{N}\sum_{t'=t+1}^{N}\sum_{p=1}^{P} C_{tt'}X_{tpt'p} + \sum_{t=2}^{N}\sum_{t'<t} C_{tt'} \\ \text{s.c. :} & \\ (Aff) & \sum_{p=1}^{P} x_{tp} = 1 & t = 1,\ldots,N \\ (Car_1) & \sum_{p=1}^{P}\sum_{p'=1}^{P} X_{tptp'} = 1 & t = 1,\ldots,N \\ (Prod) & \sum_{p=1}^{P} X_{tpt'p'} = x_{t'p'} & 1 \leq t < t' \leq N \\ & & p' = 1,\ldots,P \\ (LS) & \sum_{t=1}^{N} s_t X_{tpt'p'} \leq n_p x_{t'p'} & p,p' = 1,\ldots,P \\ & & t' = 1,\ldots,N \\ (Sdp) & \mathbf{d}(X) = x \ ; \ \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succcurlyeq 0 & \\ (Pos) & X_{tpt'p'} \geq 0 & p,p' = 1,\ldots,P \\ & & 1 \leq t \leq N; 1 \leq t' \leq N \end{cases}$$

Toutes les contraintes où apparaît X dans (LP_2) sont conservées dans (SDP_2). Remarquons que (LS) est bien équivalent à (LS_1) et (LS_2) grâce à $\mathbf{d}(X) = x$ (on ne distingue donc pas

le cas où $p = p'$). Il est inutile d'appliquer la règle LI1 à (Aff) pour ajouter les contraintes (Car_2). En effet, nous savons ici que les contraintes $\sum_{t=1}^{N} s_t (x_{tp} - X_{tpt'p'}) \leq n_p (1 - x_{t'p'})$ pour $p, p' \in \{1, \ldots, P\}$ et $t' \in \{1, \ldots, N\}$ sont redondantes grâce aux contraintes (Aff) et (LS). Puisque $s_t \geq 0$ pour $t \in \{1, \ldots, N\}$, les contraintes (LS) et la proposition 6.7 du chapitre 6 impliquent (Car_2). La valeur optimale de (SDP_2) est notée $S2$. On montre que :

Proposition 5 [46] *$S0$, $S1$ et $S2$ étant les valeurs optimales respectives de (SDP_0), (SDP_1) et de (SDP_2), on a $S0 \leq S1 \leq S2$.*

De plus, en appliquant notre algorithme du chapitre 6, nous avons la garantie que :

Proposition 6 [46] *On a $L1 \leq S1$ et $L2 \leq S2$.*

Nous renvoyons le lecteur à [46] dans l'annexe pour le détails des résultats expérimentaux. La figure 8.1 en donne une synthèse pour l'ensemble des instances.

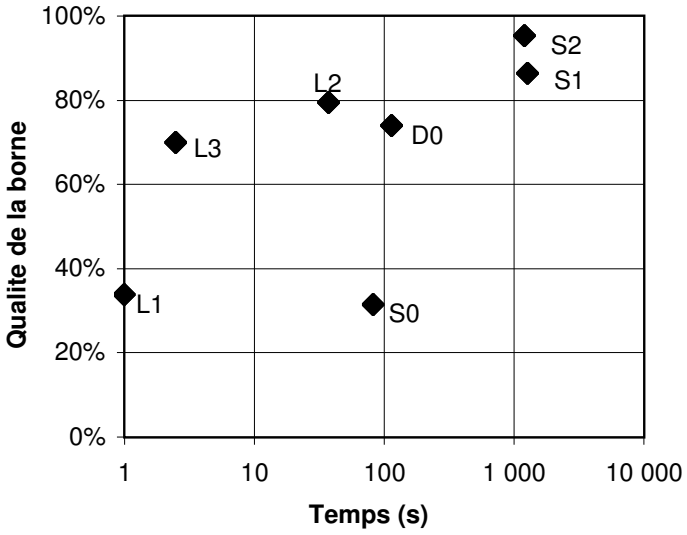

Fig. 8.1. *Comparaison des sept bornes sur l'ensemble des instances testées. D0 correspond à la méthode de décomposition lagrangienne.*

L'approche semidéfinie améliore considérablement les bornes obtenues par programmation linéaire pour le problème $(CMAP)$. Les temps de calcul semblent cependant rédhibitoires pour utiliser cette approche dans une méthode exacte. Cette impression doit être nuancée par la grande qualité des bornes obtenues par (SDP_2) ainsi que par la rapidité de convergence du logiciel SB au

début de la résolution de cette relaxation. Pour l'exemple donné dans la figure 8.3 on ne dégrade la borne fournie par (SDP_2) que de 2% pour un temps de calcul divisé par 4. (SDP_2) peut ainsi être utilisée pour obtenir plus rapidement les valeurs optimales de la relaxation linéaire (LP_2) (voir plus de détails dans [100] en annexe).

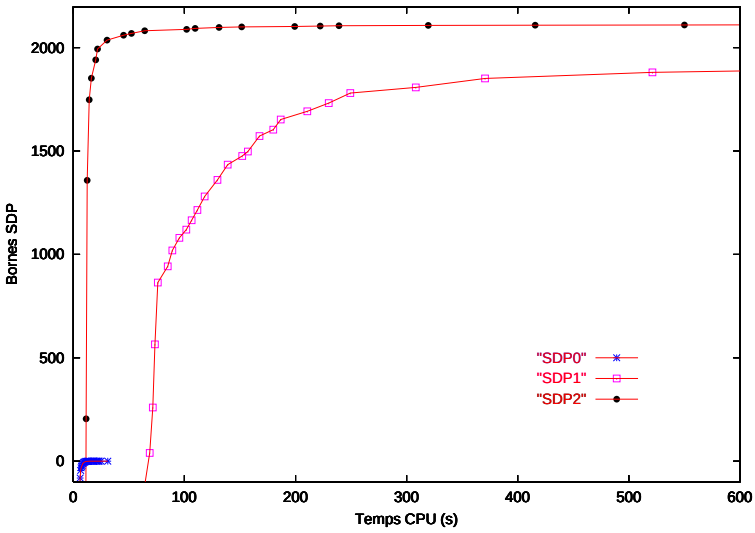

Fig. 8.2. *Convergence du logiciel SB lors du calcul de S0, S1, et S2 pour une instance où N=20, P=5. La valeur optimale vaut ici 2316.*

8.4 QAP : le problème de l'affectation quadratique

Le problème de l'affectation quadratique (QAP) est l'un des problèmes les plus difficiles et les plus étudiés de l'optimisation combinatoire. Ce problème ayant une grande importance pratique [50], de nombreuses approches par programmation mathématique ont été étudiées pour obtenir des bornes inférieures. On peut citer la programmation linéaire [1, 2, 27, 86], quadratique convexe [5, 6], semidéfinie [31, 75, 95, 100, 111], et plus récemment des algorithmes de coupes utilisant la programmation semidéfinie [11, 12, 49]. Une base d'instances variées est disponible sur internet [32], et permet donc de tester expérimentalement une approche. La formulation standard pour les instances purement quadratiques du QAP est :

$$(QAP)\begin{cases} \min \sum_{i,j,k,l} C_{ijkl}\, x_{ij} x_{kl} & \\ \text{s.c.} \sum_{i=1}^{n} x_{ij} = 1 & j = 1,\dots,n \\ \sum_{j=1}^{n} x_{ij} = 1 & i = 1,\dots,n \\ x \in \{0,1\}^{n^2} & \end{cases}$$

8.4.1 Bornes par programmation semidéfinie

Il existe plusieurs relaxations semidéfinies pour ce problème dans la littérature. Les premières bornes obtenues par cette approche ont été présentées dès 1995 dans la thèse de S. Karisch [75]. A cette époque, les méthodes (de point intérieur) utilisées pour résoudre numériquement les SDP nécessitaient de disposer de points strictement réalisables dans le primal et le dual pour garantir leur stabilité numérique. De plus, il était impossible de résoudre de grandes instances, et malgré une étude poussée pour implémenter efficacement l'algorithme de résolution, certaines bornes proposées dans [75] n'étaient pas à l'époque calculables en pratique. Certaines contraintes (trop nombreuses) ont donc été relâchées et réintroduites sous forme de coupes, et les résultats numériques de [75] correspondent donc à des bornes parfois nettement inférieures aux valeurs optimales réelles des SDP proposés. L'arrivée de nouveaux outils numériques ne régla pas totalement cet état de fait car le réglage des paramètres de ces derniers est parfois délicat et leur convergence parfois très longue en fin de résolution. De plus, les auteurs développant (ou adaptant) parfois leur algorithme de résolution et choisissant des critères d'arrêt différents, certaines relaxations furent considérées comme meilleures que d'autres ou même difficilement comparables entre elles [31].

Appliquons les résultats du chapitre 5. Nous savons qu'il est possible d'obtenir la valeur de la relaxation lagrangienne partielle de (QAP) où les contraintes d'intégrité $x \in \{0,1\}^{n^2}$ sont écrites $x_{ij}^2 = x_{ij}$ pour tout i et j dans $\{1,\ldots,n\}$, et où les contraintes linéaires ne sont pas relâchées. Il suffit en effet de résoudre le programme semidéfini suivant (il correspond au programme $(SDP)_{\mathfrak{P}}$ du chapitre 5) :

$$(SDP_P)\begin{cases} \min C \bullet Y \\ \text{s.c. } (Af1)\sum_{i=1}^{n} x_{ij} = 1 & j = 1,\ldots,n \\ \quad (Af2)\sum_{j=1}^{n} x_{ij} = 1 & i = 1,\ldots,n \\ \quad (P1)\sum_{i=1}^{n} Y_{ijkl} = x_{kl} & j,k,l \in \{1,\ldots,n\} \\ \quad (P2)\sum_{j=1}^{n} Y_{ijkl} = x_{kl} & i,k,l \in \{1,\ldots,n\} \\ \quad Z = \begin{bmatrix} 1 & x^T \\ x & Y \end{bmatrix} \succcurlyeq 0 \quad \text{et} \quad \mathbf{d}(Y) = x \\ \quad (Y,x) \in S_{n^2} \times \Re^{n^2} \end{cases}$$

Ce SDP, appelé $(QAP_{\overline{R1}})$ dans [75] est considéré par l'auteur mais il est rapidement abandonné car il ne possède pas de point strictement réalisable, ce qui posait des problèmes de résolution numérique à l'époque. Ce n'est pas le cas pour son dual. En effet, nous avons vu au chapitre 4 qu'il suffit de prendre μ, vecteur des variables duales associées à $\mathbf{d}(Y) = x$, suffisamment grand, et r associé à la contrainte $Z_{11} = 1$ suffisamment négatif pour obtenir des solutions strictement réalisables (voir le programme (SD) du chapitre 4). Il n'y a donc pas de saut de dualité entre (SDP_P) et son dual. Pour régler ce problème, S. Karisch propose naturellement de travailler dans l'intérieur relatif de la région admissible de (SDP_P). Il y parvient en considérant la matrice

réelle $\hat{V} = \left[\begin{array}{c|c} 0 & n \\ \hline W & e_{n^2} \end{array}\right]$ de dimension $(n^2+1) \times (n-1)^2 + 1$, où la matrice réelle $n^2 \times (n-1)^2$ W est le produit tensoriel de $V = \begin{bmatrix} I_{n-1} \\ -e_{n-1} \end{bmatrix}$ avec lui-même c'est-à-dire $W = \begin{bmatrix} V & 0 & \dots & 0 \\ 0 & V & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & V \\ -V & \dots & \dots & -V \end{bmatrix}$.

Puis il prend R tel que $\hat{V}R\hat{V}^T = Z = \begin{bmatrix} 1 & x^T \\ x & Y \end{bmatrix}$ pour obtenir :

$$(QAP_{R1}) \begin{cases} \min C \bullet Y \\ \text{s.c. } \hat{V}R\hat{V}^T = \begin{bmatrix} 1 & x^T \\ x & Y \end{bmatrix} \quad \text{et} \quad \mathbf{d}(Y) = x \\ \qquad R \succcurlyeq 0 \qquad R \in S_{(n-1)^2+1} \end{cases}$$

En exprimant (QAP_{R1}) uniquement en fonction de R, on obtient un SDP dont la région admissible d'intérieur non vide. On a (voir le lemme 4.3.6 page 65 et la page 66 de [75]) :

Proposition 7 *(SDP_P) est équivalent à (QAP_{R1}).*

Les matrices $Z = \hat{V}R\hat{V}^T$ admissibles de (QAP_{R1}) satisfont donc les contraintes $(P1)$ et $(P2)$, ainsi que les contraintes d'affectation $(Af1)$ et $(Af2)$. Cette démarche a ensuite été reprise dans [111] puis [95]. Mais les relaxations équivalentes (QAP_{R1}) et (SDP_P) fournissent de très mauvaises bornes. Considérant le fait que pour x admissible de (QAP) xx^T est une matrice de permutation, S. Karisch a utilisé les égalités d'orthogonalité valides $YY^T = Y^TY = I$ pour obtenir les contraintes :

$(Ortho)$: $\text{b}^{\text{o}}\text{diag}(Y) = \sum_{k=1}^{n} Y_{k.k.} = I$ et $\text{o}^{\text{o}}\text{diag}(Y) = \sum_{k=1}^{n} Y_{.k.k} = I$

L'élément de la ligne $(i-1)n+j$ et colonne $(k-1)n+l$ de Y étant Y_{ijkl}, Les notations $Y_{k.k.}$ et $Y_{.k.k}$ correspondent donc aux matrices $n \times n$ correspondantes extraites de Y pour un k donné. Les égalités $(Ortho)$ sont redondantes dans (QAP_{R1}) [75] si on ajoute :

$(Gan) : Y_{ijkl} = 0$ pour (i,j,k,l) dans $J = \Big\{\{1,\dots,n\}^4 \,:\, (i = k \text{ et } j \neq l) \text{ ou } (j = l \text{ et } i \neq k)\Big\}$.

Ces contraintes (appelées poétiquement "Gangster" en référence aux "trous" opérés dans la matrice Z), sont déjà utilisées implicitement dans [86] dans le cadre d'une relaxation linéaire pour limiter la taille de cette dernière. Enfin, S. Karisch propose d'ajouter les inégalités valides $Y_{ijkl} \geq 0$ pour tout i,j,k, et l dans $\{1,\dots,n\}$. Mais devant l'impossibilité de résoudre à l'époque le programme semidéfini obtenu, il introduit ces contraintes comme coupes pour ses tests numériques.

En appliquant l'algorithme du chapitre 6 section 6.4 nous avons proposé dans [100] d'utiliser la relaxation (SDP_{PP}) en la construisant à partir du programme linéaire $(QAPLP)$ présenté dans [2] et utilisé également dans [86] (voir [100] dans l'annexe pour plus de détails).

$$(QAPLP)\begin{cases} \min C \bullet Y \\ \text{s.c. } (Af1)\ \sum_{i=1}^{n} x_{ij} = 1 & j = 1,\dots,n \\ \quad (Af2)\ \sum_{j=1}^{n} x_{ij} = 1 & i = 1,\dots,n \\ \quad (P1')\ \sum_{i=1}^{n} Y_{ijkl} = x_{kl} & j,k,l = 1,\dots,n\,;\, l \neq j \\ \quad (P2')\ \sum_{j=1}^{n} Y_{ijkl} = x_{kl} & i,k,l = 1,\dots,n\,;\, k \neq i \\ \quad (Pos)\ Y_{ijkl} \geq 0 & (i,j,k,l) \in \{1,\dots,n\}^4 - J \end{cases}$$

$$(SDP_{PP})\begin{cases} \min \quad C \bullet Y \\ \text{s.c. } (Af1)\ \sum_{i=1}^{n} x_{ij} = 1 & j = 1,\dots,n \\ \quad (Af2)\ \sum_{j=1}^{n} x_{ij} = 1 & i = 1,\dots,n \\ \quad (P1)\ \sum_{i=1}^{n} Y_{ijkl} = x_{kl} & j,k,l = 1,\dots,n \\ \quad (P2)\ \sum_{j=1}^{n} Y_{ijkl} = x_{kl} & i,k,l = 1,\dots,n \\ \quad (Pos)\ Y_{ijkl} \geq 0 & i,j,k,l = 1,\dots,n \\ \quad Z = \begin{bmatrix} 1 & x^T \\ x & Y \end{bmatrix} \succcurlyeq 0 \quad \text{et} \quad \mathbf{d}(Y) = x \\ \quad (Y,x) \in S_{n^2} \times \Re^{n^2} \end{cases}$$

Il s'agit du programme (SDP_P) auquel on a ajouté les contraintes (Pos). Le "Gangster" n'a pas pris la fuite. En effet, examinons les contraintes $(P1)$ lorsque $l = j$ et $(P2)$ lorsque $i = k$. Puisque $\mathbf{d}(Y) = x$, on obtient $\sum_{i=1|i\neq k}^{n} Y_{ijkj} = 0$ pour tout k et j et $\sum_{j=1|j\neq l}^{n} Y_{ijil} = 0$ pour tout i et l. Les termes de ces sommes étant tous positifs, nous avons retrouvé les contraintes (Gan). Par conséquent les égalités $(Ortho)$ sont également satisfaites par (SDP_{PP}). Dans [100] nous avions également ajouté les contraintes $\sum_{i=1}^{n}\sum_{k=1}^{n} Y_{ijkj} = 1$ pour $j \in \{1,\dots,n\}$ et $\sum_{j=1}^{n}\sum_{k=1}^{n} Y_{ijik} = 1$ pour $i \in \{1,\dots,n\}$ qui améliorent légèrement la convergence de SB [67], l'outil de résolution alors utilisé. Mais rappelons que ces dernières sont redondantes (voir la proposition 6.3 au chapitre 6), et ne sont donc pas essentielles dans notre analyse. De nouveaux tests expérimentaux utilisant un algorithme de lagrangien augmenté pour (SDP_{PP}) ont été publiés récemment dans [31].

$$(QAP_{R3})\begin{cases} \min \quad C \bullet Y \\ \text{s.c. } (Gan)\ Y_{ijkl} = 0 & (i,j,k,l) \in J \\ \quad (Pos2)\ Y_{ijkl} \geq 0 & (i,j,k,l) \in \{1,\dots,n\}^4 - J \\ \quad \hat{V} R \hat{V}^T = \begin{bmatrix} 1 & x^T \\ x & Y \end{bmatrix} \quad \text{et} \quad \mathbf{d}(Y) = x \\ \quad R \succcurlyeq 0 \qquad R \in S_{(n-1)^2+1} \end{cases}$$

De la même façon que (QAP_{R1}) est équivalente à (SDP_P), la relaxation semidéfinie (QAP_{R3}) donnée dans [111] et utilisée de nouveau dans [95] est équivalente à (SDP_{PP}). Notons que (QAP_{R3}) était déjà proposée dès 1995 dans [75] (sous un autre nom). Les auteurs de [95] utilisent également une relaxation plus faible, notée (QAP_{R2}) qui est (QAP_{R3}) sans les contraintes (Pos). Cette relaxation est donc équivalente à (SDP_{PP}) sans les contraintes (Pos) mais avec les contraintes (Gan) (qui ne sont plus couvertes par $(P1)$ et $(P2)$ puisque (Pos) a été retiré). Remarquons que bien que l'algorithme de résolution des SDP utilisé dans [95] soit semblable à celui de SB [67] (méthode "Spectral Bundle" [69, 81]), ces auteurs ont conservé l'approche de S. Karisch en utilisant (QAP_{R3}) et non pas (SDP_{PP}). Dans [31], d'autres auteurs indiquent par ailleurs que le lien de dominance théorique entre ces deux relaxations n'est pas connu. Nous venons de voir qu'elles sont équivalentes.

Ces résultats sont une illustration de ceux du chapitre 5. Nous vérifions que toutes ces relaxations "butent" sur la borne fournie par $(DP)_{QAP}$, la relaxation lagrangienne partielle de (QAP) où : les contraintes $x \in \{0,1\}^{n^2}$ sont formulées $x_{ij}^2 - x_{ij} = 0$ pour tout i et j, les contraintes linéaires ne sont pas relâchées, on a ajouté initialement les contraintes (Pos) i.e. $-Y_{ijkl} \leq 0$.

$$(DP)_{QAP} \qquad \sup_{\omega \geq 0, \mu} \quad \inf_{x \in \Re^{n^2} \mid Ax = b} x^T \left(C + \mathbf{diag}\,(\mu) - \sum_{i,j,k,l} \omega_{ijkl} E^{ijkl} \right) x - \mu^T x$$

μ est le vecteur des variables duales associées à $x_{ij}^2 - x_{ij} = 0$, ω celui associé à (Pos), $Ax = b$ représente les égalités linéaires $(Af1)$ et $(Af2)$, et E^{ijkl} est la matrice $n^2 \times n^2$ nulle partout sauf à la ligne $(i-1)n + j$ et la colonne $(k-1)n + l$ où l'élément vaut 1. Nous avons en effet établi que dans le cas booléen le programme $(SDP)_{\mathfrak{P}}$, qui correspond ici à (SDP_{PP}), est équivalent à (DP), la relaxation lagrangienne partielle du problème où les contraintes linéaires ne sont pas relâchées. Grâce aux résultats du chapitre 6, nous pouvons même proposer une autre relaxation équivalente en utilisant l'approche présentée dans [83]. En effet, nous avons montré que le programme $(SDP)_{\mathfrak{P}}$ était équivalent à $(SDP)_{\mathfrak{C}}$ qui peut se formuler pour (QAP) avec les contraintes (Pos) comme :

$$(SDP_{CP}) \begin{cases} \min & C \bullet Y \\ \text{s.c.} \ (C) & A^T A \bullet Y - 2b^T Ax + b^2 = 0 \\ (Pos) & Y_{ijkl} \geq 0 \qquad\qquad\qquad\qquad i,j,k,l = 1,\ldots,n \\ & \begin{bmatrix} 1 & x^T \\ x & Y \end{bmatrix} \succcurlyeq 0 \quad \text{et} \quad \mathbf{d}(Y) = x \\ & (Y,x) \in S_{n^2} \times \Re^{n^2} \end{cases}$$

Nous pouvons à présent résumer les résultats que nous avons présentés :

Proposition 8 *(SDP_{PP}), (SDP_{CP}) et (QAP_{R3}) sont tous équivalents à $(DP)_{QAP}$.*

Dans le cadre d'une utilisation pratique, il est fondamental d'examiner le comportement numérique des différents programmes semidéfinis considérés dans cette section. Comme remarqué dans [95], l'utilisation du logiciel CSDP [29] n'est pas adaptée ici. Il faut par exemple 1,5 Go de RAM et presque 3 heures sur un PC Pentium IV 2.2 GHz pour obtenir la valeur optimale égale à 568 de (SDP_{PP}) pour l'instance Nug12.

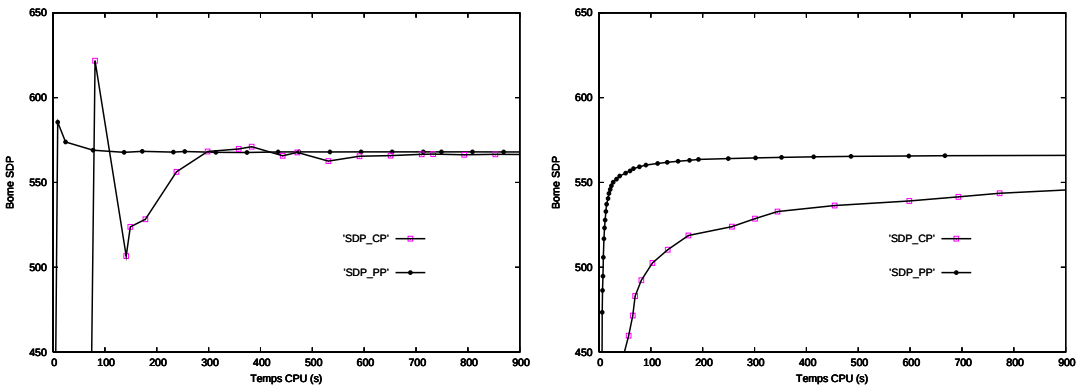

Fig. 8.3. *Instance Nug12 : comparaison des bornes obtenues en fonction du temps CPU pour* (SDP_{PP}) *et* (QAP_{CP}) *avec le logiciel SDPLR (à gauche) et SB (à droite). Leur valeur optimale commune est 568.*

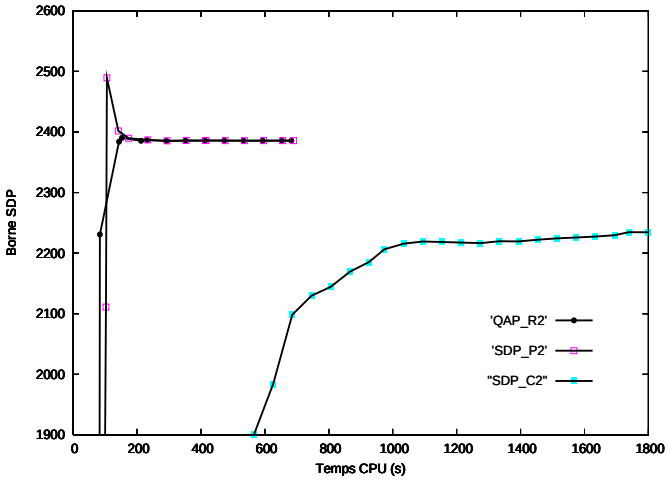

Fig. 8.4. *Instance Nug20 : comparaison des bornes obtenues en fonction du temps CPU pour* (SDP_{P2}), (QAP_{R2}) *et* (SDP_{CP2}) *avec le logiciel SDPLR. Leur valeur optimale commune est 2385,6.*

Dans la figure 8.3 nous avons comparé sur un Pentium 4 2,2 GHz les deux relaxations (SDP_{PP}) et (SDP_{CP}) (équivalentes théoriquement) avec les logiciels SB [67] (méthode "Spec-

tral Bundle"), et SDPLR [30] (fondé sur une méthode de lagrangien augmenté). On constate que la convergence obtenue avec (SDP_{PP}) est bien meilleure dans les deux cas, celle de (SDP_{CP}) étant particulièrement lente avec SB (le tracé est interrompu au bout de 900 secondes mais la convergence s'effectue au bout de nombreuses heures). Dans la figure 8.4, nous avons comparé sur l'instance Nug20 trois relaxations équivalentes : (QAP_{R2}) qui est (QAP_{R3}) sans les contraintes (Pos), (SDP_{P2}) qui est (SDP_{PP}) sans les contraintes (Pos) mais avec les contraintes (Gan), et enfin (SDP_{C2}) qui est (SDP_{CP}) sans les contraintes (Pos) mais avec les contraintes (Gan). Le logiciel SDPLR [30] a été utilisé car c'est le plus efficace pour ces trois relaxations. Pour des raisons de lisibilité le tracé est interrompu au bout d'une demi-heure, mais la relaxation (SDP_{C2}) converge bien vers la même valeur optimale. Les différents tests dans la littérature montrent que (SDP_{PP}) et (QAP_{R3}) ont un intérêt proche dans la pratique [31, 95, 100]. Remarquons en effet qu'en formulant (QAP_{R3}) uniquement en fonction de R, les contraintes (Gan) et $(Pos2)$ perdent leur caractère creux. Cependant la diversité des formulations et l'évolution des outils de résolution numérique permettent de mieux comprendre pourquoi à l'examen de bornes obtenues lors de divers tests expérimentaux (utilisant la plupart du temps des critères d'arrêt différents) certaines de ces relaxations peuvent sembler meilleures que d'autres pourtant théoriquement équivalentes.

Ajouter encore d'autres contraintes redondantes à (QAP) construites à partir des contraintes $Ax = b$ (i.e. $(Af1)$ $(Af2)$) ne conduira à aucune amélioration de la borne fournie par les programmes équivalents (SDP_{PP}), (SDP_{CP}) ou (QAP_{R3}). En effet, les relaxations lagrangiennes totales correspondantes (qui sont équivalentes aux programmes semidéfinis alors construits) ne pourront pas donner de meilleures valeurs que celle de $(DP)_{QAP}$ que nous avons déjà atteinte avec (SDP_{PP}). Rappelons en effet qu'au chapitre 5 nous avons établi que les valeurs des relaxations $(DT)_{\mathfrak{J}}$ sont toujours inférieures ou égales à celle de (DP), et ceci pour tout ensemble de fonctions quadratiques nulles sur $\Omega = \{x \in \Re^n : Ax = b\}$. Nous allons par conséquent nous tourner vers d'autres renforcements (i.e. des inégalités valides) pour obtenir de meilleures bornes.

8.4.2 Un algorithme de coupes pour le QAP fondé sur la SDP

Dans [95], puisque les contraintes (Pos) sont déjà présentes dans (QAP_{R3}) les auteurs proposent de renforcer cette relaxation en utilisant les autres inégalités de linéarisation : $Y_{ijkl} \leq x_{kl}$, et $x_{ij} + x_{kl} - 1 \leq Y_{ijkl}$ pour i, j, k, l dans $\{1, ..., n\}$. Mais les premières sont déjà impliquées par $(P1)$ et $(P2)$ dans (SDP_{PP}) puisque nous avons les contraintes (Pos). Quant aux secondes, elles sont également satisfaites par toute solution admissible (Y, x) de (SDP_{PP}). En effet, en reprenant un résultat démontré dans [23] on a $1 - x_{ij} - x_{kl} + Y_{ijkl} = \sum_{r \neq j} x_{ir} - x_{kl} + Y_{ijkl} = \sum_{r \neq j} x_{ir} - \sum_{r \neq j} Y_{irkl} - Y_{ijkl} + Y_{ijkl} = \sum_{r \neq j} (x_{ir} - Y_{irkl}) \geq 0$. Nous avons donc proposé dans [49] d'utiliser une autre famille de coupes pour renforcer (SDP_{PP}). Puisque dans [100] nous avons construit ce programme semidéfini à partir de $(QAPLP)$, il est naturel de considérer les coupes proposées pour ce programme linéaire dans [27].

Elles consistent en deux familles notées I'_C et I'_L :

$$I'_C \ : \ \sum_{c\in C} Y_{ijlc} \leq \sum_{k\in A} Y_{ijhk} + \sum_{c\in C} \sum_{b\in B, b\neq c} Y_{lchb}$$

où i, h, et l sont des indices de ligne distincts de x, j est un indice de colonne de x, $(A, B, \{j\})$ est une partition de $\{1, ..., n\}$, et $C \subset B$.

$$I'_L \ : \ \sum_{c\in L} Y_{ijcl} \leq \sum_{h\in A} Y_{ijhk} + \sum_{c\in L} \sum_{b\in B, b\neq c} Y_{clbk}$$

où j, k, l sont des indices de colonnes x, i est un indice de ligne de x, $(A, B, \{i\})$ est une partition de $\{1, ..., n\}$, et $L \subset B$. Le problème associé de séparation étant $\mathcal{NP}$-complet [27], mais polynomial lorsque $|C| = 1$, nous avons utilisé l'heuristique proposée dans [27] pour engendrer des coupes avec $|C| > 1$. L'idée est de construire une coupe où $|C| = 1$, puis d'ajouter des indices dans C afin d'obtenir une coupe encore plus violée. Afin d'obtenir des solutions admissibles du problème à la fin de notre algorithme de coupes, nous avons utilisé l'heuristique proposée dans [75]. Soit (Y', x') la solution obtenue à la fin de notre algorithme de coupes. L'idée est de projeter x' sur l'ensemble des solutions admissibles du QAP. Cela revient à minimiser $\|x' - x\|_F = \sqrt{\text{Trace}(x'x)}$ où x décrit l'ensemble des matrices de permutation. On a $\|x' - x\|_F^2 = \|x'\|_F^2 + \|x\|_F^2 - 2\text{Trace}(x'x) = 2n^2 - 2\sum_{i=1}^n \sum_{j=1}^n x'_{ij} x_{ij}$. Ce problème d'affectation linéaire est résolu en quelques secondes par CPLEX [39] même pour des instances de grande taille. Nous avons ainsi prouvé l'optimalité de nos solutions pour de nombreuses instances. Nos résultats expérimentaux sont présentés dans [49] en annexe de ce document. Nous en donnons la synthèse dans le tableau 8.1 (les instances sont celles de la QAPLIB [32]).

Problèmes	$(QAPLP)$	LP CUT	(SDP_{PP})	SDP CUT
Chr	0.7%	0%	4.4%	0.2%
Had	3.6%	1.2%	0%	0%
Nug	11.1%	4.7%	1.4%	0.8%
Scr	7.4%	3.2%	1.4%	1.3%
Rou	8.1%	6.0%	3.9%	3.7%
Tai	8.0%	5.8%	3.6%	2.9%
Esc	48.4%	27.3%	7.9%	7.9%

Tab. 8.1. *Erreur moyenne pour différentes familles d'instances du QAP. PL CUT et SDP CUT sont les algorithmes de coupes fondés respectivement sur $(QAPLP)$ et (SDP_{PP}).*

Un phénomène paradoxal au premier abord est l'accélération de la convergence du logiciel SB [67] utilisé pour la résolution des programmes semidéfinis lorsqu'on ajoute les coupes à la relaxation semidéfinie de départ (SDP_{PP}). En effet, la borne est bien sûr améliorée, mais en

plus l'ancienne valeur est atteinte en un temps plus court que celui nécessaire à la résolution de (SDP_{PP}). Puisqu'à chaque étape nous engendrons nos coupes après avoir résolu seulement approximativement le SDP courant, le temps total d'exécution de notre algorithme de coupes peut donc être inférieur au temps de résolution de (SDP_{PP}). Il peut donc être plus avantageux en temps de calcul pour une borne donnée d'exécuter un certain nombre d'itérations de l'algorithme de coupes que de résoudre un seul programme semidéfini.

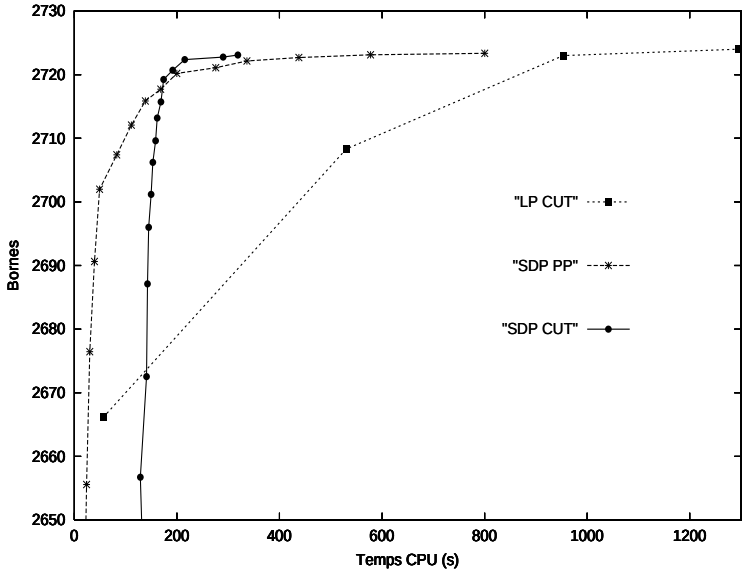

Fig. 8.5. *Instance Had14 : comparaison des bornes obtenues en fonction du temps CPU pour* (SDP_{PP}) *et les algorithmes de coupes fondés sur les programmations linéaire (LP CUT) et semidéfinie (SDP CUT)*

Nug20(opt=2570)	2182	2292	2478	2503	2511
$(QAPLP)$	38mn9s	-	-	-	-
PL CUT	38mn9s	21h48mn	-	-	-
(SDP_{PP})	1mn38s	2mn19s	23mn28s	14h45mn	-
SDP CUT	1mn38s	2mn19s	23mn28s	3h35mn	15h36mn

Tab. 8.2. *Instance NUG20 du QAP : temps CPU pour atteindre quelques valeurs données. PL CUT et SDP CUT sont les algorithmes de coupes fondés respectivement sur* $(QAPLP)$ *et* (SDP_{PP})

Ce point est illustré dans la figure 8.5 où sont tracées : la courbe de convergence de (SDP_{PP}), celle du deuxième (et dernier) programme semidéfini de notre algorithme de coupes (translaté du temps passé à résoudre approximativement (SDP_{PP})), et enfin celle de l'algorithme de coupe

fondé sur $(QAPLP)$ et proposé dans [27]. Les trois approches fournissent toutes la valeur optimale de l'instance (Had14), mais c'est l'algorithme de coupes fondé sur (SDP_{PP}) qui fournit le plus rapidement cette valeur. Nous parlons ici du temps *total* d'exécution de l'algorithme de coupes et non pas du temps de résolution du dernier SDP. Plus généralement, nos tests ont montré que pour la majorité des instances, et cela quelle que soit la qualité de la borne souhaitée, l'approche semidéfinie (simple ou algorithme de coupes) est la meilleure. Cette dernière fournit toujours en un temps plus court les bornes obtenues par programmation linéaire. Le tableau 8.2 en donne un autre exemple. En particulier, la courbe de convergence de la borne SDP est souvent au-dessus de celle de la convergence de la borne par programmation linéaire. Ce phénomène est déjà mentionné dans [100] pour les relaxations (SDP_{PP}) et $(QAPLP)$ et est illustré dans la figure 8.6.

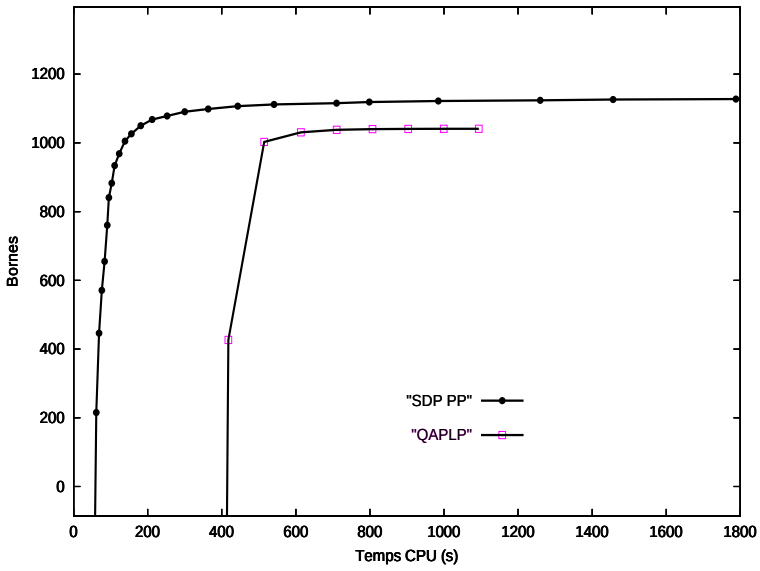

Fig. 8.6. *Instance NUG15 : Courbes de convergence de CPLEX pour* $(QAPLP)$ *et de SB pour* (SDP_{PP})

8.5 Conclusion

L'interprétation lagrangienne décrite au chapitre 5 permet de comparer facilement les différentes relaxations semidéfinies proposées dans la littérature, et même d'en proposer de nouvelles. Elle guide également leur élaboration et évite des recherches infructueuses lorsque la valeur de la relaxation lagrangienne partielle du programme considéré est déjà atteinte. D'un autre côté, les résultats du chapitre 6 nous ont permis d'obtenir à chaque fois des relaxations semidéfinies présentant un excellent compromis entre temps de calcul et qualité de la borne. Cela a pu être possible en prenant en compte les travaux déjà effectués en programmation linéaire.

9

Conclusion Générale et perspectives

9.1 Bilan des résultats obtenus

Dans cette synthèse de travaux de recherche, nous avons pu noter l'omniprésence de la programmation mathématique : obtention de bornes, élaboration d'algorithmes exacts ou approchés, et preuves de la validité de ces derniers. On peut dégager de l'ensemble des résultats présentés dans ce document plusieurs idées directrices :

L'usage conjoint de la programmation mathématique et des approches purement combinatoires. La formulation d'un problème combinatoire par un programme mathématique permet non seulement de prouver éventuellement sa polynomialité mais également d'utiliser ce programme pour concevoir ou démontrer la validité d'un algorithme combinatoire.

L'utilisation systématique des solutions des relaxations linéaires et semidéfinies et pas seulement des bornes obtenues pour construire des solutions optimales ou approchées. Si cette démarche est commune en approximation polynomiale, elle est cruciale dans l'approche par programmation semidéfinie pour rentabiliser le temps passé à la résolution numérique des SDP (en particulier dans le cadre d'une résolution exacte). Cette vision utilitaire des techniques algorithmiques issues du domaine théorique de l'approximation au pire cas nous a permis de rendre efficace dans la pratique l'approche semidéfinie.

L'élaboration mécanique de relaxations semidéfinies pour les problèmes quadratiques. Plus précisément, le cadre général de l'approche lagrangienne pour convexifier un programme quadratique en programme semidéfini, ainsi que les résultats obtenus par l'approche par programmation linéaire nous ont guidés pour construire de telles relaxations. Les bénéfices d'une telle démarche sont à la fois théoriques (obtenir directement d'excellentes bornes, comparer facilement différentes relaxations), et pratiques (utilisation efficace des outils numériques de résolution).

La prise en compte de la résolution numérique des programmes semidéfinis lors de leur élaboration comme relaxations. Considérant les rapides progrès des outils numériques disponibles ainsi que leurs spécificités, une vue purement primale des relaxations peut conduire à des résultats pratiques très différents. En effet, les algorithmes de résolution travaillant dans le primal et/ou le dual, ces derniers peuvent fortement influencer le choix des relaxations semidéfinies (pourtant parfois équivalentes) et même leur élaboration (existence de points intérieurs, existence de solutions optimales, vitesse de convergence).

La reconnaissance de la spécificité de la programmation semidéfinie par rapport à la programmation linéaire. Nous avons pu constater qu'ajouter des contraintes (même redondantes) à un programme semidéfini peut conduire à des temps de résolution plus courts avec certains logiciels de résolution numérique. Nous avons également montré que la programmation semidéfinie peut permettre d'obtenir les bornes atteintes par d'autres approches (et en particulier la programmation linéaire) en un temps inférieur. Ces deux phénomènes peuvent paraître paradoxaux compte tenu de la "lenteur" habituellement constatée lors de la résolution numérique des SDP. Elles permettent donc d'envisager de nouveaux usages de cette approche. En particulier la mise en oeuvre d'algorithmes de coupes fondés sur la programmation semidéfinie est une voie très prometteuse.

9.2 Perspectives et voies de recherche

Les travaux concernant les problèmes de multiflots entiers et de multicoupes présentés dans le chapitre 3 sont actuellement poursuivis dans le cadre de la thèse de Nicolas Derhy co-encadrée par Marie-Christine Costa, Professeur au CNAM, et moi-même. Nous étudions en particulier le problème de la multicoupe lorsque le nombre d'arêtes dans la coupe est limité. Ces problèmes avec contrainte de cardinalité sont particulièrement difficiles, et très peu de travaux existent sur le sujet. Les approches primal-dual fondées sur la programmation linéaire et également la programmation dynamique nous ont déjà permis d'obtenir des algorithmes polynomiaux dans des graphes particuliers [14]. D'autre part, nous nous intéressons à la résolution exacte de problèmes de multicoupes et de multiflots entiers dans les graphes quelconques (ainsi qu'à d'autres problèmes connexes tels que Min-k-Cut et Min-Equipartition). Nous essayons en particulier de déterminer une approche efficace par programmation semidéfinie pour ce type de problèmes.

Au chapitre 5, nous avons donné une caractérisation complète des fonctions quadratiques constantes sur une variété affine, ce qui nous a permis de convexifier en programme semidéfini un programme quadratique soumis à des contraintes linéaires. D'autres applications ou extensions à des cas plus difficiles de ce résultat sont possibles. En particulier, on pourrait chercher à caractériser certaines familles de fonctions constantes sur un domaine plus restreint qu'une simple variété affine. L'ensemble des fonctions possibles sera donc plus important mais moins facile à

déterminer. Cela permettrait d'élaborer de nouvelles relaxations pour les problèmes considérés, et donnerait une meilleure compréhension des relaxations existantes.

Concernant les résultats du chapitre 6, une extension naturelle de nos approches de construction de relaxations semidéfinies développées pour les programmes quadratiques à variables bivalentes consiste en leur généralisation à des programmes à variables entières ou mixtes. Le gain apporté par les relaxations semidéfinies proposées est directement lié à l'effet de levier de la contrainte $X - xx^T \succcurlyeq 0$ qui lie l'ensembles de variables (initiales et de linéarisation). Cela entraîne de fait un nombre de variables important. Mais on peut également utiliser la SDP pour représenter géométriquement différentes valeurs ou sous-ensembles par un polyèdre régulier comme cela a déjà été proposé pour le problème de la coloration d'un graphe [74] ou pour le problème K-MAX-CUT [56]. L'idée peut être généralisée pour représenter différentes valeurs entières ou pour diminuer le nombre de variables en les agrégeant dans un unique vecteur. Cette voie a été explorée dans des cas particuliers mais mérite d'être développée dans un cadre plus général.

Sur le plan pratique, les possibilités sont nombreuses puisque la recherche d'algorithmes et d'implémentations efficaces pour résoudre les programmes semidéfinis est un domaine encore jeune et très actif (voir par exemple [107]). Nous avons montré dans certains de nos travaux que d'ores et déjà la programmation semidéfinie permettait de résoudre efficacement plusieurs problèmes combinatoires $\mathcal{NP}$-difficiles ou du moins d'obtenir de très bonnes bornes dans un temps raisonnable. Nous avons en particulier présenté un algorithme de coupes fondé sur la programmation semidéfinie pour le problème d'affectation quadratique [49] que nous essayons à présent d'utiliser pour la résolution exacte de plus grandes instances. On peut également envisager des méthodes de séparation/évaluation utilisant à la fois les programmations linéaire, quadratique convexe et semidéfinie en fonction du sommet de l'arbre de recherche, ou utiliser la programmation semidéfinie pour élaborer des relaxations quadratiques convexes de meilleure qualité (les résultats présentés dans [24] en sont une bonne illustration).

Nous avons appliqué nos approches à des problèmes combinatoires fondamentaux en conservant toujours pour objectif la résolution efficace d'instances réelles. Nous sommes convaincus que, outre les applications évidentes de nos algorithmes pour les problèmes d'optimisation dans les télécommunications par exemple, nos approches pourraient être utilisées efficacement dans d'autres domaines : en finance, en classification de données, ou en biologie. En effet, bien que quelques travaux existent pour des problèmes plus appliqués (en allocation de fréquences par exemple) la programmation semidéfinie a été peu utilisée dans un contexte industriel pour l'instant. L'élaboration d'approches et de relaxations semblables à celles présentées dans cette synthèse pour résoudre des formulations plus réalistes ferait ainsi définitivement passer la programmation semidéfinie à un usage aussi commun que celui de la programmation linéaire.

Références

1. W. P. Adams, M. Guignard, P. M. Hahn, and W. L. Hightower. A level-2 reformulation-linearization technique bound for the quadratic assignment problem. *European Journal of Operational Research*. A paraître. Version préliminaire disponible sur http ://www.seas.upenn.edu/˜hahn/. University of Pennsylvania, Systems Engineering Department Report, 2001.
2. W. P. Adams and T. A. Johnson. Improved linear programming-based lower bounds for the quadratic assignment problem. In *Proceedings of the DIMACS Workshop on Quadratic Assignment Problems*, volume 16, pages 43–75. American Mathematical Society, 1994.
3. W. P. Adams and H. D. Sherali. A tight linearization and an algorithm for zero-one quadratic programming problems. *Management Science*, 32(10) :1274–1290, 1986.
4. U. Adamy, C. Ambuehl, R. Sai Anand, and T. Erlebach. Call control in rings. *In Proceedings ICALP. Lecture Notes in Computer Science*, 2380 :788–799, 2002.
5. K. M. Anstreicher and N. Brixius. A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming*, 89 :341–357, 2001.
6. K. M. Anstreicher, N. Brixius, J.-P. Goux, and J. Linderoth. Solving large quadratic assignment problems on computational grids. *Mathematical Programming B*, 91 :563–588, 2002.
7. S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of np-hard problems. *Journal of Computer and System Sciences*, 58(1) :193–210, 1999.
8. Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2) :203–221, 2000.
9. J.P. Aubry, G. Delaporte, S. Jouteau, C. Ritte, and F. Roupin. User guide for SDP_SX, 2005. Documentation du logiciel SDP_SX, l'interface graphique de SDP_S.
10. M. Benaichouche, V.D. Cung, S. Dowaji, B. Le Cun, T. Mautor, and C. Roucairol. Bob : une plate-forme unifiée de développement pour les algorithmes de type branch-and-bound. Technical report, Université de Versaille, Prism, Rapport 1995/12, 1995.
11. W. Benajam. *Relaxations semidéfinies pour les problèmes d'affectation de fréquences dans les réseaux mobiles et de l'affectation quadratique*. Thèse de doctorat en informatique, Université Paris Sud, Orsay, 2005.

12. W. Benajam, A. Lisser, and M. Minoux. Relaxations et calculs de bornes inférieures pour le qap, 2005. Rapport de Recherche RR1417. LRI, Université Paris Sud, Orsay. Disponible à http ://www.lri.fr/Rapports-internes/2005/RR1417.pdf.

13. C. Bentz. *Résolution exacte et approchée de problèmes de multiflot entier et de multicoupe*. Thèse de doctorat en informatique, CNAM, Paris, 2006.

14. C. Bentz, M.-C. Costa, N. Derhy, and F. Roupin. Etude du problème de la multicoupe minimale à cardinalité contrainte, 2006. ROADEF 2006, 6 au 8 février, Lille.

15. C. Bentz, M.-C. Costa, L. Létocart, and F. Roupin. A bibliography on multicut and integer multiflow problems, 2004. Rapport de Recherche CEDRIC 654. Disponible à http ://cedric.cnam.fr.

16. C. Bentz, M.-C. Costa, L. Létocart, and F. Roupin. Minimal multicut and maximal integer multiflow in rings, 2006. Rapport de Recherche CEDRIC 1050. Soumis à Information Processing Letters. Disponible à http ://cedric.cnam.fr.

17. C. Bentz, M.-C. Costa, and F. Roupin. Maximum integer multiflow and minimum multicut problems in uniform grid graphs. *Journal of Discrete Algorithms*, Disponible en ligne. A paraître. 2006.

18. A. Billionnet. Different formulations for solving the heaviest k-subgraph problem. *Information Systems and Operational Res.*, 43(3) :171–186, 2005.

19. A. Billionnet, M. C. Costa, and A. Sutter. An efficient algorithm for a task allocation problem. *Journal of the ACM*, 39(3) :502–518, July 1992.

20. A. Billionnet and S. Elloumi. Placement de tâches dans un système distribué et dualité lagrangienne. *Revue d'Automatique, d'Informatique et de Recherche Opérationnelle (R.A.I.R.O.), série verte*, 26(1) :83–97, 1992.

21. A. Billionnet and S. Elloumi. Placement des tâches d'un programme à structure arborescente sur un réseau de processeurs : synthèse de résultats récents. *Information Systems and Operational Research (INFOR)*, 32(2) :65–86, 1994.

22. A. Billionnet and S. Elloumi. An algorithm for finding the K-best allocations of a tree structured program. *Journal of Parallel and Distributed Computing*, 26(2) :225–232, 1995.

23. A. Billionnet and S. Elloumi. Best reduction of the quadratic semi-assignment problem. *Discrete Applied Mathematics*, 109 :197–213, 2001.

24. A. Billionnet and S. Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem, 2003. Rapport scientifique CEDRIC No 466, 2003. A paraître dans Mathematical Programming. Disponible à http ://cedric.cnam.fr.

25. A. Billionnet and F. Roupin. Linear programming to approximate quadratic 0-1 maximization problems. In *Proceedings of the 35th Southeast ACM conference*, 2-4 avril, Murfreesboro, USA, pages 171–173, 1997.

26. A. Billionnet and F. Roupin. A deterministic approximation algorithm for the densest k-subgraph problem. *International Journal of Operational Research*, 2006. A paraître. Rapport de Recherche CEDRIC 486 disponible à http ://cedric.cnam.fr.

27. A. Blanchard, S. Elloumi, A. Faye, and N. Wicker. Un algorithme de génération de coupes pour le problème de l'affectation quadratique. *INFOR*, 41(1) :35–49, 2003.

28. P. Bonami and M. Minoux. Exact max-2sat solution via lift-and-project closure. *Operations Research Letters*, 34(4) :387–393, 2006.

29. B. Borchers. Csdp, a c library for semidefinite programming. Technical report, Mathematics Department, New Mexico Tech, Socorro, USA. http ://infohost.nmt.edu/˜borchers/csdp.html, 1997.

30. S. Burer and R.D.C. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. logiciel disponible sur http ://dollar.biz.uiowa.edu/˜burer/software/sdplr/. *Mathematical Programming (series B)*, 95(2) :329–357, 2003.

31. S. Burer and D. Vandenbussche. Solving lift-and-project relaxations of binary integer programs. *SIAM Journal on Optimization*, 16(3) :726–750, 2006.

32. R.E. Burkard, S. Karisch, and F. Rendl. QAPLIB - a quadratic assignment problem library. *Journal of Global Optimization.*, 10 :391–403, 1997. http ://www.seas.upenn.edu/qaplib/.

33. M. S. Chern, G. H. Chen, and P. Liu. An lc branch-and-bound algorithm for the module assignment problem. *Information Processing Letters*, 32(2) :61–71, 1989.

34. D.G. Corneil and Y. Perl. Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, 9(1) :7–39, 1984.

35. S. Cosares and I. Saniee. An optimization problem related to balancing loads on sonet rings. *Telecommunication Systems*, 3 :165–181, 1994.

36. M.-C. Costa, J.-L. Crémieu, and F. Roupin. A variable neighborhood search using an interior point descent method for the module allocation problem. In *ECCO XIII, European chapter on combinatorial optimization*, Capri, Italie, 18-20 mai, 2000.

37. M.-C. Costa, L. Létocart, and F. Roupin. A greedy algorithm for multicut and integral multiflow in rooted trees. *Operations Research Letters*, 31(1) :21–27, 2003.

38. M.-C. Costa, L. Létocart, and F. Roupin. Minimal multicut and maximal integer multiflow : a survey. *European Journal on Operations Research*, 162(1) :55–69, 2005.

39. Cplex. *ILOG CPLEX Division.* ILOG CPLEX Division, 889 Alder Avenue, Suite 200, Incline Village, NV 89451.

40. V. D. Cung, W. J. Van Hoeve, and F. Roupin. A parallel branch-and-bound algorithm using a semidefinite programming relaxation for the maximum independent set problem. In *ROADEF'2000*, 26-28 Janvier, Nantes, 2000.

41. V. D. Cung and F. Roupin. A parallel branch-and-bound algorithm using a semidefinite programming relaxation for the vertex-cover problem. In *ECCO XII*, Bandol, France, 26-29 Mai, 1999.

42. G.B. Dantzig. On the significance of solving linear programming problems with some integer variables, 1958. The Rand Corporation, document p. 1486.

43. W. Fernandez de la Vega and M. Lamari. The task allocation problem with constant communication. *Discrete Applied Mathematics*, 131(1) :169–177, 2003.

44. G. Delaporte, S. Jouteau, and F. Roupin. SDP_S : a tool to formulate and solve semidefinite relaxations for bivalent quadratic problems. In Proceedings ROADEF 2003, Avignon 26-28 Février 2003. Logiciel et documentation disponibles à http ://semidef.free.fr.

45. J. Dongarra. Performance of various computers using standard linear equations software. disponible à http ://www.netlib.org/utk/people/jackdongarra/papers.htm. Technical Report CS-89-85, University of Tennessee, 2006.

46. S. Elloumi, F. Roupin, and E. Soutif. Comparison of different lower bounds for the constrained module allocation problem. Technical Report 473, CEDRIC, 2003.

47. S. Elloumi and E. Soutif. Comparaison expérimentale de différentes bornes pour un problème de placement de tâches. Technical Report 323, CEDRIC, 2002.

48. A. Faye and F. Roupin. Partial lagrangian relaxation for general quadratic programming. *4'OR, A Quarterly Journal of Operations Research*, 2006. Disponible en ligne. A paraître.

49. A. Faye and F. Roupin. A cutting planes algorithm based upon a semidefinite relaxation for the quadratic assignment problem. In *ESA 2005, Lecture Notes in Computer Science 3669*, pages 850–861, 3-6 Octobre, Majorque, Espagne, 2005.

50. F.Çela. *The Quadratic Assignment Problem : Theory and Algorithms*. Kluwer, Massachessets, USA, 1998.

51. U. Feige and M. X. Goemans. Approximating the value of two prover proof systems, with applications to max 2sat and max dicut. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems, Tel Aviv, Israel*, pages 182–189, 1995.

52. U. Feige and M. Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *Journal of Algorithms*, 41(2) :174–211, 2001.

53. U. Feige and M. Langberg. The rpr^2 rounding technique for semidefinite programs. *In ICALP 2001, Lecture Notes in Computer Science*, 2076 :213–224, 2001.

54. R. Fortet. L'algebre de boole et ses applications en recherche operationelle. *Cahier du Centre d'Etudes de Recherche Opérationnelle*, 4 :5–36, 1959.

55. A. Frank. Disjoint paths in a rectilinear grid. *Combinatorica*, 2 :361–371, 1982.

56. A. Freize and M. Jerrum. Improved approximation algorithms for max k-cut and max bisection. *Algorithmica*, 18 :67–81, 1997.

57. N. Garg, V.V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1) :3–20, 1997.

58. F. Glover. Improved linear integer programming formulations of nonlinear integer problems. *Management Science*, 22 :455–460, 1975.

59. M.X. Goemans and J. Kleinberg. The lovasz theta function and a semidefinite programming relaxation of vertex cover. *SIAM Journal on Discrete Mathematics*, 11 :196–204, 1998.

60. M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum-cut and satisfiability problems using semidefinite programming. *Journal of ACM*, 42(6) :1115–1145, 1995.

61. M. Guignard. Lagrangian relaxation. *Top*, 11(2) :151–228, 2003. Disponible sur http ://top.umh.es/top11201.pdf.

62. Y. Hamam and K. Hindi. Assignment of program modules to processors : A simulated annealing approach. *European Journal of Operational Research*, 122 :509–513, 2000.

63. Q. Han, Y. Ye, and J. Zhang. Approximation of dense-k-subgraph. Technical report, Department of Management Sciences, Henry B. Tippie College of Business, The University of Iowa, Iowa City, IA 52242, USA, 2000.

64. Q. Han, Y. Ye, and J. Zhang. An improved rounded method and semidefinite programming relaxation for graph partition. *Mathematical Programming*, 92(3) :509–535, 2002.

65. R. Hariharan and S. Mahajan. Derandomizing approximation algorithms based on semidefinite programming. *SIAM Journal on Computing*, 28(5) :1641–1663, 1999.

66. R. Hassin and A. Tamir. Improved complexity bounds for location problems on the real line. *Operations Research Letters*, 10 :395–402, 1991.

67. C. Helmberg. *A C++ implementation of the Spectral Bundle Method*. http ://www-user.tu-chemnitz.de/˜helmberg/SBmethod/. Version 1.1.3.

68. C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming A*, 82(3) :291–315, 1998.

69. C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3) :673–696, 2000.

70. C. Helmberg, F. Rendl, and R. Weismantel. A semidefinite programming approach to the quadratic knapsack problem. *Journal of Combinatorial Optimization*, 4 :197–215, 2000.

71. W. J. Van Hoeve. A parallel branch-and-bound algorithm using a semidefinite programming relaxation for the maximum independent set problem. Technical report, Master of Science Université de Twente, Hollande, Laboratoire PRiSM, Versailles. Rapport disponible à http ://www.cs.cornell.edu/˜vanhoeve/, 1999.

72. G. Jäger and A. Srivastav. Improved approximation algorithms for maximum graph partitioning problems. *Journal of Combinatorial Optimization*, 2005. A paraître. Disponible à http ://www.numerik.uni-kiel.de/˜discopt/home/asr/publ.en.html.

73. S. Joy, J.E. Mitchell, and B. Borchers. Solving max-sat and weighted max-sat problems using branch-and-cut. Technical report, Rapport de recherche disponible sur http ://www.rpi.edu/ mitchj/papers.html, 1998.

74. D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45(2) :246–265, 1998.

75. S.E. Karisch. *Nonlinear approaches for the quadratic assignment and graph partition problems*. Phd thesis, Graz University of Technology, Graz, Austria, 1995.

76. B. Korte, L. Lovász, H.J. Prömel, and A. Schrijver. *Paths, Flows and VLSI-Layout*. Springer-Verlag. Berlin, 1990.

77. J. Krarup, D. Pisinger, and F. Plastria. Discrete location problems with push-pull objectives. *Discrete Applied Mathematics*, 123 :363–378, 2002.

78. M. Laurent. A comparison of the sherali-adams, lovász-schrijver and lasserre relaxations for 0-1 programming. *Mathematics of Operations Research*, 28 :470–496, 2003.

79. M. Laurent, S. Poljak, and F. Rendl. Connections between semidefinite relaxations of the max-cut and stable set problems. *Mathematical Programming*, 77 :225–246, 1997.

80. M. Laurent and F. Rendl. Semidefinite programming and integer programming. Technical report, Report PNA-R0210, CWI, Amsterdam. Disponible sur http ://www.optimization-online.org, 2002.

81. C. Lemaréchal. An algorithm for minimizing convex functions. In *Proceedings of IFIP'74 Congress. North Holland*, pages 552–556, 1974.

82. C. Lemaréchal. The omnipresence of lagrange. *4'OR, A Quarterly Journal of Operations Research*, 1 :7–25, 2003.

83. C. Lemaréchal and F. Oustry. *Semidefinite relaxations in combinatorial optimization from a lagrangian point of view. In Advances in Convex Analysis and Global Optimization.* N. Hadjisavvas et P.M. Pardalos, Kluwer, 2001.

84. A. Leroyer. Approche semidéfinie pour programmes quadratiques mixtes. Technical report, Mémoire de fin d'études d'ingénieur IIE, Evry, 2004.

85. L. Létocart. *Problèmes de multicoupes minimales et de multiflots maximaux en nombres entiers.* Thèse de doctorat en informatique, CNAM, Paris, 2002.

86. Y. Li, P. M. Pardalos, K. G. Ramakrishnan, and M. G. C. Resende. Lower bounds for the quadratic assignment problem. *Annals of Operations Research*, 50 :387–411, 1994.

87. L. Lovász. On the shannon capacity of a graph. *IIIE Transactions on Information Theory*, 25 :1–7, 1979.

88. L. Lovász and A. Schrijver. Cones of matrices and set functions and 0-1 optimization. *SIAM Journal on Optimization*, 1 :166–190, 1991.

89. D.G. Luenberger. *Linear and nonlinear programming.* Addison Wesley, 1989.

90. P.M. Pardalos, M.G.C. Resende, and J. Rappe. An exact parallel algorithm for the maximum clique problem. In R. De Leone et al., editor, *High performance algorithms and software in nonlinear optimization.* Kluwer Academic Publishers, 1998.

91. A.L. Peressini, F.E. Sullivan, and J.J. Uhl Jr. *The mathematics of nonlinear programming.* Undergraduate Texts in Mathematics, Springer Verlag, 1988.

92. S. Poljak, F.Rendl, and H. Wolkowicz. A recipe for semidefinite relaxation for (0,1)-quadratic programming. *Journal of Global Optimization*, 7 :51–73, 1995.

93. P. Raghavan. Probabilistic construction of deterministic algorithms. approximate packing integer problems. *Journal of Computer and System Sciences*, 37(2) :130–143, 1988.

94. P. Raghavan and C. Thompson. Randomized rounding : a technique for provably good algorithms and algorithmic proofs. probabilistic construction of deterministic algorithms. *Combinatorica*, 7 :365–374, 1987.

95. F. Rendl and R. Sotirov. Bounds for the quadratic assignment problem using the bundle method. *Mathematical Programming B*, 2003. A paraître. Disponible à Optimization-online.org.

96. F. Roupin. *Approximation de programmes quadratiques en 0-1 soumis à des contraintes linéaires. Application aux problèmes de placement et de partition de graphes.* Thèse de doctorat en informatique, CNAM, Paris, 1996.

97. F. Roupin. A fast heuristic for the module allocation problem. In *15th IMACS World Congress,* 24-29 Août, Berlin, Allemagne, volume 1, pages 405–410, 1997.

98. F. Roupin. On approximating the memory-Constrained Module Allocation Problem. *Information Processing Letters*, 61(4) :205–208, 1997.

99. F. Roupin. Résolution de max 2sat par programmation semidéfinie. In *Francoro III*, 9-12 mai, Québec, Canada, 2001.

100. F. Roupin. From linear to semidefinite programming : an algorithm to obtain semidefinite relaxations for bivalent quadratic problems. *Journal of Combinatorial Optimization*, 8(4) :469–493, 2004.

101. F. Roupin. L'approche par programmation semidéfinie en optimisation combinatoire. *Bulletin de la Société Française de Recherche Opérationnelle et d'aide à la décision*, 13 :7–11, Décembre 2004.

102. A. Schrijver. A compararison of the delsarte and lovász bounds. *IIIE Transactions on Information Theory*, 25 :425–429, 1979.

103. N.Z. Shor. Quadratic optimization problems. *Originally published in Tekhnicheskaya Kibernetika, No 1, 1987. Soviet Journal of computer Systems Sciences*, 25 :1–11, 1987.

104. A. Srivastav. *Derandomization in Combinatorial Optimization. In : Handbook of Randomized Computing.* Pardalos, Rajasekaran, Reif, Rolim (eds.) Kluwer Academic Publishers., Danvers, MA, USA, 2001.

105. A. Srivastav and K. Wolf. Finding dense subgraphs with semidefinite programming. In *International Workshop on Approximation'98. Lecture Notes in Computer Science*, volume 1444, pages 181–191. Springer, 1998.

106. A. Srivastav and K. Wolf. Erratum on finding dense subgraphs with semidefinite programming. preprint. Technical report, Mathematisches Seminar, Universitaet zu Kiel, 1999.

107. M. Stingl. *On the Solution of Nonlinear Semidefinite Programs by Augmented Lagrangian Methods.* Phd thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2005.

108. M.J. Todd. Semidefinite optimization. *Acta Numerica*, 10 :515–560, 2001.

109. L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38 :49–95, 1996.

110. H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors. *Handbook of semidefinite programming, Theory, Algorithms, and applications.* Kluwer Academic Publishers, 2000.

111. Q. Zhao, S. E. Karish, F. Rendl, and H. Wolkowicz. Semidefinite programming relaxations for the quadratic assignment problem. *Journal of Combinatorial Optimization*, 2(1) :71–109, 1998.