Conservatoire National des Arts et Métiers
CEDRIC

THESE

Pour l'obtention du titre de
DOCTEUR EN INFORMATIQUE

Reformulations quadratiques convexes pour la programmation quadratique en variables 0-1

Marie-Christine PLATEAU

Directeurs de thèse : Alain Billionnet
Professeur à l'Ecole Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise

Sourour Elloumi
Maître de Conférences au Conservatoire National des Arts et Métiers

Rapporteurs : Abdel Lisser
Professeur à l'Université Paris XI

Philippe Michelon
Professeur à l'Université d'Avignon et des Pays de Vaucluse

Examinateurs : Fabio Tardella
Professeur à l'Université de Rome

Leo Liberti
Maître de conférences à l'École Polytechnique

Présentée et soutenue publiquement le 27 novembre 2006

Remerciements

Je tiens tout d'abord à remercier Alain Billionnet et Sourour Elloumi, qui m'ont encadrée pendant mes trois années de thèse. Ce fut une expérience passionnante et je tiens à leur exprimer ma profonde reconnaissance pour leur très bon encadrement, leur patience, leurs nombreux conseils et bien sûr, leur gentillesse. Ils ont su m'orienter et me transmettre le goût de la recherche.

Mes remerciements vont également aux rapporteurs de cette thèse, Abdel Lisser et Philippe Michelon, pour avoir accepté de rapporter sur cette thèse et pour l'intérêt qu'ils y ont porté. Merci également aux autres membres du jury qui ont accepté de juger ce travail : Fabio Tardella et Leo Liberti. Je tiens en particulier à remercier Leo Liberti de m'avoir fait confiance, proposé des sujets très intéressants et ouvert à d'autres domaines.

Je remercie également l'ensemble des membres de l'équipe Optimisation Combinatoire (Alain F., Eric, Agnès, Alain B., Sourour, Christophe, Marie-Christine, Frédéric) et plus généralement l'ensemble des personnes du laboratoire CEDRIC du CNAM pour l'ambiance, l'accueil et l'aide fournie. Je remercie plus particulièrement les membres du bureau C.N.A.M., à savoir Cédric, Nicolas et Aurélie pour leur bonne humeur quotidienne, leur grande patience lors de la rédaction de la thèse, leur grand soutien et la très bonne ambiance qui a toujours régné dans notre bureau. Merci également à nos dîners réguliers. Je remercie aussi Tatiana et Jean-Christophe pour la bonne atmosphère lors de nos déjeuners toujours très détendus, pour leur grand soutien, leur implication et leurs aides très fréquentes.

Je remercie également Bernard Lemaire de m'avoir fait confiance en me proposant des groupes d'EDs et pour ses nombreuses anecdotes et conseils avisés. Mes remerciements vont également à Marianne pour son aide toujours très précieuse.

Dans les couloirs du CNAM, de l'IIE et du LIP6, je remercie tous ceux ayant contribué à la bonne atmosphère et aux discussions transdisciplinaires. De façon non exhaustive, je pense en particulier à Vincent, Olivier, Joelle, Joel, Francis, Safia, etc.

Je tiens à remercier les autres doctorants, avec qui nos discussions sur nos expériences communes m'ont été d'un grand réconfort et d'une grande aide. Je pense en particulier à Dominique, Jerôme, Karima, mes amis du DEA IRO avec qui j'ai pu, pour la grande majorité, faire des conférences en leur compagnie : Nathalie, Guillaume et Cathy pour nos mails intensifs et nos soutiens quotidiens, Yasmin pour m'avoir proposé de travailler avec elle sur un sujet très intéressant, Aurélie, Julien et Laure.

Je remercie également Hortense d'avoir toujours été présente et très compréhensive pendant ces trois années. Je pense aussi à Laurence, Claire, Linda, Thialouni, Laurent, Fabienne, Axel, Antoine et tous les autres dont la liste serait trop longue.

Un remerciement très particulier à Lucas, qui a su gérer mon stress et les périodes difficiles de rédaction, des "avant-présentations". Merci pour sa grande patience, ses relectures, son implication dans tout ce que j'ai entrepri, sa bonne humeur ainsi que son aptitude à me réconforter : merci pour tout.

Finalement, toute ma gratitude va à mes parents, sans qui cette thèse n'aurait probablement pu se faire. Merci également à ma soeur Agnès, Laurent, Alexandre et la toute jeune Juliette. Merci à tous de m'avoir soutenue, poussée jusqu'au bout de ce que je voulais faire et de m'avoir aidée de par vos expériences. Merci aussi pour les, ô combien, innombrables conseils que vous m'avez promulgués.

Table des matières

Chapitre 1

Introduction

Parmi tous les problèmes rencontrés par le chercheur, l'ingénieur, le gestionnaire ou l'économiste, les problèmes d'optimisation discrets occupent une place de premier ordre. Ils appararaissent dans divers domaines tels que la médecine, les télécommunications, le secteur économique. Par ailleurs, parmi ces problèmes d'optimisation discrets, un grand nombre, difficiles à résoudre, se formulent aisément comme des programmes mathématiques avec un objectif quadratique, soumis à des contraintes linéaires et en variables 0-1, c'est-à-dire sous la forme :

$$(Q01) \ : \min \left\{ q(x) : Ax = b, A'x \leq b', x \in \{0,1\}^n \right\}$$

Les problèmes de bipartition de graphe, d'affectation quadratique, du sac-à-dos quadratique en sont des exemples très connus et amplement étudiés dans la littérature de l'optimisation combinatoire. On les recontre également, de façon plus concrète, dans les domaines scientifiques, économiques, industriels comme les choix d'investissements, la gestion de portefeuilles, la conception de réseaux, le transport, etc.

Malgré des travaux pionniers datant de plus de vingt ans, les problèmes non linéaires en variables bivalentes demeurent en général mal résolus. Parvenir à résoudre les programmes $(Q01)$ de grande taille représente donc un enjeu très important.

L'optimisation d'une fonction quadratique de variables 0-1 sous contraintes linéaires ou sans contrainte fait l'objet d'une littérature abondante, dont des travaux de synthèse réalisés par Boros et Hammer [29] et Billionnet [16] mais aussi par Hansen, Jaumard et Mathon [74] qui ont étudié la programmation

non linéaire en variables 0-1 en élargissant au cas des contraintes non linéaires. L'intérêt de ces problèmes et leur complexité (ils sont NP-difficiles [61]) ont donc donné lieu au développement de nombreuses heuristiques, souvent performantes, au moins pour certaines classes de problèmes.

Le but de cette thèse est de proposer des méthodes originales et efficaces de *résolution exacte* des programmes quadratiques en variables bivalentes sous contraintes linéaires. Ces problèmes sont généralement non convexes. Plus précisément, leur relaxation continue (consistant à remplacer $x \in \{0,1\}^n$ par $x \in [0,1]^n$) n'est pas un problème convexe.

Pour pallier cette difficulté, notre approche consiste à reformuler un problème d'optimisation quadratique en 0-1 dont la fonction objectif est non convexe en un problème d'optimisation quadratique en 0-1 dont la fonction objectif est convexe. On peut ainsi utiliser les méthodes générales de résolution des programmes quadratiques en variables mixtes, avec un objectif convexe et des contraintes linéaires. Ces méthodes sont implémentées efficacement par des solveurs standards, généralement fondés sur un schéma classique de séparation et d'évaluation progressive (branch-and-bound) et utilisant la borne donnée par relaxation continue.

Notre approche peut être comparée, dans son esprit, à la reformulation de problèmes non linéaires en variables 0-1 par des problèmes linéaires en variables mixtes, technique étudiée depuis longtemps et connue sous le nom de *linéarisation*. Les problèmes ainsi reformulés peuvent alors être résolus par des solveurs standards de programmes linéaires en variables mixtes. Un chapitre de cette thèse est consacré à la présentation d'une nouvelle reformulation linéaire des programmes d'optimisation quadratique en 0-1.

Reformuler $(Q01)$ par un programme quadratique convexe consiste à trouver une fonction quadratique convexe $q'(x)$ telle que $\forall x \in X,\ q'(x) = q(x)$. Ainsi, le problème

$$(Q01') \ : \min \left\{ q'(x) : Ax = b, A'x \leq b', x \in \{0,1\}^n \right\}$$

peut il être soumis à un solveur de programmes quadratiques convexes.

Par un abus de langage, nous appellerons *convexification* de $(Q01)$ toute reformulation de ce programme par un programme quadratique en variables 0-1 dont la fonction objectif est convexe, les contraintes restant linéaires. La méthode principale que nous avons développée et que nous notons **QCR**

(Quadratic Convex Reformulation) est très générale puisqu'elle peut s'appliquer à tout problème quadratique en 0-1. Le principe de QCR est de trouver une convexification de (Q01) dont la borne calculée par relaxation continue est la plus fine possible, parmi un ensemble de convexifications. Les convexifications envisagées consistent à ajouter à $q(x)$ une fonction quadratique nulle sur le domaine admissible. Trouver la meilleure fonction à ajouter, parmi une certaine famille de fonctions, requiert la résolution d'une relaxation semidéfinie du problème initial. On peut considérer ainsi QCR comme un pré-traitement du problème initial, le problème convexifié pouvant alors être résolu par différentes approches. Dans ce travail, nous le résolvons par un solveur standard. La valeur optimale de la relaxation continue du problème convexifié est égale à la valeur optimale de la relaxation semidéfinie du problème initial. Ce pré-traitement par QCR du problème (Q01) permet de capturer, dans une reformulation quadratique convexe, la valeur d'une certaine relaxation semidéfinie de ce problème.

De plus, QCR s'applique aux problèmes déjà convexes. On parlera alors de *reconvexification*. L'intérêt de cette démarche originale réside dans la possible amélioration de la borne inférieure obtenue par relaxation continue directe du problème initial. Nous illustrons cette amélioration par différents exemples.

Nous présentons également des variantes de notre méthode générale. Dans le même esprit que QCR, elles modifient la fonction économique de (Q01) dans le but de trouver une convexification dont la valeur optimale de la relaxation continue est la meilleure possible. Par construction, ces variantes sont moins efficaces que QCR en ce qui concerne la borne inférieure obtenue par relaxation continue, mais les reformulations convexes associées sont généralement plus rapides à obtenir et peuvent ainsi être parfois plus efficaces dans le processus de résolution exacte global. En effet, QCR requiert la résolution d'un programme semidéfini positif, ce qui, dans certains cas, se révèle très coûteux en temps de calcul, notamment lorsque le problème à résoudre est de grande taille. Ainsi, les variantes étudiées, qui sont en fait des prémices de QCR, peuvent parfois la remplacer efficacement : la borne obtenue par relaxation continue est alors moins bonne mais le temps de calcul total de résolution de (Q01) est inférieur grâce à une réduction importante du temps consacré à la convexification.

Une autre approche de résolution de (Q01), fondée sur une reformulation linéaire, est également présentée dans ce manuscrit. L'idée consiste à trouver une reformulation linéaire compacte, c'est-à-dire une reformulation qui ne comporte pas trop de variables et de contraintes. On souhaite, de plus, que la valeur optimale de la relaxation continue de cette reformulation soit de bonne qualité. Nous l'appelons **reformulation linéaire compacte positive** car la fonction économique reformulée s'exprime comme une constante plus des fonctions de variables 0-1, positives ou nulles pour toute solution admissible du domaine relâché $\{Ax = b, A'x \leq b', x \in [0,1]^n\}$. La constante est donc une borne inférieure de la valeur optimale de (Q01). Dans notre reformulation, cette constante correspond à la valeur optimale d'une relaxation continue d'une autre linéarisation de (Q01), donnant généralement une bonne borne inférieure. On peut donc considérer, comme pour QCR, cette reformulation linéaire compacte positive comme un pré-traitement du problème (Q01) permettant de capturer, dans une reformulation linéaire, la valeur optimale d'une autre relaxation linéaire (efficace) du problème initial. La résolution exacte du problème reformulé peut alors se faire à l'aide d'un solveur standard et bénéficie ainsi à la fois de la bonne qualité de la borne obtenue à la racine de l'arborescence de recherche grâce au pré-traitement et de la concision de la reformulation linéaire.

Pour faciliter la mise en œuvre de QCR, nous avons développé un logiciel spécifique, résolvant tout programme quadratique 0-1 sous contraintes linéaires. Ce logiciel, que nous avons appelé QCR, permet à son utilisateur de résoudre exactement un programme de type (Q01) par la reformulation QCR, en lui donnant le choix entre deux logiciels de programmation semidéfinie et deux logiciels de résolution exacte de programmes quadratiques convexes en 0-1. La résolution de la relaxation semidéfinie est effectuée et la reformulation convexe est construite. Enfin, la reformulation, et donc le problème initial, sont résolus par un solveur standard.

Grâce à ce logiciel, nous avons pu tester facilement QCR sur différents problèmes classiques d'optimisation combinatoire. Nous avons tout d'abord considéré des problèmes non convexes : le k-cluster et la bipartition, qui sont des problèmes d'optimisation dans les graphes, la minimisation des échanges d'outils dans le magasin d'une machine-outils. Nous avons également testé

QCR sur des problèmes convexes : un problème d'investissements et un problème de plus court chemin probabiliste. Les résultats que nous avons obtenus sont très encourageants. Pour le problème du k-cluster ou de la bipartition de graphe, QCR permet d'améliorer significativement les résultats de la littérature. Notre méthode, pourtant très générale et basée sur des logiciels standards, se révèle donc très compétitive par rapport à des méthodes spécifiques pour ces problèmes. L'application sur des problèmes déjà convexes permet également de conclure qu'il peut être préférable de reformuler d'abord le problème de départ par QCR plutôt que de le soumettre directement à un solveur. Par ailleurs, nous avons étudié un problème d'ordonnancement, initialement non convexe. Pour ce problème, la méthode QCR s'est révélée peu efficace. En revanche, une des variantes de QCR, appliquée à une reformulation convexe spécifique à ce problème d'ordonnancement et déjà proposée dans la littérature, donne d'assez bons résultats.

Le **chapitre 2** est un état de l'art présentant les principales reformulations quadratiques convexes et linéaires de (Q01). Le **chapitre 3** est consacré à QCR. En particulier, nous présentons les motivations qui nous ont amenés à l'élaboration de cette méthode avant de la décrire en détail. Nous mettons en évidence l'utilisation d'une solution optimale d'une relaxation semidéfinie pour la construction de la convexification QCR. Par ailleurs, nous montrons que QCR procure la meilleure convexification quadratique possible d'un problème particulier : l'optimisation d'une fonction quadratique de variables 0-1 sous une contrainte de cardinalité. Le **chapitre 4** présente plusieurs variantes de QCR : certaines déjà développées dans la littérature et d'autres élaborées lors de ce travail de thèse. Le **chapitre 5** présente une approche originale de reformulation : la reformulation linéaire compacte positive citée plus haut. Nous mettons en parallèle l'aspect de pré-traitement dans cette linéarisation et dans QCR, ce pré-traitement visant dans les deux cas à obtenir une bonne reformulation. Les deux derniers chapitres sont consacrés aux expérimentations. Le **chapitre 6** considère des problèmes quadratiques non convexes alors que **le chapitre 7** traite des problèmes déjà convexes.

Notations

Voici une liste de notations qui seront utilisées au cours de ce manuscrit :

- $(Q01)$: problème quadratique sous contraintes linéaires en variables 0-1
- $(\overline{Q01})$: relaxation continue du problème $(Q01)$ qui consiste simplement à relâcher les contraintes d'intégrité de X
- $X = \{x : Ax = b, A'x \leq b', x \in \{0,1\}^n\}$: ensemble des solutions réalisables de $(Q01)$
- $\overline{X} = \{x : Ax = b, A'x \leq b', x \in [0,1]^n\}$: ensemble des solutions réalisables de la relaxation continue de $(Q01)$
- $v(Q01)$: valeur optimale du problème $Q01$
- O : matrice de dimension $m \times n$ dont les coefficients sont tous nuls
- e_n : vecteur unité de dimension n
- $Diag(u)$: matrice diagonale de dimension $n \times n$ où la diagonale n'est autre que le vecteur $u \in \mathbb{R}^n$
- S_n : ensemble des matrices réelles symétriques de dimension $n \times n$
- $\lambda_{\min}(Q)$: plus petite valeur propre de la matrice Q
- $MIQP$: Mixed Integer Quadratic Programming
- MIP : Mixed Integer Programming
- QCR : Quadratic Convex Reformulation
- $EQCR$: Eigenvalue Quadratic Convex Reformulation
- $IQCR$: Integrity Quadratic Convex Reformulation

Chapitre 2

Reformulations linéaires et quadratiques convexes : état de l'art

Un problème quadratique en variables bivalentes sous contraintes linéaires consiste à minimiser une fonction quadratique pseudo-booléenne sous la contrainte que les variables du problème doivent prendre des valeurs binaires dans $\{0,1\}$ et satisfaire un ensemble de contraintes linéaires, d'égalité et/ou d'inégalité. Une fonction pseudo-booléenne quadratique peut s'écrire sous la forme $q(x) = \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} x_i x_j$.

Le problème $(Q01)$ qui nous intéresse est le suivant :

$$
\begin{aligned}
(Q01): \quad \min \quad & q(x) = \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} x_i x_j \\
\text{s.c.} \quad & \\
& \sum_{i=1}^{n} a_{ki} x_i = b_k \qquad k = 1, \ldots, m \\
& \sum_{i=1}^{n} a'_{\ell i} x_i \leq b'_\ell \qquad \ell = 1, \ldots, p \\
& x_i \in \{0,1\} \qquad i = 1, \ldots, n
\end{aligned}
$$

On rappelle (voir la section Notations) que :
$X = \{x : Ax = b, A'x \leq b', x \in \{0,1\}^n\}$ et
$\overline{X} = \{x : Ax = b, A'x \leq b', x \in [0,1]^n\}$

Sans perte de généralité, on suppose que tous les termes diagonaux de Q sont nuls. On suppose également que Q est symétrique. Si ce n'est pas le cas, Q peut se réécrire ainsi : $\frac{Q+Q^t}{2}$.

La complexité des problèmes (Q01) a donné lieu au développement de nombreuses heuristiques, souvent performantes. Beaucoup d'entre elles ont été proposées pour des instances particulières de (Q01) : par exemple des recherches tabou pour le cas non contraint [13], [64], [106], [109] ou des recherches locales pour le problème de bipartition [87], [91]. Pour le cas non contraint, la méthode de décomposition de Chardaire et Sutter [37] est une approche qui s'est révélée très efficace en terme de taille des instances traitées (100 variables et pour toute densité) : leur méthode fournit une solution approchée avec une bonne garantie.

Dans cette section, nous allons nous intéresser aux méthodes exactes développées pour la résolution de (Q01) et fondées sur des reformulations. Les méthodes exactes actuelles constituent deux grandes classes : les approches par reformulation linéaire (section 2.2) et par reformulation quadratique convexe (section 2.1). Nous appelons *reformulation* de (Q01) sa réécriture soit sous forme d'un programme linéaire en variables mixtes, soit sous forme d'un programme quadratique convexe en variables 0-1. Dans les deux cas, ces reformulations préservent le domaine des solutions admissibles et la valeur optimale. Les plus connues, et sûrement les plus utilisées, sont les linéarisations, qui consistent à reformuler la fonction objectif de (Q01) en une fonction linéaire par l'ajout de variables et de contraintes appropriées. La valeur optimale de la relaxation continue du problème transformé peut ainsi fournir une borne au problème en 0-1. La programmation quadratique convexe, quant à elle, garde le caractère quadratique de l'objectif. Si la fonction économique est déjà convexe, on peut soumettre directement le programme à un solveur MIQP par exemple. Si elle ne l'est pas, alors l'idée est de modifier l'objectif tel que le nouveau problème quadratique soit convexe. Et ainsi, la résolution peut se faire via des solveurs appropriés.

2.1 Reformulations quadratiques convexes

2.1.1 La méthode de la plus petite valeur propre, Hammer et Rubin, 1970

Dans leur article [72], Hammer et Rubin étudient quelques propriétés générales des programmes quadratiques en variables 0-1. Ils contribuent aux prémices de la reformulation quadratique convexe et décrivent de manière théorique une reformulation de $q(x)$ utilisant les contraintes d'intégrité et le calcul de valeurs propres.

En utilisant la propriété $x_i^2 = x_i$, on peut montrer facilement que la fonction économique $q(x)$ de $(Q01)$ peut se réécrire en perturbant la diagonale de la matrice Q par une quantité γ :

$$q_\gamma(x) \;=\; x^t(Q+\gamma I)x - \gamma\sum_{i=1}^{n} x_i$$

où $\gamma \in \mathbb{R}$ et I représente la matrice identité de dimension $n \times n$. Il est clair que $q_\gamma(x) = q(x)\ \forall x \in \{0,1\}^n$.

On peut donc construire une famille de fonctions $q_\gamma(x)$, toutes égales à $q(x)\ \forall\gamma \in \mathbb{R}$ et $\forall x \in X$. L'idée est de rendre l'objectif convexe et donc de choisir les paramètres γ tels que la nouvelle matrice $(Q+\gamma I)$ soit semidéfinie positive, autrement dit $\lambda_{\min}(Q+\gamma I) \geq 0$ (voir l'Annexe 1). Hammer et Rubin montrent que pour tout $\gamma \geq -\lambda_{\min}(Q)$, la fonction $q_\gamma(x)$ est convexe. Ils montrent aussi que comme $\min_{x\in\overline{X}}\{q_\gamma(x)\}$ est une fonction décroissante en γ, si $\gamma = -\lambda_{\min}(Q)$ alors la valeur optimale de la relaxation continue associée est la plus grande possible.

On étudiera plus en détail la réécriture de Hammer et Rubin à la section 4.1. On peut noter qu'elle modifie seulement l'objectif et aucune contrainte ni variable n'est ajoutée au problème de départ.

2.1.2 Une reformulation quadratique convexe pour les problèmes quadratiques en 0-1 sans contraintes, Billionnet et Elloumi, 2004

Billionnet et Elloumi [18] s'intéressent aux programmes quadratiques en 0-1 sans contraintes. Ils présentent, dans ce travail, une méthode de pré-

traitement visant à convexifier, non pas ($Q01$), mais une fonction pseudo-booléenne quadratique. Cette méthode consiste à déterminer un vecteur $u \in \mathbb{R}^n$ via la programmation semidéfinie et à ajouter à l'objectif la quantité $\sum_{i=1}^{n} u_i \left(x_i^2 - x_i\right)$, qui est nulle sur le domaine admissible ($x_i^2 = x_i \ \forall i = 1, \ldots, n$). Le vecteur u convexifiant l'objectif n'est autre que le vecteur des variables duales optimales des contraintes (2.1) du programme semidéfini suivant :

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} X_{ij} \\ \text{s.c.} \quad & \\ & X_{ii} = x_i \qquad i = 1, \ldots, n \\ & \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\ & x \in \mathbb{R}^n, X \in S_n \end{aligned} \tag{2.1}$$

Comme nous le verrons dans le chapitre 4, de façon triviale, cette approche se généralise au cas des problèmes sous contraintes puisque l'on peut ne pas tenir compte des contraintes dans le processus de convexification.

Tout comme la méthode de la plus petite valeur propre, aucune contrainte ou variable n'est ajoutée mais la phase de pré-traitement peut bien sûr se révéler être plus coûteuse en temps puisqu'elle nécessite la résolution d'un programme semidéfini.

2.1.3 La reformulation quadratique convexe de Carter, 1984

Carter [35] décrit une transformation de ($Q01$) en un nouveau problème quadratique convexe de même taille. Cette approche se concentre sur la reformulation de la fonction objectif via une transformation adéquate de la matrice Q. Tous les termes diagonaux de Q sont perturbés par l'ajout d'une quantité ϕ_i et on construit ainsi une matrice $\bar{Q}$ telle que $\bar{q_{ii}} = q_{ii} + \phi_i$ tout en compensant avec les termes linéaires $c_i = c_i - \phi_i$. Ainsi, un nouveau problème quadratique ($\bar{Q}01$) est créé. Reste à choisir les valeurs de ϕ_i telles que $\bar{Q}$ soit semidéfinie positive. Pour cela, Carter développe une variante de la factorisation de Cholesky [62] qui produit une nouvelle matrice avec des

termes diagonaux de petites valeurs et garantit que le problème résultant est bien défini positif. Dans son approche, cette factorisation est très importante car, non seulement elle permet de rendre l'objectif convexe, mais aussi, elle sera utilisée comme initialisation des autres techniques développées par l'auteur. En effet, après la factorisation modifiée de Cholesky, Carter affine les termes diagonaux de manière à ce que la solution optimale de la relaxation continue de ($\bar{Q}01$) comporte des valeurs entières, tout en gardant, bien sûr, le caractère semidéfini de $\bar{Q}$. En particulier, il étudie l'effet produit sur la valeur optimale d'une diminution d'un seul élément diagonal et développe ainsi un algorithme perturbant chaque élément diagonal de $\bar{Q}$.

Toutes ces reformulations permettent de convexifier les programmes de type ($Q01$) tout en conservant le caractère quadratique de la fonction économique. D'autres méthodes de résolution exacte transforment l'objectif quadratique en objectif linéaire.

2.2 Reformulations linéaires

Nous appelons *"linéarisation"* les techniques permettant de remplacer la formulation quadratique par une formulation linéaire via l'introduction de variables et de contraintes supplémentaires.

2.2.1 Une reformulation linéaire classique, Fortet, 1959

La linéarisation la plus utilisée et probablement la plus naturelle fut présentée la première fois par Fortet [57],[58]. Elle est parfois appelée linéarisation *classique*. Elle consiste à remplacer chaque produit $x_i x_j$ par une variable additionnelle y_{ij} et à introduire les contraintes suivantes :

$$y_{ij} \leq x_i \qquad 1 \leq i < j \leq n \tag{2.2}$$

$$y_{ij} \leq x_j \qquad 1 \leq i < j \leq n \tag{2.3}$$

$$y_{ij} \geq x_i + x_j - 1 \qquad 1 \leq i < j \leq n \tag{2.4}$$

$$y_{ij} \geq 0 \qquad 1 \leq i < j \leq n \tag{2.5}$$

L'ensemble des contraintes (2.2) à (2.5) forcent la variable y_{ij} à être égale à 0 si et seulement si l'une au moins des deux variables x_i ou x_j est égale à 0. De la même manière, elles forcent y_{ij} à être égale à 1 si et seulement si les deux variables x_i et x_j sont fixées à 1. La reformulation linéaire résultante est alors :

$$
\begin{aligned}
(RL): \quad \min \quad & F(x) = \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} y_{ij} \\
\text{s.c.} \quad & \\
& \sum_{i=1}^{n} a_{ki} x_i = b_k \qquad k = 1, \ldots, m \qquad (2.6) \\
& \sum_{i=1}^{n} a'_{\ell i} x_i \leq b'_\ell \qquad \ell = 1, \ldots, p \qquad (2.7) \\
& (2.2), (2.3), (2.4), (2.5) \\
& y_{ij} = y_{ji} \qquad i < j \qquad (2.8) \\
& x \in \{0,1\}^n
\end{aligned}
$$

La reformulation linéaire classique ajoute donc $\mathcal{O}(n^2)$ variables et $\mathcal{O}(n^2)$ contraintes.

2.2.2 La méthode RLT, Sherali, Adams, 1995

Sherali et Adams [126] montrent qu'en multipliant les inégalités (2.7) de la linéarisation classique (RL) par x_k et $1 - x_k$, les égalités (2.6) par x_k $(1 \leq k \leq n)$ et en remplaçant enfin les produits $x_j x_k$ par les variables y_{jk}, $\mathcal{O}(n \times (m + p))$ contraintes d'égalités et d'inégalités valides sont ajoutées. Ces différentes équations et inéquations contribuent à donner une meilleure valeur optimale de $(\overline{RL})$ et donc une meilleure borne inférieure de (RL).

Sherali et Adams ont donc élaboré une Technique de Reformulation Linéaire, appelée RLT. La procédure RLT est une procédure en deux étapes : une de *reformulation* et l'autre de *linéarisation*. Comme nous l'avons spécifié, elle procède tout d'abord à la multiplication des contraintes du problème et à la définition d'une nouvelle variable y_{ij}. Des contraintes de linéarisation sont nécessaires pour que le remplacement soit valide.

RLT peut être vue comme une hiérarchie de relaxations linéaires. A un niveau donné t, on procède selon les deux étapes RLT, i.e. reformulation et

linéarisation. Au niveau $t+1$, on remultiplie chaque nouvelle contrainte par x_j et $1-x_j$ puis on linéarise de nouveau. Au plus haut niveau n (où n représente le nombre de variables bivalentes), le programme linéaire est exact dans le sens où le domaine admissible est l'enveloppe convexe des solutions du programme non linéaire associé.

Dans notre travail de thèse, nous nous intéressons plus particulièrement au niveau 1 de la procédure RLT qui donne la reformulation linéaire suivante :

$$
\begin{aligned}
\min \quad & F(x,y) = \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} y_{ij} \\
\text{s.c.} \quad & \\
& \sum_{i=1}^{n} a_{ki} x_i = b_k && k = 1,..,m \\
& \sum_{i=1}^{n} a_{ki} y_{ij} = b_k x_j && k = 1,..,m; j = 1,..,n \\
& y_{ij} = y_{ji} && i,j = 1,..,n : j \neq i \\
& y_{ii} = x_i && i = 1,..,n \\
& \sum_{i=1}^{n} a'_{\ell i} x_i \leq b'_\ell && \ell = 1,..,p \\
& \sum_{i=1}^{n} a'_{\ell i} y_{ij} \leq b'_\ell x_j && \ell = 1,..,p; j = 1,..,n \\
& \sum_{i=1}^{n} a'_{\ell i} (x_i - y_{ij}) \leq b'_\ell (1 - x_j) && \ell = 1,..,p; j = 1,..,n \\
& y_{ij} \leq x_i && i,j = 1,..,n : j \neq i \\
& x_i + x_j - y_{ij} \leq 1 && i = 1,..,n; j = i+1,..,n \\
& x_i \in \{0,1\}, y_{ij} \geq 0 && i,j = 1,..,n
\end{aligned}
$$

Cette linéarisation peut être vue comme un renforcement, par des coupes, de la linéarisation classique rappelée en section 2.2.1.

2.2.3 La méthode de Glover, 1975

En 1975, Glover [63] a proposé une méthode de reformulation linéaire qui consiste à remplacer une sélection d'expressions non linéaires de ($Q01$) par

des variables continues. Si on pose $z_j = x_j \sum_{i=1, i\neq j}^{n} q_{ij}x_i$, $(Q01)$ devient :

$$
\begin{aligned}
(RL_g): \quad \min \quad & F(x) = \sum_{i=1}^{n} c_i x_i + \sum_{j=1}^{n} z_j \\
\text{s.c.} \quad & \\
& \sum_{i=1}^{n} a_{ki} x_i = b_k && k = 1, \ldots, m \\
& \sum_{i=1}^{n} a'_{\ell i} x_i \leq b'_\ell && \ell = 1, \ldots, p \\
& L_j x_j \leq z_j \leq U_j x_j && j = 1, \ldots, n \\
& \sum_{i=1, i\neq j}^{n} q_{ij}x_i - U_j(1-x_j) \leq z_j \leq \sum_{i=1, i\neq j}^{n} q_{ij}x_i - L_j(1-x_j) && j = 1, \ldots, n \\
& x \in \{0,1\}^n, z \geq 0
\end{aligned}
$$

où L_j et U_j sont des bornes inférieures et supérieures des fonctions linéaires $\sum_{i=1, i\neq j}^{n} q_{ij}x_i$:

$$L_j = \min \left\{ \sum_{i=1, i\neq j}^{n} q_{ij}x_i : x \in X \right\}$$

$$U_j = \max \left\{ \sum_{i=1, i\neq j}^{n} q_{ij}x_i : x \in X \right\}$$

Par des considérations d'optimalité, puisque les coefficients des variables z_j sont positifs, les inégalités $z_j \leq U_j x_j$ et $z_j \leq \sum_{i=1}^{n} q_{ij}x_i - L_j(1-x_j)$ peuvent être supprimées. $\mathcal{O}(n)$ variables et $\mathcal{O}(n)$ contraintes sont ajoutées lors de la reformulation.

Cette linéarisation est qualifiée de compacte car elle contient un nombre de variables et de contraintes qui est du même ordre de grandeur que le problème de départ.

2.2.4 Une reformulation linéaire compacte, Adams, Forrester et Glover, 2004

Enfin, Adams, Forrester et Glover [2] présentent une linéarisation basée sur la méthode classique de Glover et sur RLT. Leur approche contribue à

deux améliorations : une sur le calcul des bornes L_j et U_j définies par Glover et l'autre sur la réécriture de la fonction objectif.

Tout d'abord, le calcul des bornes U_j et L_j est affiné. Maintenant, quatre bornes supérieures et inférieures sont définies :

$$L_j^1 = \min \left\{ \sum_{i=1, i\neq j}^{n} q_{ij} x_i : x \in X, x_j = 1 \right\}$$

$$U_j^1 = \max \left\{ \sum_{i=1, i\neq j}^{n} q_{ij} x_i : x \in X, x_j = 1 \right\}$$

$$L_j^0 = \min \left\{ \sum_{i=1, i\neq j}^{n} q_{ij} x_i : x \in X, x_j = 0 \right\}$$

$$U_j^0 = \max \left\{ \sum_{i=1, i\neq j}^{n} q_{ij} x_i : x \in X, x_j = 0 \right\}$$

Ensuite, en utilisant les inégalités suivantes :

$$x_i x_j = x_j x_i \quad i < j$$

$$x_i \bar{x_j} = x_i - x_i x_j \quad i \neq j$$

$$\bar{x_j} = 1 - x_j \quad i = 1, \ldots, n$$

on réécrit la fonction objectif :

$$\begin{aligned} q_\alpha(x) &= q(x) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \alpha_{ij}^1 (x_i x_j - x_j x_i) + \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{ij}^2 (x_i \bar{x_j} - x_i + x_i x_j) \\ &\quad + \sum_{i=1}^{n} \alpha_1^3 (\bar{x_i} - 1 + x_i) \end{aligned}$$

Bien sûr, $q_\alpha(x) = q(x) \ \ \forall x \in X$ et un nouveau problème $(Q01_\alpha)$ est créé, utilisant la reformulation de Glover. Il contient $\mathcal{O}(n)$ contraintes supplémentaires.

Il faut maintenant trouver les paramètres $\alpha^1, \alpha^2, \alpha^3$ optimaux, c'est-à-dire ceux pour lesquels la borne inférieure obtenue par relaxation continue de $(Q01_\alpha)$, notée v^*, est maximale. Adams, Forrester et Glover [2] montrent que v^* n'est autre que la valeur optimale de la relaxation continue de RLT de niveau 1 du problème initial et que les paramètres optimaux α^1, α^2 et α^3 correspondent aux valeurs optimales de certaines variables duales associées aux contraintes de cette relaxation continue.

2.3 Un critère de comparaison des reformulations

Une caractéristique commune aux deux schémas de reformulation présentés ci-dessus est qu'ils aboutissent tous les deux à des programmes mathématiques dont la résolution de la relaxation continue est un problème polynomial. Mieux, en préservant un nombre de variables et de contraintes qui est du même ordre de grandeur que celui du problème de départ, on s'assure que le temps de résolution des relaxations continues restera raisonnable. Ainsi peut-on envisager la résolution exacte des problèmes reformulés par une méthode arborescente, basée sur la résolution de la relaxation continue à chaque nœud de l'arborescence de recherche. Pour ce faire, il nous suffit d'utiliser un logiciel standard de Programmation Linéaire en Nombres Entiers dans un cas et un logiciel de Programmation Quadratique Convexe en Nombres Entiers dans l'autre.

Par ailleurs, on sait qu'un des paramètres essentiels de l'efficacité d'une méthode arborescente est la qualité de la borne inférieure obtenue à la racine de l'arborescence de recherche. Cette borne est égale à la valeur optimale de la relaxation continue du problème linéarisé ou convexifié. Nous dirons qu'une reformulation est bonne si sa relaxation continue donne une borne de bonne qualité. Nous dirons aussi que, à l'intérieur d'un même schéma de reformulation, une reformulation est meilleure qu'une autre si sa relaxation continue donne une meilleure borne.

Définition 1 *Soit* $(\Pi 01)$ *et* $(\Pi' 01)$ *deux reformulations de* $(Q01)$ *(toutes deux linéaires ou quadratiques).* $(\Pi 01)$ *est une meilleure reformulation que* $(\Pi' 01)$ *si et seulement si* $v\left(\overline{\Pi 01}\right) \geq v\left(\overline{\Pi' 01}\right)$

Notre contribution a consisté à développer des méthodes de résolution exactes basées sur des reformulations du problème $(Q01)$. Nous proposons des méthodes de pré-traitement, soit par la programmation semidéfinie soit par la programmation linéaire. Le chapitre suivant présente une de ces approches de pré-traitement.

Chapitre 3

QCR : Une nouvelle Reformulation Quadratique Convexe

En section 3.1, nous allons mettre en évidence les motivations qui nous ont amenés à développer notre méthode de résolution exacte, pour ensuite la présenter en section 3.2. L'algorithme général de résolution ainsi élaboré est donné dans cette même section ainsi qu'un parallèle avec la dualité lagrangienne. On définit également une condition nécessaire de convexification de (Q01) lorsque l'on utilise la reformulation QCR. Enfin, la section 3.3 montre que la méthode QCR est la meilleure reformulation quadratique convexe pour les problèmes d'optimisation quadratique avec une contrainte de cardinalité.

3.1 Motivations, une méthode en deux phases

Nous allons supposer, dans ce chapitre, que la matrice Q n'est pas semidéfinie positive ; c'est le cas, par exemple, si tous ses termes diagonaux sont nuls (voir Annexe 1). Dans ce cas, la fonction objectif de (Q01) n'est pas convexe et la résolution de sa relaxation continue est un problème NP-difficile.

L'utilisation des logiciels spécifiques à la programmation quadratique convexe, tels que la version 9 de CPLEX [41] a été la *première motivation* de notre travail. Ces solveurs, de plus en plus efficaces, requièrent que le hessien

du programme à résoudre soit semidéfini positif. Puisqu'ils utilisent généralement un algorithme de branch-and-bound basé sur la relaxation continue en chaque nœud de l'arborescence de recherche, *une seconde motivation*, toute aussi importante, a été de maximiser la borne inférieure obtenue par relaxation continue, à la racine de l'arborescence.

La méthode que nous avons développée est une méthode en deux phases : la *Phase I* consiste à transformer $(Q01)$ en un nouveau problème quadratique $(Q01')$, équivalent et convexe. En d'autres termes, la fonction objectif $q(x)$ est remplacée par une nouvelle fonction quadratique $q'(x)$ convexe et telle que, pour tout x solution réalisable de $(Q01)$, $q'(x) = q(x)$. La *Phase II* consiste, quant à elle, à soumettre tout simplement le problème modifié à un solveur quadratique convexe.

La *Phase I*, qui est au cœur de notre travail, peut être vue comme une phase de reformulation ou encore comme une phase de pré-traitement. Elle s'appuie sur une relaxation semidéfinie. La réécriture ainsi obtenue tire profit de la qualité de la relaxation sur laquelle elle se base. Le schéma proposé contourne donc, d'une certaine façon, la difficulté d'intégrer les relaxations, coûteuses en temps de calcul, dans un algorithme de branch-and-bound. En effet, il est important de noter qu'une fois la réécriture de l'objectif établie (*Phase I*), on résout par un branch-and-bound, sans nouvelle transformation du problème.

Nous avons appelé cette nouvelle approche **QCR** pour **Quadratic Convex Reformulation** [19].

Nous avons supposé que la fonction objectif n'était pas convexe. Cependant, nous verrons par la suite (chapitre 7) que la méthode QCR peut aussi être utilisée sur des fonctions déjà convexes. Dans ce cas, l'intérêt d'appliquer notre méthode de reformulation réside dans une possible amélioration de la borne inférieure obtenue par relaxation continue du problème transformé.

3.2 Présentation de QCR

Notre méthode de reformulation travaille sur la fonction objectif en utilisant à la fois les contraintes d'intégrité et les contraintes d'égalité de $(Q01)$:

$$
\begin{aligned}
(Q01) \ : \quad \min \quad & q(x) = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} x_i x_j + \sum_{i=1}^{n} c_i x_i \\
\text{s.c.} \quad & \sum_{i=1}^{n} a_{ki} x_i = b_k \quad k = 1, \ldots, m \\
& \sum_{i=1}^{n} a'_{li} x_i \leq b'_l \quad l = 1, \ldots, p \\
& x \in \{0,1\}^n
\end{aligned}
$$

On rappelle (voir Notations) que :
$X = \left\{ x \in \{0,1\}^n \ : \sum_{i=1}^{n} a_{ki} x_i = b_k, \quad k = 1, \ldots, m; \sum_{i=1}^{n} a'_{li} x_i \leq b'_l, \quad l = 1, \ldots, p \right\}$
et
$\overline{X} = \left\{ x \in [0,1]^n \ : \sum_{i=1}^{n} a_{ki} x_i = b_k, \quad k = 1, \ldots, m; \sum_{i=1}^{n} a'_{li} x_i \leq b'_l, \quad l = 1, \ldots, p \right\}$
Sans perte de généralité, on suppose que $Ax = b$ possède au moins une solution.

L'approche consiste à ajouter à la fonction objectif $q(x)$ deux fonctions quadratiques nulles sur l'ensemble admissible et qui dépendent de deux paramètres $\alpha \in \mathbb{R}^{m \times n}$ et $u \in \mathbb{R}^n$:

$$
\begin{aligned}
q_\alpha(x) &= \sum_{k=1}^{m} \left(\sum_{i=1}^{n} \alpha_{ki} x_i \right) \left(\sum_{j=1}^{n} a_{kj} x_j - b_k \right) \\
q_u(x) &= \sum_{i=1}^{n} u_i \left(x_i^2 - x_i \right)
\end{aligned}
$$

La première fonction $q_\alpha(x)$ utilise les contraintes d'égalité et est égale à 0 pour tout x solution admissible du problème. Ici, chaque *kième* contrainte d'égalité est multipliée par chaque variable x_i et chaque composante (k, i) de α.

La seconde fonction $q_u(x)$ utilise, quant à elle, les contraintes d'intégrité

$x_i^2 = x_i$ et est égale à 0 si x est un vecteur de $\{0,1\}^n$.

Si l'on ajoute ces deux fonctions paramétrées à la fonction initiale $q(x)$, nous obtenons une nouvelle fonction objectif $q_{\alpha,u}(x)$:

$$\begin{aligned} q_{\alpha,u}(x) &= q(x) + \sum_{k=1}^{m} \left(\sum_{i=1}^{n} \alpha_{ki} x_i \right) \left(\sum_{j=1}^{n} a_{kj} x_j - b_k \right) + \sum_{i=1}^{n} u_i \left(x_i^2 - x_i \right) \\ &= q(x) + q_\alpha(x) + q_u(x) \end{aligned}$$

et donc un nouveau problème quadratique en 0-1 sous contraintes linéaires que nous notons $(Q01_{\alpha,u})$:

$$(Q01_{\alpha,u}) \quad : \quad \min \quad \left\{ q_{\alpha,u}(x) \ : Ax = b,\ A'x \leq b',\ x \in \{0,1\}^n \right\}$$

Ce nouveau problème n'a pas de contraintes ni de variables supplémentaires : seul l'objectif est modifié.

Notons Q_α (resp. c_α) la matrice Q (resp. le vecteur c) perturbée par le paramètre $\alpha \in \mathbb{R}^{m \times n}$ et $Q_{\alpha,u}$ (resp. $c_{\alpha,u}$) la matrice Q (resp. le vecteur c) perturbée à la fois par le paramètre $\alpha \in \mathbb{R}^{m \times n}$ et par le paramètre $u \in \mathbb{R}^n$. On définit ainsi deux nouvelles matrices et deux nouveaux vecteurs :

$$\begin{gathered} Q_\alpha = Q + \frac{1}{2} \left(\alpha^t A + A^t \alpha \right), \quad Q_{\alpha,u} = Q_\alpha + Diag(u), \\ c_\alpha = c - \alpha^t b, \quad c_{\alpha,u} = c_\alpha - u. \end{gathered}$$

plus précisément :

$$Q_{\alpha,u} = \begin{pmatrix} q_{11} + \sum_{\mathbf{k=1}}^{\mathbf{m}} \alpha_{\mathbf{k1}} \mathbf{a_{k1}} + \mathbf{u_1} & \cdots & \cdots & q_{1j} + \sum_{\mathbf{k=1}}^{\mathbf{m}} \alpha_{\mathbf{k1}} \mathbf{a_{kj}} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & q_{ii} + \sum_{\mathbf{k=1}}^{\mathbf{m}} \alpha_{\mathbf{ki}} \mathbf{a_{ki}} + \mathbf{u_i} & \cdots & q_{ij} + \sum_{\mathbf{k=1}}^{\mathbf{m}} \alpha_{\mathbf{ki}} \mathbf{a_{kj}} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & q_{nn} + \sum_{\mathbf{k=1}}^{\mathbf{m}} \alpha_{\mathbf{kn}} \mathbf{a_{kn}} + \mathbf{u_n} \end{pmatrix}$$

et

$$c_{\alpha,u} = \begin{pmatrix} c_1 - \mathbf{u_1} - \sum_{\mathbf{k=1}}^{\mathbf{m}} \alpha_{\mathbf{k1}} \mathbf{b_k} \\ \cdots \\ \cdots \\ c_i - \mathbf{u_i} - \sum_{\mathbf{k=1}}^{\mathbf{m}} \alpha_{\mathbf{ki}} \mathbf{b_k} \\ \cdots \\ \cdots \\ c_1 - \mathbf{u_n} - \sum_{\mathbf{k=1}}^{\mathbf{m}} \alpha_{\mathbf{kn}} \mathbf{b_k} \end{pmatrix}$$

La matrice $Q_{\alpha,u}$ est le nouveau hessien du programme $(Q01_{\alpha,u})$.

Il est facile de vérifier que pour tout $x \in \{0,1\}^n$ tel que $Ax = b$, la fonction $q_{\alpha,u}(x)$ est égale à $q(x)$ et donc, $(Q01_{\alpha,u})$ est équivalent à $(Q01)$

Nous nous intéressons donc aux différentes reformulations de $q(x)$ en la fonction paramétrée $q_{\alpha,u}(x)$. Ces différentes reformulations n'ont de sens, pour notre problématique, que si la nouvelle fonction est convexe : il faut donc pouvoir déterminer les paramètres α et u qui convexifient la fonction transformée : c'est toujours possible. En effet, si l'on pose $\alpha = O$ et $u = -\lambda_{min}(Q)e$, $q_{\alpha,u}(x)$ est convexe (voir la section 4.1). D'après les objectifs définis en section 3.1, nous allons déterminer $\alpha \in \mathbb{R}^{m \times n}$ et $u \in \mathbb{R}^n$ tels que :

1. $q_{\alpha,u}(x)$ **soit convexe et**

2. la valeur optimale de la relaxation continue de $(Q01_{\alpha,u})$ **soit maximale.**

Plus précisément, nous allons chercher à résoudre le problème suivant :

$$(C(Q01)) \ : \max_{\substack{\alpha \in \mathbb{R}^{m \times n}, u \in \mathbb{R}^n \\ Q_{\alpha,u} \succeq 0}} \left\{ \min_{x \in \overline{X}} \ q_{\alpha,u}(x) \right\}$$

Dans le théorème 1 suivant, nous montrons que le problème $(C(Q01))$ est équivalent au dual semidéfini d'une relaxation semidéfinie de $(Q01)$, notée $(SDQ01)$. Par conséquent, une solution optimale (α^*, u^*) de $(C(Q01))$ peut être obtenue en résolvant $(SDQ01)$, qui est un problème polynomial [78].

Theorème 1 La valeur optimale de $(C(Q01))$ est égale à la valeur optimale du programme semidéfini suivant :

$$
\begin{aligned}
(SDQ01): \quad \min \quad & \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} X_{ij} \\
\text{s.c.} \quad & \\
& X_{ii} = x_i \qquad\qquad i = 1, .., n \qquad (3.1) \\
& -b_k x_i + \sum_{j=1}^{n} a_{kj} X_{ij} = 0 \quad k = 1, .., m; i = 1, .., n \quad (3.2) \\
& Ax = b \\
& A'x \leq b' \\
& \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\
& x \in \mathbb{R}^n, \ X \in S_n
\end{aligned}
$$

Les valeurs optimales u_i^* de u_i $(i = 1, \ldots, n)$ sont données par les valeurs optimales des variables duales associées aux contraintes (3.1) et les valeurs optimales α_{ki}^* de α_{ki} $(i = 1, \ldots, n; k = 1, \ldots, m)$ sont données par les valeurs optimales des variables duales associées aux contraintes (3.2).

Preuve 1 *Supposons que $q_{\alpha,u}(x)$ est une fonction convexe.*
Puisque $\{x : Ax = b, A'x \leq b'\}$ est un ensemble convexe, par dualité lagrangienne, $(C(Q01))$ est équivalent à :

$$
(LD): \max_{\substack{\alpha \in \mathbb{R}^{m\times n}, u \in \mathbb{R}^n \\ Q_{\alpha,u} \succeq 0}} \left\{ \max_{\substack{\beta^t \in \mathbb{R}^m \\ \beta'^t \in \mathbb{R}^p_+}} \left\{ \min_{x \in [0,1]^n} \left\{ c_{\alpha,u}^t x + x^t Q_{\alpha,u} x + \beta^t (Ax - b) + \beta'^t (A'x - b') \right\} \right\} \right\}
$$

en dualisant les contraintes d'égalité et d'inégalité de $\overline{X}$.
Pour des valeurs données des paramètres $\alpha \in \mathbb{R}^{n\times m}, \beta \in \mathbb{R}^m$ et $\beta' \in \mathbb{R}^p_+$, notées respectivement $\tilde{\alpha}, \tilde{\beta}$ et $\tilde{\beta}'$, on considère le problème suivant :

$$
\left(L_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right): \max_{\substack{u \in \mathbb{R}^n \\ Q_{\tilde{\alpha},u} \succeq 0}} \left\{ \min_{x \in [0,1]^n} \left\{ c_{\tilde{\alpha},u}^t x + x^t Q_{\tilde{\alpha},u} x + \tilde{\beta}^t (Ax - b) + \tilde{\beta}'^t (A'x - b') \right\} \right\}
$$

Définissons maintenant la fonction suivante,

$$
\begin{aligned}
f_u(x) &= c_{\tilde{\alpha},u}^t x + x^t Q_{\tilde{\alpha},u} x + \tilde{\beta}^t (Ax - b) + \tilde{\beta}'^t (A'x - b') \\
&= c_{\tilde{\alpha}}^t x + x^t Q_{\tilde{\alpha}} x + \tilde{\beta}^t (Ax - b) + \tilde{\beta}'^t (A'x - b') + \sum_{i=1}^{n} u_i (x_i^2 - x_i)
\end{aligned}
$$

le problème suivant,

$$\left(L'_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right) : \max_{\substack{u\in\mathbb{R}^n \\ Q_{\tilde{\alpha},u}\succeq 0}} \left\{ \min_{x\in\mathbb{R}^n} f_u(x) \right\}$$

et montrons tout d'abord que :

$$v\left(L_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right) = v\left(L'_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right)$$

Si l'on observe que l'ensemble convexe $[0,1]^n$ *peut aussi s'écrire :*

$$\left\{x \in \mathbb{R}^n : x_i^2 \leq x_i,\ i = 1,\ldots,n\right\}$$

alors, $\left(L_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right)$ *devient :*

$$\left(L_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right) : \max_{\substack{u\in\mathbb{R}^n \\ Q_{\tilde{\alpha},u}\succeq 0}} \left\{ \min_{\substack{x_i^2\leq x_i \\ x\in\mathbb{R}^n}} f_u(x) \right\}$$

Par hypothèse, $f_u(x)$ *est une fonction convexe et* $\left\{x \in \mathbb{R}^n : x_i^2 \leq x_i, i = 1,\ldots,n\right\}$ *définit un ensemble convexe. Donc, par dualité lagrangienne (plus précisément en dualisant les* n *contraintes* $x_i^2 \leq x_i$*), on obtient :*

$$\begin{aligned} v\left(L_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right) &= \max_{\substack{u\in\mathbb{R}^n \\ Q_{\tilde{\alpha},u}\succeq 0}} \left\{ \max_{\eta\in\mathbb{R}^n_+} \left\{ \min_{x\in\mathbb{R}^n} \left\{ f_u(x) + \sum_{i=1}^n \eta_i(x_i^2 - x_i) \right\} \right\} \right\} \\ &= \max_{\substack{u\in\mathbb{R}^n \\ Q_{\tilde{\alpha},u}\succeq 0}} \left\{ \min_{x\in\mathbb{R}^n} \{f_u(x)\} \right\} = v\left(L'_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right) \end{aligned}$$

en posant $u = u + \eta$*.*

Lemaréchal et Oustry [99] ont prouvé qu'une condition nécessaire pour que $f_u(x)$ *ait un minimum fini est que* $Q_{\tilde{\alpha},u} \succeq 0$*.*

Il a été montré que le problème $\left(L'_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right)$ *(et donc* $\left(L_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right)$*) a la même valeur optimale que le programme semidéfini* $(P_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'})$ *suivant (voir par exemple [95], [100], [119], [128]) et que les valeurs optimales* u_i^* *de* u_i *pour le problème* $\left(L_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}\right)$ *sont données par les valeurs optimales des variables duales*

associées aux contraintes (3.3) suivantes :

$$(P_{\tilde{\alpha},\tilde{\beta},\tilde{\beta}'}): \quad \begin{array}{ll} min & c^t_{\tilde{\alpha}}x + \sum_{i=1}^{n}\sum_{j=1}^{n} q_{\tilde{\alpha}_{ij}} X_{ij} + \tilde{\beta}^t(Ax-b) + \tilde{\beta}'^t(A'x-b') \\ s.c. & \end{array}$$

$$\begin{array}{ll} X_{ii} = x_i & i = 1,\ldots,n \\ \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 & \\ x \in \mathbb{R}^n,\ X \in S_n & \end{array} \tag{3.3}$$

où $q_{\tilde{\alpha}_{ij}}$ représente le coefficient (i,j) de la matrice $Q_{\tilde{\alpha}}$.

Ainsi, le problème (LD) consiste à résoudre $\max_{\alpha\in\mathbb{R}^{n\times m},\beta\in\mathbb{R}^m,\beta'\in\mathbb{R}^p_+} v(P_{\alpha,\beta,\beta'})$.

La fonction objectif de $(P_{\alpha,\beta,\beta'})$ est linéaire en x et X et son ensemble de solutions réalisables est convexe. Ainsi, encore par dualité lagrangienne, nous pouvons conclure que (LD) et donc $(C(Q01))$ a la même valeur optimale que $(SDQ01)$. □

Corollaire 1 $v(SDQ01) = v(\overline{Q01_{\alpha^*,u^*}})$

Schématiquement, la relaxation semidéfinie $(SDQ01)$ de $(Q01)$, définie dans le théorème 1, est obtenue comme suit : si l'on multiplie chaque contrainte d'égalité $\sum_{i=1}^{n} a_{ki}x_i = b_k$ par x_i $(i = 1,\ldots,n)$ puis si l'on remplace les produits x_ix_j par les variables X_{ij}, on obtient le problème équivalent :

$$(Q01_m): \quad \begin{array}{ll} \min & q(x) = \sum_{i=1}^{n} q_i x_i + \sum_{i=1}^{n}\sum_{j=i,j\neq i}^{n} q_{ij}X_{ij} \\ \text{s.c.} & \end{array}$$

$$\begin{array}{ll} \sum_{i=1}^{n} a_{ki}x_i = b_k & k = 1,\ldots,m \\ \sum_{j=1}^{n} a_{kj}X_{ij} = b_k x_i & k = 1,\ldots,m; i = 1,\ldots,n \\ \sum_{i=1}^{n} a'_{\ell i}x_i \leq b'_\ell & \ell = 1,\ldots,p \\ X_{ij} = x_i x_j & i = i,\ldots,n; j = 1,\ldots,n \\ x \in \mathbb{R}^n,\ X \in S_n & \end{array}$$

La relaxation semidéfinie $(SDQ01)$ que nous considérons consiste à relâcher l'ensemble des contraintes $X_{ij} = x_ix_j$ en imposant à la matrice $X - xx^t$

d'être semidéfinie positive. Il est connu que $X - xx^t \succeq 0$ est équivalent à $\begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0$ (lemme de Schur [78]).

Un dual lagrangien

La reformulation QCR peut être vue comme un dual lagrangien du programme $(Q01_p)$ suivant, obtenu à partir de $(Q01)$ en ajoutant, au préalable, des contraintes quadratiques produit redondantes. Ces contraintes sont obtenues en multipliant chaque contrainte d'égalité par chaque variable x_i ; on réécrit également les contraintes d'intégrité :

$$
\begin{aligned}
(Q01_p): \quad \min \quad & q(x) = \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} x_i x_j \\
\text{s.c.} \quad & \\
& \sum_{i=1}^{n} a_{ki} x_i = b_k && k = 1, \ldots, m \\
& \sum_{j=1}^{n} a_{kj} x_i x_j = b_k x_i && k = 1, \ldots, m, i = 1, \ldots, n \\
& \sum_{i=1}^{n} a'_{\ell i} x_i \leq b'_\ell && \ell = 1, \ldots, p \\
& x_i^2 - x_i = 0 && i = 1, \ldots, n
\end{aligned}
$$

On ajoute donc $\mathcal{O}(nm)$ contraintes d'égalité. Clairement, ce nouveau problème est équivalent à $(Q01)$. Nous allons montrer que le dual lagrangien de $(Q01_p)$, obtenu en dualisant toutes ses contraintes, n'est autre que $(C(Q01))$ (nous utiliserons le même schéma de preuve que celui du théorème 1).

Soit $(C(Q01))$, déjà défini précédemment :

$$
\max_{\substack{\alpha \in \mathbb{R}^{m\times n}, u \in \mathbb{R}^n \\ Q_{\alpha,u} \succeq 0}} \left\{ \min_{x \in [0,1]^n} \left\{ \; q_{\alpha,u}(x) : Ax = b, A'x \leq b' \right\} \right\}
$$

En notant que $0 \leq x_i \leq 1$ $(i = 1, \ldots, n)$ est équivalent à $x_i^2 \leq x_i$ $(i = 1, \ldots, n)$, notre problème de convexification peut être réécrit :

$$
(D1) \; : \max_{\substack{\alpha \in \mathbb{R}^{m\times n}, u \in \mathbb{R}^n \\ Q_{\alpha,u} \succeq 0}} \left\{ \min_{x \in \mathbb{R}^n} \; \left\{ q_{\alpha,u}(x) : Ax = b, A'x \leq b', x_i^2 \leq x_i (i = 1, \ldots, n) \right\} \right\}
$$

Or, $q_{\alpha,u}(x)$ est une fonction convexe et les contraintes $Ax = b$, $A'x \leq b$ et $x_i^2 \leq x_i$ définissent un ensemble convexe. Ainsi, encore par dualité lagrangienne, $(D1)$ est équivalent à $(D2)$:

$$(D2) \; : \max_{\substack{\alpha \in \mathbb{R}^{m \times n}, u \in \mathbb{R}^n \\ \beta \in \mathbb{R}^m, \beta' \in \mathbb{R}^p_+, \lambda \in \mathbb{R}^n_+ \\ Q_{\alpha,u} \succeq 0}} \left\{ \min_{x \in \mathbb{R}^n} \left\{ q_{\alpha,u}(x) + \sum_{i=1}^{n} \lambda_i (x_i^2 - x_i) + \beta^t (Ax - b) + \beta'^t (A'x - b) \right\} \right\}$$

Enfin, $(D2)$ est équivalent au problème $(D3)$ suivant :

$$(D3) \; : \max_{\substack{\alpha \in \mathbb{R}^{m \times n}, u \in \mathbb{R}^n \\ \beta \in \mathbb{R}^m, \beta' \in \mathbb{R}^p_+ \\ Q_{\alpha,u} \succeq 0}} \left\{ \min_{x \in \mathbb{R}^n} \left\{ z_{\alpha,u,\beta,\beta'}(x) = q_{\alpha,u}(x) + \sum_{i=1}^{n} \beta^t (Ax - b) + \beta'^t (A'x - b) \right\} \right\}$$

en posant $u = u + \lambda$. En effet, si $(\alpha^*, u^*, \beta^*, \beta'^*, \lambda^*)$ est une solution optimale de $(D2)$ alors $(\alpha^*, u^* = u^* + \lambda^*, \beta^*, \beta'^*)$ est une solution réalisable de $(D3)$, de même valeur. Il est connu qu'une condition nécessaire pour que la fonction quadratique $z_{\alpha,u,\beta,\beta'}(x)$ ait un minimum fini est que $Q_{\alpha,u}$ soit semidéfinie positive. Ainsi $(D3)$ est équivalent à $(D4)$:

$$(D4) \; : \max_{\substack{\alpha \in \mathbb{R}^{m \times n}, u \in \mathbb{R}^n \\ \beta \in \mathbb{R}^m, \beta' \in \mathbb{R}^p_+}} \left\{ \min_{x \in \mathbb{R}^n} \; z_{\alpha,u,\beta,\beta'}(x) \right\}$$

qui représente le dual lagrangien associé à $(Q01_p)$ et obtenu en relâchant toutes les contraintes.

Algorithme QCR

D'après le théorème 1, nous pouvons construire un algorithme de résolution exacte de $(Q01)$ basé sur la reformulation QCR :

Algorithme 1 Résolution exacte d'un programme quadratique en 0-1 sous contraintes linéaires basée sur la reformulation QCR

1. Résoudre le programme semidéfini $(SDQ01)$.
Soient α^* et u^* les valeurs optimales des variables duales associées aux contraintes (3.1) et (3.2) respectivement de $(SDQ01)$.
2. Résoudre le programme quadratique $(Q01_{\alpha^*,u^*})$ dont la relaxation continue $(\overline{Q01_{\alpha^*,u^*}})$ est un problème convexe et dont la valeur optimale est égale à la valeur optimale de $(SDQ01)$.

La méthode QCR est une méthode très générale de résolution pour la programmation quadratique en variables 0-1 sous contraintes linéaires. Nous verrons, dans les chapitres 5, 6 et 7, diverses applications qui démontrent l'efficacité de la méthode.

Une condition nécessaire de convexité

Est-il toujours possible de convexifier l'objectif en lui ajoutant les fonctions paramétrées $q_u(x)$ et $q_\alpha(x)$? Pourrait-on convexifier l'objectif en ajoutant seulement la fonction $q_\alpha(x)$? On sait déjà que l'ajout seul de la fonction $q_u(x)$ permet de convexifier l'objectif (voir [18] ou encore la section 4.1).

Pour répondre à la question suivante : *"existe-t-il toujours un paramètre* $\alpha \in \mathbb{R}^{m\times n}$ *permettant de convexifier l'objectif* $q(x) + q_\alpha(x)$ *?"*, nous allons nous intéresser à un problème particulier d'optimisation combinatoire : le problème du k-cluster, étudié en détail dans le chapitre 6 consacré aux expérimentations. Nous allons montrer que, pour ce problème, il est impossible de trouver des valeurs de α telles que $q(x) + q_\alpha(x)$ soit convexe.

Le problème du k-cluster consiste à sélectionner un sous-ensemble S de k sommets dans un graphe $G = (V, U)$ ($V = \{v_1, \ldots, v_n\}$ est l'ensemble des sommets de G) tel que $S \subseteq V$ et tel que le sous-graphe induit par S maximise le nombre d'arêtes. Ce problème se modélise comme un problème quadratique en variables 0-1 sous une contrainte de cardinalité :

$$(KC') \quad : \quad \max \left\{ f'(x) = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \frac{1}{2} \delta_{ij} x_i x_j : \sum_{i=1}^{n} x_i = k,\ x \in \{0,1\}^n \right\}$$

où les coefficients binaires $\delta_{ij} = 1$ si et seulement si $[v_i, v_j]$ est une arête de G. La variable binaire x_i est égale à 1 si et seulement si le sommet v_i appartient au sous-ensemble S. (KC') peut se réécrire comme un problème de minimisation (KC) dont la valeur optimale est l'opposée de celle de (KC') :

$$(KC) \quad : \quad \min \left\{ f(x) = -\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \frac{1}{2} \delta_{ij} x_i x_j : \sum_{i=1}^{n} x_i = k,\ x \in \{0,1\}^n \right\}$$

Nous allons montrer que, $\forall \alpha$, l'ajout seul de la fonction $f_\alpha(x) = \sum_{i=1}^{n} \alpha_i x_i \left(\sum_{j=1}^{n} x_j - k \right)$ à $f(x)$ ne permet pas de rendre la nouvelle fonction $f(x) + f_\alpha(x)$ convexe.

On remarque ici que α est un vecteur à n composantes car (KC) ne contient qu'une seule contrainte d'égalité.

Proposition 1 Pour le problème (KC), $\forall \alpha \in \mathbb{R}^{m \times n}$, la fonction $f(x) + f_\alpha(x)$ n'est pas convexe.

Preuve 2 *Considérons donc le problème du k-cluster. D'après sa modélisation, on a :*
$f(x) + f_\alpha(x) = x^t Q_\alpha x + c_\alpha^t x$ *avec*

$\alpha \in \mathbb{R}^n$ *et*

$$Q_{\alpha_{ij}} = \begin{cases} \alpha_i & si \quad i = j \\ -\frac{1}{2}\delta_{ij} + \frac{1}{2}(\alpha_i + \alpha_j) & si \quad i \neq j \end{cases}$$

Montrons que $\forall \alpha \in \mathbb{R}^n$, Q_α *n'est pas une matrice semidéfinie positive. Autrement dit, montrons qu'il existe un vecteur* $\tilde{x} \in \mathbb{R}^n$ *tel que* $\tilde{x}^t Q_\alpha \tilde{x} < 0$ *(voir la Propriété 3 de l'Annexe 1).*

$$\tilde{x}^t Q_\alpha \tilde{x} = \sum_{i=1}^{n} \alpha_i \tilde{x}_i^2 + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (\alpha_i + \alpha_j - \delta_{ij})\, \tilde{x}_i \tilde{x}_j.$$

Soient 4 indices k, l, m, p *tels que* $k < l < m < p$ *et tels que :*

- $\delta_{kl} = \delta_{mp} = 1$ *et*
- $\delta_{km} = \delta_{kp} = \delta_{lm} = \delta_{lp} = 0$

On considère donc deux arêtes (v_k, v_l) *et* (v_m, v_p) *disjointes, ce qui n'est pas restrictif pour des graphes de grande taille.*
Posons maintenant $\tilde{x_k} = \tilde{x_l} = 1$ *;* $\tilde{x_m} = \tilde{x_p} = -1$ *et* $\tilde{x_i} = 0$ $\forall i \neq (k, l, m, p)$.
Ainsi,

$$\begin{aligned} \tilde{x}^t Q_\alpha \tilde{x} &= \alpha_k + \alpha_l + \alpha_m + \alpha_p + (\alpha_k + \alpha_l - 1 - \alpha_k - \alpha_m - \alpha_k - \alpha_p - \alpha_l - \alpha_m \\ &\quad -\alpha_l - \alpha_p + \alpha_m + \alpha_p - 1) \\ &= -2 < 0 \end{aligned}$$

$\forall \alpha \in \mathbb{R}^n$, *il existe donc un vecteur* $x \in \mathbb{R}^n$, *non nul, tel que* $x^t Q_\alpha x < 0$. Q_α *n'est donc pas semidéfinie positive.* □

Ainsi, convexifier l'objectif en ajoutant seulement la fonction $q_\alpha(x)$ n'est pas toujours possible.

Corollaire 2 On ne peut pas toujours trouver de paramètre $\alpha \in \mathbb{R}^{m \times n}$ tel que $q(x) + q_\alpha(x)$ soit convexe pour le problème général $(Q01)$.

3.3 Optimalité de QCR pour l'optimisation d'une fonction quadratique sous une contrainte de cardinalité

QCR se base sur un schéma de réécriture bien particulier : ajouter des fonctions nulles à l'objectif initial. Nous pouvons nous interroger sur cet ajout de fonctions nulles : est-ce que, pour un problème d'optimisation combinatoire donné, la méthode QCR permet d'ajouter *toutes* les fonctions nulles ? Autrement dit, est-ce que QCR peut être une reformulation optimale pour des problèmes d'optimisation particuliers ? Bien sûr, cette question n'a de sens que si l'on ne considère que notre schéma de réécriture. Cette section va répondre à cette question et mettre en évidence l'optimalité de QCR pour une famille de problèmes combinatoires.

Définition 2 *Soient $\tilde{c}$ un vecteur réel à $(n+1)$ coefficients et $\tilde{Q}$ une matrice réelle de dimension $n \times n$. $min\left\{\tilde{c_0} + \tilde{c}^t x + x^t \tilde{Q} x : x \in X\right\}$ est une reformulation quadratique convexe de $(Q01)$ si*

1. *$\tilde{c_0} + \tilde{c}^t x + x^t \tilde{Q} x = q(x)$ pour tout $x \in X$ et,*
2. *$\tilde{c_0} + \tilde{c}^t x + x^t \tilde{Q} x$ est une fonction convexe, c'est-à-dire si $\tilde{Q}$ est une matrice semidéfinie positive.*

Soit $Conv(Q01)$ l'ensemble de toutes les reformulations convexes de $(Q01)$.

Définition 3 *Un programme quadratique en variables bivalentes (P) est la* **meilleure reformulation quadratique** *de $(Q01)$ si*

1. *$(P) \in Conv(Q01)$ et*
2. *$v(\overline{P}) \geq v(\overline{P'})$ pour tout problème (P') de $Conv(Q01)$.*

La meilleure reformulation quadratique convexe pour les programmes quadratiques en variables 0-1

Soit G_0 l'ensemble des fonctions quadratiques nulles dans X.

Theorème 2 La meilleure reformulation convexe de $(Q01)$ est

$$(Q01_{g^*}) \ : \min \ \left\{c^t x + x^t Q x + g^*(x) \ : \ x \in X\right\}$$

où g^* est une fonction quadratique convexe de G_0 telle que :

$$v(\overline{Q01_{g^*}}) = \max_{\substack{g(x)\in G_0 \\ x^tQx+g(x)\ convexe}} \left\{\min\ \left\{c^tx + x^tQx + g(x)\ :\ x \in \overline{X}\right\}\right\}$$

Preuve 3 *Raisonnons par l'absurde et supposons que* $(Q01_{g^*})$ *n'est pas la meilleure reformulation quadratique convexe de* $(Q01)$*. Autrement dit, il existe un vecteur* $\tilde{c} \in \mathbb{R}^{n+1}$ *et une matrice* $\tilde{Q} \in \mathbb{R}^{n\times n}$ *tels que le programme* $\min\left\{\tilde{c_0} + \tilde{c}^tx + x^t\tilde{Q}x : x \in X\right\}$ *est une reformulation quadratique convexe de* $(Q01)$ *dont la valeur optimale de la relaxation continue est meilleure que* $v(\overline{Q01_{g^*}})$*.*
Soit $h(x) = \tilde{c_0} + \tilde{c}^tx + x^t\tilde{Q}x - q(x)$*.*
Clairement, $h(x) = 0\ \forall x \in X$*.*
Par hypothèse, on a :

$$\min\ \left\{q(x) + h(x)\ :\ x \in \overline{X}\right\} > \min\ \left\{q(x) + g^*(x)\ :\ x \in \overline{X}\right\}$$

et donc en particulier :

$$\min\ \left\{q(x) + h(x)\ :\ x \in \overline{X}\right\} > v\left(\overline{Q01^*_g}\right)$$

Cependant, cette dernière inégalité contredit la définition de $(Q01_{g^*})$ *puisque la fonction* $h(x)$ *satisfait les contraintes de* $v(\overline{Q01_{g^*}})$*. En effet,* $h(x)$ *est une fonction de* G_0 *et* $q(x) + h(x)$ *est convexe.* □

La meilleure reformulation quadratique convexe de $(Q01)$ pour les problèmes avec une contrainte de cardinalité

On considère les problèmes $(Q01)$ ne possédant qu'une seule contrainte de cardinalité : le vecteur x à n composantes satisfait la contrainte $\sum_{i=1}^{n} x_i = k$. Autrement dit, x doit avoir exactement k coefficients non nuls. Le problème associé s'écrit donc ainsi :

$$(QCC)\quad \min\ \left\{q(x) = c_0 + \sum_{i=1}^{n} c_ix_i + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} q_{ij}x_ix_j\ :\ x \in X_{n,k}\right\}$$

en posant $X_{n,k} = \left\{x : \sum_{i=1}^{n} x_i = k, x \in \{0,1\}^n\right\}$

Par exemple, les problèmes du k-cluster et de la bipartition de graphes peuvent facilement se modéliser comme la minimisation d'une fonction quadratique en 0-1 sous une contrainte de cardinalité. Ces deux problèmes sont étudiés en détail dans les sections 6.2 et 6.3.

Le lemme suivant rappelle une caractérisation des fonctions quadratiques nulles définies sur $X_{n,k}$ [22].

Lemme 1 Pour tout $2 \leq k \leq n-2$, la fonction quadratique

$$c(x) \;=\; r_0 + \sum_{i=1}^{n} r_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} s_{ij} x_i x_j$$

est égale à 0 sur $X_{n,k}$ si et seulement si $\exists (a_0, a_1, \ldots, a_n) \in \mathbb{R}^{n+1}$ tels que

$$c(x) = \left(a_0 + \sum_{i=1}^{n} a_i x_i \right) \left(\sum_{j=1}^{n} x_j - k \right) + \sum_{i=1}^{n} a_i \left(x_i^2 - x_i \right)$$

On peut facilement déduire du Lemme 1 une caractérisation de toutes les fonctions quadratiques nulles sur $X_{n,k}$.

Corollaire 3 Pour tout $2 \leq k \leq n-2$, la fonction quadratique

$$c(x) \;=\; r_0 + r^t x + x^t S x,$$

où $S \in \mathbb{R}^{n \times n}$ et $r \in \mathbb{R}^{n+1}$, est égale à 0 sur $X_{n,k}$ si et seulement si $\exists \alpha \in \mathbb{R}^{n+1}$ et $u \in \mathbb{R}^n$ tels que

$$c(x) \;=\; \left(\alpha_0 + \sum_{i=1}^{n} \alpha_i x_i \right) \left(\sum_{j=1}^{n} x_j - k \right) \;+\; \sum_{i=1}^{n} u_i \left(x_i^2 - x_i \right)$$

Preuve 4 *La condition suffisante est évidente par la définition de $X_{n,k}$ puisque $X_{n,k}$ représente l'ensemble des variables x telles que $\sum_{i=1}^{n} x_i \;=\; k$ et $x_i \;=\; x_i^2 \;\forall i = 1, \ldots, n$.*
Montrons maintenant l'implication suivante :

$$c(x) = 0 \;\forall x \in X_{n,k}$$

$$\Rightarrow$$

$\exists \alpha \in \mathbb{R}^{n+1} et\, u \in \mathbb{R}^n$ *tels que* $c(x) = \left(\alpha_0 + \sum_{i=1}^{n} \alpha_i x_i\right)\left(\sum_{j=1}^{n} x_j - k\right) + \sum_{i=1}^{n} u_i \left(x_i^2 - x_i\right)$

On a :

$c(x) = 0 \ \forall x \in X_{n,k}$

$\Rightarrow r_0 + r^t x + x^t S x + \sum_{i=1}^{n} r_{ii}\left(x_i^2 - x_i\right) = 0 \ \forall x \in X_{n,k}$

$\Rightarrow (lemme\ 1)\ \ \exists (\alpha_0, \alpha_1, \ldots, \alpha_n) \in \mathbb{R}^{n+1}$ *tels que*
$c(x) + \sum_{i=1}^{n} s_{ii}\left(x_i^2 - x_i\right) = \left(\alpha_0 + \sum_{i=1}^{n} \alpha_i x_i\right)\left(\sum_{j=1}^{n} x_j - k\right) + \sum_{i=1}^{n} \alpha_i \left(x_i^2 - x_i\right)$

$\Rightarrow \exists (\alpha_0, \alpha_1, \ldots, \alpha_n) \in \mathbb{R}^{n+1}$ *tels que*
$c(x) = \left(\alpha_0 + \sum_{i=1}^{n} \alpha_i x_i\right)\left(\sum_{j=1}^{n} x_j - k\right) + \sum_{i=1}^{n} \left(c_{ii} - \alpha_i\right)\left(x_i^2 - x_i\right)$
Cette fonction est égale à 0 pour tout $x \in X_{n,k}$.

Par conséquent, toutes les fonctions quadratiques nulles définies sur $X_{n,k}$ *peuvent être écrites comme* $\left(\alpha_0 + \sum_{i=1}^{n} \alpha_i x_i\right)\left(\sum_{j=1}^{n} x_j - k\right) + \sum_{i=1}^{n} u_i \left(x_i - x_i^2\right)$ *où* $\alpha \in \mathbb{R}^{n+1}$ *et* $u \in \mathbb{R}^n$. □

Le corollaire 3 permet ainsi de caractériser l'ensemble des fonctions nulles définies sur $X_{n,k}$. L'idée est de montrer que la reformulation QCR permet d'ajouter à $q(x)$ ***toutes*** les fonctions nulles sur le domaine admissible et de déterminer ainsi une borne optimale.

Considérons les notations du chapitre 3. Etant donnée une fonction quadratique $q(x)$, $x \in \mathbb{R}^n$, une constante $k \in \{2, \ldots, n-2\}$ et deux paramètres $\alpha \in \mathbb{R}^n$ et $u \in \mathbb{R}^n$, posons :

$$q_{\alpha,u}(x) = q(x) + \sum_{i=1}^{n} \alpha_i x_i \left(\sum_{j=1}^{n} x_j - k\right) + \sum_{i=1}^{n} u_i \left(x_i^2 - x_i\right)$$

Theorème 3 Le meilleure reformulation de (QCC) est :

$$\min \left\{ q_{\alpha^*,u^*}(x) : x \in X_{n,k} \right\}$$

où α^* et u^* satisfont la propriété suivante :

$$\min\left\{q_{\alpha^*,u^*}(x) : x \in \overline{X}_{n,k}\right\} = \max_{\substack{\alpha,u\in\mathbb{R}^n \\ q_{\alpha,u}(x)\ convexe}} \left\{\min\left\{q_{\alpha,u}(x) : x \in \overline{X}_{n,k}\right\}\right\}$$

Preuve 5 *D'après le théorème 2, la meilleure reformulation quadratique de* (QCC) *est :*

$$(QCC_{g^*}) \ : \ \min\{q(x) + g^*(x) \ : \ x \in X_{n,k}\}$$

où g^* *est une fonction quadratique convexe de* G_0 *telle que :*

$$v(\overline{QCC}_{g^*}) \ : \max_{\substack{g(x)\in G_O \\ q(x)+g(x)\ convexe}} \left\{\min\left\{q(x) + g(x) \ : \ x \in \overline{X}_{n,k}\right\}\right\} \qquad (3.4)$$

D'après le corollaire 3, (3.4) peut se réécrire :

$$v(\overline{QCC}_{g^*}) \ : \max_{\substack{\alpha\in\mathbb{R}^{n+1},u\in\mathbb{R}^n \\ q_{\alpha,u}(x)\ convexe}} \{\min \left\{q_{\alpha,u}(x) = q(x) + \left(\alpha_0 + \sum_{i=1}^n \alpha_i x_i\right)\left(\sum_{j=1}^n x_j - k\right) + \sum_{i=1}^n u_i\left(x_i^2 - x_i\right) \ : \ x \in \overline{X}_{n,k}\right\}\} \qquad (3.5)$$

Puisque $x \in \overline{X}$ *alors* $\sum_{i=1}^n x_i = k$ *et (3.5) devient :*

$$v(\overline{Q01}_{g^*}) \ : \max_{\substack{\alpha\in\mathbb{R}^{n+1},u\in\mathbb{R}^n \\ q_{\alpha,u}(x)\ convexe}} \{\min \left\{q_{\alpha,u}(x) = q(x) + \left(\sum_{i=1}^n \alpha_i x_i\right)\left(\sum_{j=1}^n x_j - k\right) + \sum_{i=1}^n u_i\left(x_i^2 - x_i\right) \ : \ x \in \overline{X}_{n,k}\right\}\}$$

On rappelle que les solutions optimales de $\max_{\substack{\alpha,u\in\mathbb{R}^n \\ q_{\alpha,u}(x)\ convexe}} \left\{\min\left\{q_{\alpha,u}(x) : x \in \overline{X}_{n,k}\right\}\right\}$ *sont obtenues par la résolution du programme semidéfini suivant (théorème*

1 du chapitre 3, [19]) :

$$
\begin{aligned}
(SDQ01) \quad \min \quad & c^t x + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} X_{ij} \\
\text{s.c.} \quad & X_{ii} = x_i && i = 1, \ldots, n \\
& -k x_i + \sum_{j=1}^{n} X_{ij} = 0 && i = 1, \ldots, n;\ k = 1, \ldots, m \\
& \sum_{i=1}^{n} x_i = k \\
& \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\
& x \in \mathbb{R}^n,\ X \in \mathbb{R}^{n \times n}
\end{aligned}
$$

□

Le theorème 3 permet de conclure que la méthode QCR construit la meilleure reformulation des problèmes de type (QCC) : elle choisit la meilleure des fonctions quadratiques nulles à ajouter parmi toutes les fonctions quadratiques nulles sur $X_{n,k}$.

Le chapitre 4 présente une étude approfondie de trois autres méthodes de résolution exacte de $(Q01)$ basée sur la reformulation quadratique convexe : celle de Hammer et Rubin [72] et celle de Billionnet et Elloumi [18] (appliquée aux problèmes $(Q01)$ non contraints) qui se révèlent être deux cas particuliers de QCR et enfin, une méthode que nous avons créée [19], légèrement différente et qui se situe dans les "prémices" de QCR.

Chapitre 4

Présentation et comparaison de cas particuliers de QCR : "les prémices"

Nous présentons dans ce chapitre différentes méthodes de résolution exacte de (Q01) fondées sur des pré-traitements consistant en une reformulation quadratique convexe du problème initial. Sans perte de généralité, on suppose que $Ax = b$ possède au moins une solution.

Considérons la reformulation QCR qui ajoute à $q(x)$ deux fonctions nulles sur l'ensemble admissible :

$$q_{\alpha,u}(x) = q(x) + q_\alpha(x) + q_u(x)$$

avec :

$q_\alpha(x) = \sum_{k=1}^{m} \left(\sum_{i=1}^{n} \alpha_{ki} x_i \right) \left(\sum_{j=1}^{n} a_{kj} x_j - b_k \right)$ et

$q_u(x) = \sum_{i=1}^{n} u_i \left(x_i^2 - x_i \right)$

où $\alpha \in \mathbb{R}^{m \times n}$ et $u \in \mathbb{R}^n$ sont des paramètres.

Les différentes méthodes de convexification que nous allons présenter utilisent le même schéma de réécriture ; elles consistent à ajouter à l'objectif une fonction nulle sur X, choisie dans une famille de fonctions définies par deux paramètres α et u. Les différentes approches sont caractérisées par les valeurs attribuées à ces paramètres : par exemple, on supposera que tous les

coefficients du paramètre α sont égaux, voire nuls, que tous les coefficients de u sont égaux, etc.

La section 4.1 s'intéresse aux prémices des méthodes de convexification : c'est la méthode de la plus petite valeur propre, développée par Hammer et Rubin en 1970 et qui peut être vue comme un cas particulier de QCR où $\alpha = O$ et $u = \lambda_{\min}(Q).e_n$. La section 4.2 présente une amélioration de la méthode d'Hammer et Rubin que nous avons développée et dont la réécriture est un peu différente de QCR. La section 4.3 étudie une première extension de la convexification de Billionnet et Elloumi au cas avec contraintes et consiste à poser $\alpha = O$ et à calculer $u \in \mathbb{R}^n$ via la programmation semidéfinie.

De manière à évaluer l'efficacité des différentes approches, nous présenterons une comparaison théorique de ces quatre reformulations dans la section 4.4. Ces reformulations seront illustrées tout au long de ce chapitre à l'aide de l'exemple (Π) ci-dessous, dont la valeur optimale est -2.

Exemple 1 :

$$\begin{array}{lll} (\Pi): & min & p(x) = x^t P x \\ & s.c. & Ax = b \\ & & A'x \leq b' \\ & & x \in \{0,1\}^5 \end{array}$$

où

$$P = \begin{pmatrix} 0 & -0.5 & -0.5 & -0.5 & -0.5 \\ -0.5 & 0 & 0 & -0.5 & 0 \\ -0.5 & 0 & 0 & 0 & 0 \\ -0.5 & -0.5 & 0 & 0 & -0.5 \\ -0.5 & 0 & 0 & -0.5 & 0 \end{pmatrix} \not\succeq 0,$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 9 & 0 & 9 \end{pmatrix},\ b = \begin{pmatrix} 3 \\ 11 \end{pmatrix}$$

$$A' = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \end{pmatrix},\ b' = 1.$$

Soit S l'ensemble des solutions réalisables de (Π).
On note p_{ij} (resp. a_{ij}) $\forall i, j$ et b_i $\forall i$ les coefficients de la matrice P (resp. A) et du

vecteur b.

4.1 La méthode de la plus petite valeur propre

Une réécriture passant par le calcul de valeurs propres

Pour résoudre de manière exacte le problème $(Q01)$, Hammer et Rubin [72] ont proposé, en 1970, une méthode qui a pour but de rendre convexe la fonction objectif. $q(x)$ est reformulée via l'utilisation des contraintes d'intégrité $x_i^2 = x_i$ $(i = 1, \ldots, n)$ et le calcul de la plus petite valeur propre de Q, qui, on le rappelle, est notée $\lambda_{\min}(Q)$.

Sans perte de généralité, nous pouvons supposer que l'ensemble admissible possède au moins une solution et que les termes diagonaux de la matrice Q sont tous nuls.

Pour tout scalaire $u_1 \in \mathbb{R}$, associons à $q(x)$ la fonction perturbée :

$$\begin{aligned} q_{u_1}(x) &= q(x) + u_1 \sum_{i=1}^{n} (x_i^2 - x_i) \\ &= x^t(Q + Diag(u_1.e_n))x - u_1 \sum_{i=1}^{n} x_i \end{aligned}$$

Pour tout $x \in \{0,1\}^n$, $x_i^2 = x_i$ et donc $q_{u_1}(x) = q(x)$. Par conséquent, résoudre $(Q01)$ revient à résoudre le problème équivalent $(Q01_{u_1})$:

$$(Q01_{u_1}) \ : \ \min \left\{ q_{u_1}(x) \ : Ax = b, A'x \leq b', x \in \{0,1\}^n \right\}$$

où on ajoute la quantité u_1 à chaque terme diagonal de la matrice Q. Avec cette réécriture, on peut supposer qu'il existe un réel u_1 tel que $(Q + Diag(u_1.e_n)) \succeq 0$. Ainsi, la relaxation continue de $(Q01_{u_1})$ peut être résolue en un temps polynomial et permet d'obtenir la borne inférieure suivante :

$$\mu(u_1) = \min \left\{ q_{u_1}(x) \ : Ax = b, A'x \leq b', x \in [0,1]^n \right\}$$

Or, la matrice $(Q + Diag(u_1.e_n)) \succeq 0$ n'est semidéfinie positive que si $\lambda_{\min}(Q + Diag(u_1.e_n)) \geq 0$ (on rappelle que $Q \not\succeq 0$ et donc $\lambda_{\min}(Q) < 0$ par définition).

Le premier objectif est donc de rendre le nouveau hessien semidéfini positif en choisissant un paramètre u_1 approprié. On sait que :

Propriété 1 $\forall u_1 \in \mathbb{R}$, $\lambda_{\min}(Q + Diag(u_1.e_n)) = \lambda_{\min}(Q) + u_1$.

Ainsi, $\lambda_{\min}(Q + Diag(u_1.e_n))$ deviendra un nombre positif ou nul si et seulement si $u_1 \geq -\lambda_{\min}(Q)$.

Le second objectif est de trouver la meilleure (donc la plus grande) borne inférieure par relaxation continue, i.e. :

$$\mu^* = \max\{\mu(u_1) \ : u_1 \geq -\lambda_{\min}(Q),\ u_1 \in \mathbb{R}\}$$

La proposition et le corollaire suivants montrent que si $u_1 = -\lambda_{\min}(Q)$ alors la borne inférieure μ^* est maximale.

Proposition 2 Soit u_1 et u_2 tels que $(Q + Diag(u_1.e_n)) \succeq 0$ et $(Q + Diag(u_2.e_n)) \succeq 0$. Si $u_1 \geq u_2$ alors $\mu(u_1) \leq \mu(u_2)$.

Preuve 6 *Soit* $u_1 \geq u_2$ *tels que* $(Q + Diag(u_1.e_n)) \succeq 0$ *et* $(Q + Diag(u_2.e_n)) \succeq 0$. *Alors, pour tout* $x \in X$,

$$\begin{aligned} q_{u_1}(x) - q_{u_2}(x) &= x^t(Diag((u_1 - u_2).e_n))x - (u_1 - u_2)\sum_{i=1}^{n} x_i \\ &= (u_1 - u_2)\sum_{i=1}^{n}(x_i^2 - x_i) \leq 0 \end{aligned}$$

Par conséquent, $\mu(u_1) \leq \mu(u_2)$.

Donc, maximiser $\mu(u_1)$ revient à trouver une valeur de u_1 la plus petite possible. Ainsi :

Corollaire 4

$$\mu^* = \mu\left(-\lambda_{min}(Q)\right) = \min\left\{q_{-\lambda_{min}(Q)}(x) \ : Ax = b, x \in [0,1]^n\right\}.$$

Algorithme de la méthode de la plus petite valeur propre

D'après la proposition 2 et le corollaire 4, un algorithme de résolution peut être élaboré pour résoudre $(Q01)$:

Le nouveau problème à résoudre en *Phase II* est :

$$(Q01_{u_1^*}) \ : \quad \min_{x \in X}\left\{x^tQx + c^tx + u_1^*\sum_{i=1}^{n}(x_i^2 - x_i)\right\}$$

Algorithme 2 Résolution exacte d'un programme quadratique en 0-1 sous contraintes linéaires par la méthode de la plus petite valeur propre

1. Calculer la plus petite valeur propre de Q : $\lambda_{\min}(Q)$
2. Résoudre le problème quadratique $\left(Q01_{-\lambda_{\min}(Q)}\right)$ dont la relaxation continue est convexe.

avec $u_1^* = -\lambda_{\min}(Q)$. On remarque que si l'on considère la reformulation QCR alors $\alpha = O$ et $u_i = u_1^* \ \forall i$.

Le nouveau hessien s'écrit :

$$Q_{u_1^*} = \begin{pmatrix} q_{11} - \lambda_{\min}(Q) & \dots & \dots & \dots & q_{1n} \\ \dots & \dots & \dots & \dots & \dots \\ q_{i1} & \dots & q_{ii} - \lambda_{\min}(Q) & \dots & q_{in} \\ \dots & \dots & \dots & \dots & \dots \\ q_{n1} & \dots & \dots & \dots & q_{nn} - \lambda_{\min}(Q) \end{pmatrix}$$

et le nouveau vecteur $c_{u_1^*}$ a pour terme général $c_i + \lambda_{\min}(Q)$.

La méthode de la plus petite valeur propre permet de calculer rapidement une borne inférieure du problème. Cet algorithme est testé dans Billionnet et Elloumi [18] pour optimiser une fonction quadratique en variables 0-1 sans contraintes. Des résultats expérimentaux, pour certaines instances de ($Q01$), sont reportés au chapitre 6.

Exemple 1 :
Résolvons le programme (Π) *par la méthode de la plus petite valeur propre. On cherche à trouver le paramètre* $u_1 \in \mathbb{R}$ *tel que :*

$$\max_{\substack{u_1 \in \mathbb{R} \\ p_{u_1}(x) \text{ convexe}}} \left\{ \min_{x \in \overline{S}} \left\{ p_{u_1}(x) = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} p_{ij} x_i x_j + u_1 \sum_{i=1}^{5} (x_i^2 - x_i) \right\} \right\}$$

Le paramètre optimal de u_1 *n'est autre que la plus petite valeur propre de* P. *Ainsi, on calcule tout d'abord cette valeur, qui est égale à* -1.34 *et on résout :*

$$(\Pi_{u_1}) \ : \ \min_{x \in S} \left\{ \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} p_{ij} x_i x_j + 1.34 \sum_{i=1}^{5} (x_i^2 - x_i) \right\}$$

dont la relaxation continue est convexe. On obtient $v(\overline{\Pi_{u_1}}) = -3.43$ $\diamondsuit$

Notre but, maintenant, est d'améliorer cette borne inférieure en perturbant cette fois-ci tous les coefficients de la matrice Q et non pas seulement ses termes diagonaux.

4.2 La méthode EQCR

Comme nous l'avons vu dans la section précédente, la méthode de la plus petite valeur propre se fonde sur un calcul des plus petites valeurs propres et sur l'utilisation, dans la reformulation, des contraintes d'intégrité du problème. Notre contribution est d'améliorer la méthode en utilisant aussi les contraintes d'égalité de l'ensemble admissible pour convexifier l'objectif. Nous noterons cette méthode EQCR pour **Eigenvalue Quadratic Convex Reformulation** [118].

Une réécriture passant par la résolution d'un programme semidéfini

La méthode EQCR, comme la méthode QCR, est une méthode en deux phases. La *Phase I* de reformulation utilise le fait que sous les contraintes $Ax = b$, on a $\beta(Ax-b)^t(Ax-b) = 0\ \forall \beta \in \mathbb{R}$. Un problème $(Q01'_\beta)$, équivalent à $(Q01)$, est par conséquent obtenu en ajoutant $\beta(Ax-b)^t(Ax-b)$ à la fonction objectif. $(Q01'_\beta)$ est ensuite transformé grâce à la méthode de la plus petite valeur propre.

Cette partie détaille notre nouvelle approche qui, comme les méthodes présentées dans ce chapitre, transforme de manière adéquate la fonction objectif par l'ajout de fonctions nulles.

Tout d'abord, on définit un nombre réel β et une nouvelle fonction $q'_\beta(x)$:

$$\begin{aligned} q'_\beta(x) &= q(x) + \beta\,(Ax-b)^t\,(Ax-b) \\ &= x^tQ_\beta x + c_\beta^t x + l_\beta \end{aligned}$$

où $Q_\beta = Q + \beta A^tA$, $c_\beta = c - 2\beta A^tb$ et $l_\beta = \beta b^tb$.
Clairement, pour tout $x \in X$, $q'_\beta(x) = q(x)$.

Notre première idée est d'ajouter la fonction nulle $\beta\,(Ax-b)^t\,(Ax-b)$ à la fonction objectif de $(Q01)$. Une question se pose alors : existe-t-il un paramètre $\beta \in \mathbb{R}$ tel que $q'_\beta(x)$ convexe ?

Proposition 3 Il existe au moins une fonction $q(x)$ telle que $\forall \beta \in \mathbb{R}$, $q'_\beta(x)$ n'est pas convexe.

Preuve 7 *Reprenons le schéma de démonstration de la preuve 2 et considérons le problème du k-cluster pour un graphe G de n sommets $\{v_1, \ldots, v_n\}$. On cherche à maximiser le nombre d'arêtes inclues dans un sous-ensemble S de k sommets. La modélisation est la suivante :*

$$(KC) \quad : \quad \min \left\{ f(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} -\delta_{ij} x_i x_j : \sum_{i=1}^{n} x_i = k,\ x \in \{0,1\}^n \right\}$$

Les coefficients binaires δ_{ij} prennent la valeur 1 si et seulement si $[v_i, v_j]$ est une arête de G. La variable binaire x_i est égale à 1 si et seulement si le sommet v_i est dans le sous-ensemble S. Pour plus de détails, le problème du k-cluster est présenté à la section 6.2.

Nous allons montrer que $f(x) + f'_\beta(x)$ n'est jamais convexe quelque soit les valeurs de $\beta \in \mathbb{R}$.

On a : $f(x) + f_\beta(x) = x^t Q_\beta x + c_\beta^t x$ *avec* $Q_{\beta_{ij}} = \begin{cases} \beta & si \quad i = j \\ -\frac{1}{2}\delta_{ij} + \beta & sinon \end{cases}$

Montrons qu'il existe un vecteur $\tilde{x} \in \mathbb{R}^n$ non nul tel que $\tilde{x}^t Q_\beta \tilde{x} < 0$ (voir la Propriété 3 de l'Annexe 1).

$$\tilde{x}^t Q_\beta \tilde{x} = \sum_{i=1}^{n} \beta \tilde{x}_i^2 + \sum_{i=1}^{n} \sum_{j=i+1}^{n} (2\beta - \delta_{ij})\, \tilde{x}_i \tilde{x}_j .$$

Soient 4 indices k, l, m, p tels que $k < l < m < p$ et tels que :

- $\delta_{kl} = \delta_{mp} = 1$ *et*
- $\delta_{km} = \delta_{kp} = \delta_{lm} = \delta_{lp} = 0$

Autrement dit, on considère deux arêtes (v_k, v_l) et (v_m, v_p) disjointes, ce qui n'est pas restrictif pour des graphes de grande taille.

Posons maintenant $\tilde{x_k} = \tilde{x_l} = 1$; $\tilde{x_m} = \tilde{x_p} = -1$ et $\tilde{x_i} = 0$ $\forall i \neq (k, l, m, p)$.

Ainsi,

$$\tilde{x}^t Q_\beta \tilde{x} = -2 < 0$$

$\forall \beta \in \mathbb{R}$, *il existe donc un vecteur* $x \in \mathbb{R}^n$, *non nul, tel que* $x^t Q_\beta x < 0$. Q_β *n'est donc pas semidéfinie positive.* □

D'après la proposition 3, la fonction $q'_\beta(x)$ n'est pas convexe. Il faut donc rendre le hessien semidéfini positif. Si l'on définit la fonction suivante : $q_{\beta,\lambda}(x) = q'_\beta(x) + \lambda \sum_{i=1}^{n} \left(x_i^2 - x_i\right)$, il faut trouver le paramètre $\lambda \in \mathbb{R}$ qui convexifie la fonction et qui permet d'obtenir la plus grande borne inférieure par relaxation continue. D'après la section 4.1, la meilleure valeur de λ n'est autre que $-\lambda_{\min}(Q_\beta)$. On ajoute donc $-\lambda_{\min}(Q_\beta) \sum_{i=1}^{n}(x_i^2 - x_i)$ à $q'_\beta(x)$ et un nouveau problème $(Q01_\beta)$ est défini, équivalent à $(Q01)$:

$$(Q01_\beta) \ : \ \min \left\{ q_\beta(x) = q'_\beta(x) - \lambda_{\min}(Q_\beta) \sum_{i=1}^{n}(x_i^2 - x_i) \ : Ax = b, A'x \leq b', \ x \in \{0,1\}^n \right\}$$

Soit $\gamma(\beta)$ la valeur optimale de la relaxation continue de $(Q01_\beta)$, i.e. :

$$\gamma(\beta) = \min \left\{ q_\beta(x) \ : Ax = b, A'x \leq b', \ x \in [0,1]^n \right\}$$

On note que $\gamma(0) = \mu^*$ (cf. section 4.1). Il suffit donc maintenant d'ajuster le paramètre $\beta \in \mathbb{R}$ pour obtenir la plus grande valeur de $\gamma(\beta)$. Nous allons montrer tout d'abord que $\gamma(\beta)$ est une fonction croissante en β et donc que $\gamma(\beta) \geq \mu^*$, $\forall \beta \geq 0$.

Proposition 4 Si $\beta_1 \geq \beta_2$ alors $\lambda_{\min}(Q_{\beta_1}) \geq \lambda_{\min}(Q_{\beta_2})$.

Preuve 8 *Supposons que* $\beta_1 \geq \beta_2$.
Nous utilisons ici l'expression de Rayleigh des valeurs propres extrêmes et, en particulier, celle de la plus petite valeur propre :

$$\lambda_{min}(Q_{\beta_1}) = \min_{\|x\|=1} \ x^t Q_{\beta_1} x \quad et$$

$$\lambda_{min}(Q_{\beta_2}) = \min_{\|x\|=1} \ x^t Q_{\beta_2} x$$

.

Ainsi, quelque soit $x \in \mathbb{R}^n$, *on a* $x^t Q_{\beta_1} x - x^t Q_{\beta_2} x = (\beta_1 - \beta_2) x^t A^t A x \geq 0$ *puisque* $A^t A \succeq 0$.

D'après cette dernière inégalité, on peut déduire que $\lambda_{min}(Q_{\beta_1}) \geq \lambda_{min}(Q_{\beta_2})$.
□

Proposition 5 Soient β_1 et β_2 tels que $\lambda_{\min}(Q_{\beta_1}) \geq \lambda_{\min}(Q_{\beta_2})$ alors $\gamma(\beta_1) \geq \gamma(\beta_2)$.

Preuve 9 *Soit $x \in [0,1]^n$ tel que $Ax = b$. On a :*

$$\begin{aligned} q_{\beta_1}(x) - q_{\beta_2}(x) &= q(x) + \beta_1 (Ax-b)^t (Ax-b) - \lambda_{min}(Q_{\beta_1}) \sum_{i=1}^{n} (x_i^2 - x_i) \\ &\quad -q(x) - \beta_2 (Ax-b)^t (Ax-b) + \lambda_{min}(Q_{\beta_2}) \sum_{i=1}^{n} (x_i^2 - x_i) \\ &= (\lambda_{min}(Q_{\beta_2}) - \lambda_{min}(Q_{\beta_1})) \sum_{i=1}^{n} (x_i^2 - x_i) \geq 0 \end{aligned}$$

Par conséquent, pour toute solution réalisable, $q_{\beta_1}(x) \geq q_{\beta_2}(x)$. Ainsi, $\gamma(\beta_1) \geq \gamma(\beta_2)$. □

Les propositions 4 et 5 induisent le corollaire suivant :

Corollaire 5 Si $\beta_1 \geq \beta_2$ alors $\gamma(\beta_1) \geq \gamma(\beta_2)$

Ainsi, d'après le Corollaire 5, plus grand est le paramètre β, meilleure est la borne, et donc maximiser $\gamma(\beta)$ est équivalent au problème suivant :

$$(D) \qquad \max \{\lambda_{\min}(Q_\beta) \ : \beta \in \mathbb{R}\}$$

Il est connu [79] que ce problème peut être vu comme un programme semidéfini dont le dual est :

$$\begin{aligned} (SDD): \quad \min \quad & Q \bullet X \\ \text{s.c.} \quad & \\ & tr(X) = 1 \\ & A^t A \bullet X = 0 \\ & X \succeq 0 \end{aligned} \tag{4.1}$$

où $A \bullet B = tr(A^t B) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} B_{ij}$ pour toute matrice A et B de S_n.

Proposition 6 Si $(Q01)$ admet au moins une solution réalisable alors (SDD) en admet au moins une également.

Preuve 10 *Posons $x \neq 0 \in \mathbb{R}^n$ solution de $Ax = 0$ (par hypothèse, l'équation linéaire $Ax = b$ a au moins une solution).*
Soient $v \in \mathbb{R}^n$ tel que $v = \frac{x}{\|x\|}$ et $V = v^t v$.
D'après la première hypothèse de la preuve, on a $Av = 0$. De plus,

- $tr(V) = \sum_{i=1}^{n} v_i^2 = \sum_{i=1}^{n} \left(\frac{x_i}{\|x\|}\right)^2 = \frac{1}{\left(\sqrt{\sum_{i=1}^{n} x_i^2}\right)^2} \sum_{i=1}^{n} x_i^2 = 1.$
- $V \succeq 0$ *car* V *est le produit d'un vecteur et de sa transposée.*
- $Av = 0 \Rightarrow A^t A.v^t v = A^t A \bullet V = 0$

Ainsi, V *est une solution réalisable de* (SDD). □

La proposition 6 nous permet de conclure que le problème dual (D) est borné. Sa solution optimale est notée β^*. β^* est également la valeur optimale de la variable duale associée à la contrainte (4.1) du programme (SDD) et $\lambda_{\min}(Q_{\beta^*})$ est précisément la valeur optimale de (SDD) et de (D).

Algorithme EQCR

Tous les résultats précédents permettent d'élaborer un nouvel algorithme de résolution, basé sur la reformulation EQCR :

Algorithme 3 Résolution exacte d'un programme quadratique en 0-1 sous contraintes linéaires par la méthode EQCR

1. Résoudre le programme (SDD).
Soit β^* la valeur optimale de la variable duale associée à la contrainte (4.1) du programme (SDD) et $\lambda_{\min}(Q_{\beta^*})$ la valeur optimale de (SDD).
2. Résoudre le problème quadratique $(Q01_{\beta^*})$ dont la relaxation continue est convexe.

Le nouveau problème à résoudre en *Phase II* a la forme générale :

$$(Q01_{\beta^*}) \; : \; \min_{x \in X} \left\{ x^t Q x + c^t x + \beta^* \sum_{k=1}^{m} \left(\sum_{j=1}^{n} a_{kj} x_j - b_k \right)^2 - \lambda_{\min}(Q_{\beta^*}) \sum_{i=1}^{n} \left(x_i^2 - x_i \right) \right\}$$

Le nouveau hessien Q_{β^*} est donc :

$$Q_{\beta^*} = \begin{pmatrix} q_{11} + \beta^* \sum_{k=1}^{m} a_{k1} - \lambda_{\min}(Q_{\beta^*}) & q_{12} + \beta^* \sum_{k=1}^{m} a_{k1}a_{k2} & \dots & q_{1n} + \beta^* \sum_{k=1}^{m} a_{k1}a_{kn} \\ q_{21} + \beta^* \sum_{k=1}^{m} a_{k1}a_{k2} & q_{22} + \beta^* \sum_{k=1}^{m} a_{k2} - \lambda_{\min}(Q_{\beta^*}) & \dots & q_{2n} + \beta^* \sum_{k=1}^{m} a_{k2}a_{kn} \\ \dots & \dots & \dots & \dots \\ q_{n1} + \beta^* \sum_{k=1}^{m} a_{k1}a_{kn} & q_{n2} + \beta^* \sum_{k=1}^{m} a_{kn}a_{k2} & \dots & q_{n1} + \beta^* \sum_{k=1}^{m} a_{kn} - \lambda_{\min}(Q_{\beta^*}) \end{pmatrix}$$

et le nouveau vecteur c_{β^*} a pour terme général $c_i - \sum_{k=1}^{m} b_k a_{ki} + \lambda_{\min}(Q_{\beta^*})$.

Notons que pour $\beta = 0$, la méthode EQCR en deux phases est celle de la plus petite valeur propre (voir [72] et section 4.1).

Exemple 1 :
Résolvons le programme (Π) *par la méthode EQCR.*
On cherche le paramètre $\beta \in \mathbb{R}$ *tel que :*

$$\max_{\substack{\beta\in\mathbb{R} \\ p_\beta(x)\ convexe}} \left\{ \min_{x\in S} \left\{ p_\beta(x) = \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} p_{ij}x_ix_j + \beta\sum_{k=1}^{m}\left(\sum_{j=1}^{n} a_{kj}x_j - b_k\right)^2 - \lambda_{min}(Q_\beta)\sum_{i=1}^{n}\left(x_i^2 - x_i\right) \right\} \right\}$$

Pour cela, on résout tout d'abord le programme semidéfini suivant :

$$\begin{array}{lll} (SD\Pi_\beta): & min & \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} p_{ij}X_{ij} \\ & \text{s.c.} & \\ & & \sum_{i=1}^{5} X_{ii} = 1 \\ & & 2X_{11} + 4X_{12} + 20X_{13} + 2X_{14} + 20X_{15} + 2X_{22} + 20X_{23} + 2X_{24} \\ & & +20X_{25} + 82X_{33} + 2X_{34} + 164X_{35} + X_{44} + 2X_{45} + 82X_{55} = 0 \quad \leftarrow \beta^* \\ & & \\ & & X \succeq 0 \end{array}$$

afin de récupérer la valeur optimale de β, β^*, *et la valeur* $\lambda_{min}(Q_{\beta^*})$, *qui n'est autre que la valeur optimale de* $(SD\Pi_\beta)$ *:*

$$\beta^* = 219.37, \quad -\lambda_{min}(Q_{\beta^*}) = 0.12 \ (= -\lambda_{min}(P + 210.57A^tA))$$

On peut maintenant résoudre le programme quadratique en 0-1 suivant :

$$(\Pi'_{\beta^*}) \ : \ \min_{x\in S} \left\{ \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} p_{ij}x_ix_j + \beta^*\sum_{k=1}^{m}\left(\sum_{j=1}^{n} a_{kj}x_j - b_k\right)^2 - \lambda_{min}(Q_{\beta^*})\sum_{i=1}^{n}\left(x_i^2 - x_i\right) \right\}$$

dont la relaxation continue est convexe. On obtient $v(\overline{\Pi'_{\beta^*}}) = -2.29$. $\diamondsuit$

4.3 La méthode IQCR

Une réécriture passant par la résolution d'un programme semidéfini particulier

Nous nous sommes intéressés au cas particulier de QCR où $\alpha = O$, c'est-à-dire :

$$(Q01_{u_2}) \ : \ \min_{x\in X} \left\{ x^tQx + c^tx + \sum_{i=1}^{n} u_{2_i}\left(x_i^2 - x_i\right) \right\}$$

où $u_2 \in \mathbb{R}^n$ est déterminé grâce à la relaxation semidéfinie suivante :

$$(SDQ01_2): \quad \min \quad \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} X_{ij}$$

$$\text{s.c.}$$

$$\begin{aligned} & X_{ii} = x_i \qquad\qquad i = 1, \ldots, n \qquad (4.2) \\ & Ax = b \\ & A'x \leq b' \\ & \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\ & x \in \mathbb{R}^n, \ X \in S_n \end{aligned}$$

Le vecteur optimal de u_2, noté u_2^*, est donné par les valeurs optimales des variables duales associées aux contraintes $X_{ii} = x_i$.
Cette approche est notée **IQCR** pour **Integrity Quadratic Convex Reformulation**.

Algorithme IQCR

L'algorithme de résolution est donc le suivant :

Algorithme 4 Résolution exacte d'un programme quadratique en 0-1 sous contraintes linéaires par la reformulation $(Q01_{u_2})$

1. Résoudre le programme semidéfini $(SDQ01_2)$:
Soit $u_{2_i}^*$ les valeurs optimales des variables duales associées aux n contraintes (4.2).
2. Résoudre le programme quadratique $(Q01_{u_2^*})$ dont la relaxation continue est convexe.

Le nouveau problème à résoudre est alors :

$$(Q01_{u_2^*}) \ : \ \min_{x \in X} \left\{ x^t Q x + c^t x + \sum_{i=1}^{n} u_{2_i}^* \left(x_i^2 - x_i \right) \right\}$$

ou encore

$$(Q01_{u_2^*}) \ : \ \min_{x \in X} \left\{ x^t Q_{u_2^*} x + c_{u_2^*}^t x \right\}$$

où le nouveau hessien s'écrit :

$$Q_{u_2^*} = \begin{pmatrix} q_{11} + u_{2_1}^* & q_{12} & \dots & q_{1n} \\ q_{21} & q_{12} + u_{2_2}^* & \dots & q_{2n} \\ \dots & \dots & \dots & \dots \\ q_{n1} & q_{n2} & \dots & q_{2n} + u_{2_n}^* \end{pmatrix}$$

et le nouveau vecteur $c_{u_2^*}$ a pour terme général $c_i - u_{2_i}^*$.

Exemple 1 :

Résolvons le programme (Π) *par la méthode IQCR.*

On cherche le paramètre $u_2 \in \mathbb{R}^5$ *tel que :*

$$\underset{\substack{u_2 \in \mathbb{R}^5 \\ p_{u_2}(x) \text{ convexe}}}{max} \left\{ \underset{x \in \overline{S}}{min} \left\{ p_{u_2}(x) = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} p_{ij} x_i x_j + \sum_{i=1}^{5} u_{2_i} (x_i^2 - x_i) \right\} \right\}$$

Pour cela, on résout tout d'abord le programme semidéfini suivant :

$$(SD\Pi_{u_2}) : \quad \underset{s.c.}{min} \quad \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} p_{ij} X_{ij}$$

$$\begin{aligned} & X_{ii} = x_i \quad i = 1, \dots, n && \leftarrow u_{2_i}^* \\ & x_1 + x_2 + x_3 + x_4 + x_5 = 3 \\ & x_1 + x_2 + 9x_3 + 9x_5 = 11 \\ & x_1 + x_4 \leq 1 \\ & \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\ & x \in \mathbb{R}^5, \ X \in S_5 \end{aligned}$$

afin de récupérer les valeurs duales optimales $u_{2_i}^* \ \forall i = 1, \dots, 5$ *:*

$$u_2^* = \begin{pmatrix} 1.68 \\ 2.1 \\ 0.5 \\ 1.2 \\ 1.06 \end{pmatrix}$$

On peut ensuite résoudre le programme quadratique en 0-1 suivant :

$$(\Pi_{u_2^*}) \ : \underset{x \in S}{min} \left\{ \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} p_{ij} x_i x_j + \sum_{i=1}^{5} u_{2_i}^* (x_i^2 - x_i) \right\}$$

dont la relaxation continue est convexe. On obtient $v(\overline{\Pi_{u_2^*}}) = -3.27$. $\diamondsuit$

Dans la section suivante, nous allons comparer théoriquement les quatre reformulations $(Q01_{u_1^*})$, $(Q01_{u_2^*})$, $(Q01_{\beta^*})$, $(Q01_{\alpha^*,u^*})$.

4.4 Comparaison théorique des différentes reformulations

Nous allons montrer que la reformulation QCR est meilleure que les trois reformulations $(Q01_{u_1^*})$, $(Q01_{u_2^*})$ et $(Q01_{\beta^*})$. En effet, les propositions suivantes démontrent que la borne inférieure obtenue par relaxation continue de $(Q01_{\alpha^*,u^*})$ est supérieure ou égale à celles obtenues par les 3 autres relaxations.

Proposition 7 Soient les trois problèmes $(Q01_{u_1^*})$, $(Q01_{u_2^*})$ et $(Q01_{\beta^*})$ définis respectivement dans les sections 4.1, 4.3 et 4.2.
La solution u_1^* est obtenue en résolvant le problème suivant :

$$(pb1) \ : \max_{\substack{u_1\in\mathbb{R}\\ Q_{u_1}\succeq 0}} \left\{ L_1(u_1) = \min_{\substack{Ax=b\\ A'x\leq b'\\ 0\leq x\leq 1}} \left\{ \Phi^1_{u_1}(x) = q(x) + u_1 \sum_{i=1}^{n} \left(x_i^2 - x_i\right) \right\} \right\}$$

Or, $(pb1)$ n'est autre que

$$(pb1) \ : L_1 = \min_{\substack{Ax=b\\ A'x\leq b'\\ 0\leq x\leq 1}} \left\{ \Phi^1(x) = q(x) - \lambda_{\min}(Q) \sum_{i=1}^{n} \left(x_i^2 - x_i\right) \right\}$$

puisque $u_1^* = -\lambda_{\min}(Q)$.

La solution β^* est obtenue en résolvant :

$$(pb2) \ : \max_{\substack{\beta\in\mathbb{R}\\ Q_\beta\succeq 0}} \left\{ L_2(\beta) = \min_{\substack{Ax=b\\ A'x\leq b'\\ 0\leq x\leq 1}} \left\{ \Phi^2_\beta(x) = q(x) + \beta \sum_{k=1}^{m} \left(\sum_{j=1}^{n} a_{kj}x_j - b_k \right)^2 - \lambda_{\min}(Q_\beta) \sum_{i=1}^{n} \left(x_i^2 - x_i\right) \right\} \right\}$$

Enfin, la solution u_2^* est obtenue en résolvant :

$$(pb3) \ : \max_{\substack{u_2\in\mathbb{R}^n\\ Q_{u_2}\succeq 0}} \left\{ L_3(u_2) = \min_{\substack{Ax=b\\ A'x\leq b'\\ 0\leq x\leq 1}} \left\{ \Phi^3_{u_2}(x) = q(x) + \sum_{i=1}^{n} u_{2_i} \left(x_i^2 - x_i\right) \right\} \right\}$$

Les valeurs optimales de $(pb2)$ et $(pb3)$ sont plus grandes ou égales que celle de $(pb1)$. Par conséquent, $(Q01_{u_2^*})$ et $(Q01_{\beta^*})$ sont de meilleures relaxations que $(Q01_{u_1^*})$.

Preuve 11 *On se fonde sur le schéma de preuve suivant : soient deux problèmes de maximisation Π_1 et Π_2. Si à partir de toute solution optimale de Π_1, on parvient à construire une solution réalisable de Π_2 avec égalité des deux fonctions objectif, alors $v(\Pi_1) \leq v(\Pi_2)$.*

Si on pose $\tilde{\beta}$ tel que $\tilde{\beta} = O$ alors $\tilde{\beta}$ est une solution réalisable de (pb2) puisque $\Phi^2_{\tilde{\beta}}(x)$ et $\Phi^1(x)$ ont précisément le même hessien $Q_{-\lambda_{min}(Q)}$, qui est semidéfini positif d'après (pb1). Ainsi, à partir de l'unique solution optimale $u_1^* = -\lambda_{min}(Q)$ de (pb1), on peut construire une solution admissible de (pb2).
De plus, on a, pour tout $x \in X$

$$\Phi^1(x) = \Phi^2_{\tilde{\beta}}(x) = q(x) - \lambda_{min}(Q)\sum_{i=1}^{n}\left(x_i^2 - x_i\right)$$

et donc $L_2(\tilde{\beta}) = L_1$.
On a ainsi montré que la valeur optimale de (pb2) est supérieure ou égale à celle de (pb1).

Montrons maintenant que $v(pb1) \leq v(pb3)$ (on utilise le même schéma de preuve).

Si on pose $\tilde{u_2}$ tel que $\tilde{u_2}_i = -\lambda_{min}(Q)$ pour tout i, alors $\tilde{u_2}$ est une solution réalisable de (pb3) puisque $\Phi^3_{\tilde{u_2}}(x)$ et $\Phi^1(x)$ ont le même hessien.
De plus, on a, pour tout $x \in X$

$$\Phi^1(x) = \Phi^3_{\tilde{u_2}}(x) = q(x) - \lambda_{min}(Q)\sum_{i=1}^{n}\left(x_i^2 - x_i\right)$$

et ainsi $L_3(\tilde{u_2}) = L_1$. □

Proposition 8 Soient les problèmes $(Q01_{\alpha^*,u^*})$ et $(Q01_{\beta^*})$, définis respectivement dans le chapitre 3 et la section 4.2. La solution (α^*, u^*) est obtenue en résolvant le problème suivant :

$$(pb4) \; : \max_{\substack{\alpha\in\mathbb{R}^{m\times n}\\ u\in\mathbb{R}^n\\ Q_{\alpha,u}\succeq 0}} \left\{ L_4(\alpha,u) = \min_{\substack{Ax=b\\ A'x\leq b'\\ 0\leq x\leq 1}} \left\{ \Phi^4_{\alpha,u}(x) = q(x) + \sum_{i=1}^{n}\sum_{k=1}^{m}\alpha_{ki}x_i\left(\sum_{j=1}^{n}a_{kj}x_j - b_k\right) + \sum_{i=1}^{n}u_i\left(x_i^2 - x_i\right)\right\}\right\}$$

et la solution β^* est obtenue en résolvant :

$$(pb2) : \max_{\substack{\beta\in\mathbb{R}\\ Q_\beta\succeq 0}} \left\{ L_2(\beta) = \min_{\substack{Ax=b\\ A'x\leq b'\\ 0\leq x\leq 1}} \left\{ \Phi^2_\beta(x) = q(x) + \beta\sum_{k=1}^{m}\left(\sum_{j=1}^{n} a_{kj}x_j - b_k\right)^2 - \lambda_{\min}(Q_\beta)\sum_{i=1}^{n}\left(x_i^2 - x_i\right)\right\}\right\}$$

La valeur optimale de $(pb4)$ est supérieure ou égale à celle de $(pb2)$. Par conséquent, $(Q01_{\alpha^*,u^*})$ est une meilleure relaxation que $(Q01_{\beta^*})$.

Preuve 12 *On se fonde maintenant sur le schéma de preuve suivant : soient deux problèmes de maximisation Π_1 et Π_2. Si à partir de toute solution réalisable de Π_1, on parvient à construire une solution réalisable de Π_2 avec égalité des deux fonctions objectif, alors $v(\Pi_1) \leq v(\Pi_2)$.*

Tout d'abord, on remarque que $\Phi^2_\beta(x)$ peut être réécrit comme suit :

$$\Phi^2_\beta(x) = q(x) + \sum_{k=1}^{m}\sum_{i=1}^{n}\beta\, a_{ki}x_i\sum_{j=1}^{n}(a_{kj}x_j - b_k) - \beta\sum_{k=1}^{m}b_k\left(\sum_{j=1}^{n}a_{kj}x_j - b_k\right) - \lambda_{min}(Q_\beta)\sum_{i=1}^{n}\left(x_i^2 - x_i\right)$$

Soit β une solution réalisable de $(pb2)$. Si on pose $(\tilde{\alpha}, \tilde{u})$ tels que $\tilde{\alpha}_{ki} = \beta a_{ki}$ et $\tilde{u}_i = -\lambda_{min}(Q_\beta)$, alors $(\tilde{\alpha}, \tilde{u})$ est une solution réalisable de $(pb4)$ puisque $\Phi^4_{\tilde{\alpha},\tilde{u}}(x)$ et $\Phi^2_\beta(x)$ ont précisément le même hessien.
De plus, on remarque que, pour tout x tel que $Ax = b$,

$$\Phi^2_\beta(x) = \Phi^4_{\tilde{\alpha},\tilde{u}}(x) = q(x) - \lambda_{min}(Q_\beta)\sum_{i=1}^{n}\left(x_i^2 - x_i\right)$$

Cette dernière égalité montre que $L_4(\tilde{\alpha}, \tilde{u}) = L_2(\beta)$. □

Proposition 9 Soient les problèmes $(Q01_{\alpha^*,u^*})$ et $(Q01_{u_2^*})$, définis respectivement dans le chapitre 3 et la section 4.3. La solution (α^*, u^*) est obtenue en résolvant le problème $(pb4)$ défini dans la proposition 8 et la solution u_2^* est obtenue en résolvant $(pb3)$ défini dans la proposition 7.
$(Q01_{\alpha^*,u^*})$ est une meilleure relaxation que $(Q01_{u_2^*})$.

Cette proposition peut être démontrée selon le même schéma de preuve que la proposition 8.

D'après les quatre propositions précédentes, nous pouvons contruire le schéma relationnel suivant :

$$\begin{array}{ccc} (Q01_{u_1^*}) & \rightarrow & (Q01_{\beta^*}) \\ \downarrow & & \downarrow \\ (Q01_{u_2^*}) & \rightarrow & (Q01_{\alpha^*,u^*}) \end{array}$$

où $\Pi_i \rightarrow \Pi_j$ signifie que $v(\overline{\Pi_i}) \leq v(\overline{\Pi_j})$.
La méthode QCR est donc une meilleure reformulation.

Exemple 1 :
Résolvons le programme (Π) *par la méthode QCR.*
On cherche à trouver les meilleures paramètres $\alpha \in \mathbb{R}^{2\times 5}$ *et* $u \in \mathbb{R}^5$ *tels que :*

$$\max_{\substack{\alpha\in\mathbb{R}^{2\times 5}, u\in\mathbb{R}^5 \\ p_{\alpha,u}(x)\ convexe}} \left\{ \min_{x\in S} \left\{ p_{\alpha,u}(x) = \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} p_{ij}x_ix_j + \sum_{i=1}^{5}\sum_{k=1}^{2}\alpha_{ki}x_i\left(\sum_{j=1}^{5}a_{kj}x_j - b_k\right) + \sum_{i=1}^{5}u_i\left(x_i^2 - x_i\right) \right\} \right\}$$

Pour cela, il faut tout d'abord résoudre le programme semidéfini suivant :

$$\begin{array}{lll} (SD\Pi_{\alpha,u}): \quad min & \sum_{i=1}^{5}\sum_{j=1,j\neq i}^{n} p_{ij}X_{ij} & \\ s.c. & & \\ & X_{ii} = x_i & i = 1,..,n \quad \leftarrow u_i^* \\ & -3x_i + \sum_{j=1}^{5} X_{ij} = 0 & i = 1,\ldots,5 \quad \leftarrow \alpha_{1i}^* \\ & -11x_i + X_{i1} + X_{i2} + 9X_{i3} + 9X_{i5} = 0 & i = 1,\ldots,5 \quad \leftarrow \alpha_{2i}^* \\ & x_1 + x_2 + x_3 + x_4 + x_5 = 3 & \\ & x_1 + x_2 + 9x_3 + 9x_5 = 11 & \\ & x_1 + x_4 \leq 4 & \\ & \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 & \\ & x \in \mathbb{R}^5,\ X \in S_5 & \end{array}$$

afin de récupérer les valeurs duales optimales de α *et* u *:*

$$\alpha^* = \begin{pmatrix} 1523.18 & 1520.38 & -1254.08 & 1870.78 & -1253.08 \\ -280.34 & -280.23 & 497.02 & -377.56 & 496.02 \end{pmatrix}, u^* = \begin{pmatrix} -0.1 \\ 3.04 \\ -4 \\ -0.84 \\ 4 \end{pmatrix}$$

On peut maintenant résoudre le programme quadratique en 0-1 suivant :

$$(\Pi_{\alpha^*,u^*}) \ : \ \min_{x\in S}\left\{\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} p_{ij}x_ix_j + \sum_{i=1}^{5}\sum_{k=1}^{2}\alpha^*_{ki}x_i\left(\sum_{j=1}^{5}a_{kj}x_j - b_k\right) + \sum_{i=1}^{5}u^*_i\left(x_i^2 - x_i\right)\right\}$$

dont la relaxation continue est convexe. On obtient $= v(\overline{\Pi_{\alpha^*,u^*}}) = -2.005$.

Comparaison des quatre méthodes de résolutions *:*

Le programme (Π) *a été reformulé selon l'approche de la plus petite valeur propre, EQCR, IQCR et QCR. Cet exemple montre que la méthode QCR peut aussi être strictement meilleure que les trois autres méthodes de reformulation.*

Le tableau suivant résume les résultats obtenus pour les différentes reformulations quadratiques convexes :

	borne	*gap*
(Π_{u_1})	-3.43	71.5%
(Π_{u_2})	-3.27	63.5%
(Π'_{β^*})	-2.29	14.5%
(Π_{α^*,u^*})	$\mathbf{-2.005}$	$\mathbf{0.2}\%$

On rappelle que la valeur optimale de Π *est égale à -2.*
La colonne gap représente le saut d'intégrité entre la valeur de la borne inférieure obtenue par relaxation continue et la valeur optimale. Ainsi,

$$gap = \mid \frac{opt - borne}{opt} \mid$$

.
Bien sûr, les temps de résolution en 0-1 sont très rapides et donc incomparables pour notre exemple.

On voit clairement que, même sur cet exemple de petite taille, la méthode QCR permet d'obtenir un saut d'intégrité très inférieur à ceux obtenus avec les trois autres reformulations. Ainsi, on peut en conclure que l'on peut avoir :

$$v(\overline{Q01_{\alpha^*,u^*}}) > v(\overline{Q01_{u_2^*}})$$

$$v(\overline{Q01_{\alpha^*,u^*}}) > v(\overline{Q01_{\beta^*}})$$

et

$$v(\overline{Q01_{\alpha^*,u^*}}) > v(\overline{Q01_{u_1^*}})$$

♢

La comparaison théorique de la méthode QCR avec trois autres méthodes de reformulation quadratique convexe permet de conclure à son efficacité en terme de calcul de bornes. Des comparaisons expérimentales seront réalisées aux chapitre 6.

La reformulation QCR s'appuie sur la résolution d'un programme semidéfini. Plus précisément, elle se contruit à partir de la résolution d'une relaxation semidéfinie. Dans le chapitre suivant, nous allons mettre en évidence le parallèle qui existe entre QCR et une nouvelle reformulation, cette fois-ci linéaire. En effet, cette méthode utilise, quant à elle, les solutions d'une relaxation linéaire particulière.

Chapitre 5

Une nouvelle méthode de linéarisation compacte et sa mise en parallèle avec QCR

Dans ce chapitre, nous présentons une famille de reformulations linéaires compactes inspirée de la linéarisation de Glover et construite à partir d'une écriture de la fonction objectif sous forme d'une constante plus une certaine fonction positive sur le domaine $\overline{X}$ (section 5.1). Nous montrons qu'une bonne reformulation linéaire compacte peut-être obtenue par la résolution d'une relaxation linéaire. Ensuite, un parallèle sera établi avec la reformulation QCR où la fonction objectif est réécrite sous forme d'une constante plus une fonction quadratique convexe sur $\mathbb{R}^n$ et on montre qu'une reformulation quadratique convexe peut être obtenue grâce à une relaxation semidéfinie (section 5.2).

Exemple 2 :

Tout au long de ce chapitre, nous allons étudier les différentes reformulations du programme quadratique (E) suivant, dont la valeur optimale est -65 :

$$\begin{aligned}
\min \quad & \phi(x) = -9x_1 - 7x_2 + 2x_3 + 23x_4 + 12x_5 - 48x_1x_2 + 4x_1x_3 + 36x_1x_4 \\
& -24x_1x_5 - 7x_2x_3 + 36x_2x_4 - 84x_2x_5 + 40x_3x_4 + 4x_3x_5 - 88x_4x_5 \\
\text{s.c.} \quad & \\
& x_1 - 2x_2 + 5x_3 + 2x_4 - 2x_5 \geq 2 \\
& x_1 + x_2 + x_4 + x_5 = 2 \\
& x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}
\end{aligned}$$

♢

5.1 Construction d'une reformulation linéaire compacte positive utilisant la borne trouvée par une bonne relaxation linéaire (la relaxation RLT)

Une reformulation linéaire compacte positive [20]
La reformulation que nous proposons suppose d'abord que l'on identifie une constante c et des fonctions linéaires positives ou nulles sur le domaine relâché $\overline{X}$, $L(x)$, $f_i(x)$ et $g_i(x)$ pour $i = 1, \ldots, n$ de telle sorte que :

$$\forall x \in X, \quad q(x) = c + L(x) + \sum_{i=1}^{n} x_i f_i(x) + \sum_{i=1}^{n} (1 - x_i) g_i(x) \tag{5.1}$$

Un tel choix est toujours possible puisque l'on peut toujours écrire une fonction pseudobooléenne quadratique sous forme d'une constante plus une posiforme quadratique. Par définition, une posiforme quadratique s'écrit $\sum_{k=1}^{K} C_k T_k$ si T_k est un littéral ou un produit de deux littéraux et si les coefficients C_k sont tous positifs. Un littéral désigne la variable x_i ou son complément $(1 - x_i)$. A partir de cette posiforme, on obtient l'expression $F(x)$. Pour notre exemple 2, l'écriture de l'objectif sous forme d'une constante, la plus grande possible, plus une posiforme quadratique est $\phi(x) = -160 + 41(1 - x_1) + 42(1 - x_2) + x_1[48(1 - x_2) + 24(1 - x_5)] + x_2[71(1 - x_5)] + x_3(40x_4 + 4x_5) + x_4[7x_5 + 85(1 - x_5)] + (1 - x_1)[4(1 - x_3) + 36(1 - x_4)] + (1 - x_2)[7x_3 + 36(1 - x_4) + 13x_5] + (1 - x_3)(1 - x_5)$.

Ensuite, une linéarisation est effectuée. Pour ce faire, pour chaque valeur de i, nous avons besoin d'une borne supérieure $\overline{f}_i$ (resp. $\overline{g}_i$) pour la fonction

$f_i(x)$ (resp. $g_i(x)$) sur X.

On peut maintenant construire un programme linéaire en variables mixtes équivalent à $(Q01)$:

$$
\begin{array}{lll}
(RL_{comp}): & \min & q_L(x) = c + L(x) + \sum_{i=1}^{n} h_i + \sum_{i=1}^{n} h'_i \\
& \text{s.c.} & \\
& & \sum_{i=1}^{n} a_{ki} x_i = b_k \qquad k = 1, \ldots, m \\
& & \sum_{i=1}^{n} a'_{\ell i} x_i \leq b'_\ell \qquad \ell = 1, \ldots, p \\
& & h_i \geq f_i(x) - \overline{f}_i(1 - x_i) \quad i = 1, \ldots, n \\
& & h'_i \geq g_i(x) - \overline{g}_i x_i \qquad i = 1, \ldots, n \\
& & x \in \{0,1\}^n \\
& & h_i \geq 0 \;\; h'_i \geq 0 \qquad i = 1, \ldots, n
\end{array}
$$

Les contraintes imposent que, pour une solution optimale (x, h, h') de (RL_{comp}), si $x_i = 0$ alors $h_i = 0$ et $h'_i = g_i(x)$; si $x_i = 1$ alors $h_i = f_i(x)$ et $h'_i = 0$. Cela garantit que la variable h_i (resp. h'_i) est égale à $x_i f_i(x)$ (resp. $(1 - x_i) g_i(x)$). Il en découle la propriété suivante.

Proposition 10 *Les deux problèmes* $(Q01)$ *et* (RL_{comp}) *sont équivalents dans le sens où, à partir d'une solution optimale de l'un, on peut construire une solution de l'autre, de même valeur.*

Dans le programme (RL_{comp}), sont présentes les n variables x_i et les $m + p$ contraintes du problème de départ $Q01$, plus $2n$ variables positives ou nulles h_i et h'_i et $2n$ contraintes de linéarisation.

D'autres linéarisations compactes existent dans la littérature, toutes inspirées de Glover [63]. On peut dire que la linéarisation compacte positive que nous proposons ici est une variante de ces linéarisations, qui présente l'avantage de travailler sur une réecriture de F sous la forme d'une constante c plus une fonction quadratique, positive ou nulle sur le domaine $\overline{X}$. Ainsi serons-nous sûrs que la borne inférieure fournie par relaxation continue de

(RL_{comp}) sera au moins égale à c, et ceci quelles que soient les valeurs valides des bornes supérieures $\overline{f}_i$ et $\overline{g}_i$ déterminées pour les fonctions $f_i(x)$ et $g_i(x)$. Observons que la qualité de cette borne dépend de la finesse des bornes supérieures $\overline{f}_i$ et $\overline{g}_i$ déterminées pour les fontions $f_i(x)$ et $g_i(x)$.

La relaxation Produit (Rappel de la section 2.2.2)
Cette relaxation est connue dans la littérature sous le nom "RLT-niveau 1" [126], [127]. Elle consiste à multiplier chaque contrainte d'égalité de ($Q01$) par x_i et chaque contrainte d'inégalité par x_i et $(1-x_i)$, à remplacer chaque produit de variable $x_i x_j$ par une nouvelle variable réelle y_{ij} puis à ajouter des contraintes de linéarisation. Enfin, c'est la relaxation continue du problème obtenu qui constitue le programme linéaire suivant :

$$
\begin{aligned}
(PL_p): \quad \min \quad & q(x,y) = \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} y_{ij} \\
\text{s.c.} \quad & \\
& \sum_{i=1}^{n} a_{ki} x_i = b_k && k = 1,..,m \\
& \sum_{i=1}^{n} a_{ki} y_{ij} = b_k x_j && k = 1,..,m; j = 1,..,n \\
& y_{ij} = y_{ji} && i = 1,..,n; j = 1,..,n : j \neq i \\
& y_{ii} = x_i && i = 1,..,n \\
& \sum_{i=1}^{n} a'_{\ell i} x_i \leq b'_\ell && \ell = 1,..,p \qquad (5.2) \\
& \sum_{i=1}^{n} a'_{\ell i} y_{ij} \leq b'_\ell x_j && \ell = 1,..,p; j = 1,..,n \qquad (5.3) \\
& \sum_{i=1}^{n} a'_{\ell i} (x_i - y_{ij}) \leq b'_\ell (1 - x_j) && \ell = 1,..,p; j = 1,..,n \qquad (5.4) \\
& y_{ij} \leq x_i && i = 1,..,n; j = 1,..,n, j \neq i \qquad (5.5) \\
& x_i + x_j - y_{ij} \leq 1 && i = 1,..,n; j = i+1,..,n \qquad (5.6) \\
& x_i \leq 1 && \qquad (5.7) \\
& x_i \geq 0 \quad y_{ij} \geq 0
\end{aligned}
$$

Il est connu que la borne fournie par la valeur optimale de (PL_p) est souvent de bonne qualité mais coûteuse à calculer compte tenu du nombre de variables et de contraintes.

Utilisation de toute solution duale de (PL_p) pour construire une reformulation linéaire compacte positive

Soit le programme linéaire suivant :

$$\begin{array}{lll} (\mathrm{P}): & \min & f(x) = \sum_{i=1}^{n} c_i x_i \\ & \text{s.c.} & \\ & & \sum_{i=1}^{n} a_{ki} x_i = b_k \qquad k = 1, \ldots, m \\ & & \sum_{i=1}^{n} a'_{\ell i} x_i \leq b'_\ell \qquad \ell = 1, \ldots, p \\ & & x_i \geq 0 \end{array}$$

et son dual :

$$\begin{array}{lll} (\mathrm{D}): & \max & g(u,v) = \sum_{k=1}^{m} (-b_k) u_k + \sum_{\ell=1}^{p} (-b'_\ell) v_\ell \\ & \text{s.c.} & \\ & & \sum_{k=1}^{m} (-a_{ki}) u_k + \sum_{\ell=1}^{p} (-a'_{\ell i}) v_\ell \leq c_i \qquad i = 1, \ldots, n \quad (5.8) \\ & & u_k \in \mathbb{R} \ , \ v_\ell \geq 0 \end{array}$$

Propriété 2 Soient $s_i, i = 1, \ldots, n$ les variables d'écart (positives) associées aux contraintes (5.8) et soit $(\tilde{u}, \tilde{v}, \tilde{s})$ une solution admissible du problème dual (D). Alors, pour toute solution admissible x du problème primal (P), on a :

$$f(x) = g(\tilde{u}, \tilde{v}) + \sum_{\ell=1}^{p} \tilde{v}_\ell (b'_\ell - \sum_{i=1}^{n} a'_{\ell i} x_i) + \sum_{i=1}^{n} \tilde{s}_i x_i$$

Preuve 13 *Il suffit d'écrire :* $c_i = \sum_{k=1}^{m} (-a_{ki}) \tilde{u}_k + \sum_{\ell=1}^{p} (-a'_{\ell i}) \tilde{v}_\ell + \tilde{s}_i$ *et d'utiliser cette égalité dans l'expression de* $f(x)$.

On obtient, pour tout x :

$$f(x) = \sum_{i=1}^{n} (\sum_{k=1}^{m} (-a_{ki}) \tilde{u}_k + \sum_{\ell=1}^{p} (-a'_{\ell i}) \tilde{v}_\ell + \tilde{s}_i) x_i$$

$$= \sum_{k=1}^{m} \tilde{u}_k (- \sum_{i=1}^{n} a_{ki} x_i) + \sum_{\ell=1}^{p} \tilde{v}_\ell (- \sum_{i=1}^{n} a'_{\ell i} x_i) + \sum_{i=1}^{n} \tilde{s}_i x_i$$

$= \sum_{k=1}^{m}(-b_k)\tilde{u}_k + \sum_{\ell=1}^{p}(-b'_\ell)\tilde{v}_\ell + \sum_{k=1}^{m}\tilde{u}_k(b_k - \sum_{i=1}^{n}a_{ki}x_i) + \sum_{\ell=1}^{p}\tilde{v}_\ell(b'_\ell - \sum_{i=1}^{n}a'_{\ell i}x_i) + \sum_{i=1}^{n}\tilde{s}_i x_i$

$= g(\tilde{u},\tilde{v}) + \sum_{k=1}^{m}\tilde{u}_k(b_k - \sum_{i=1}^{n}a_{ki}x_i) + \sum_{\ell=1}^{p}\tilde{v}_\ell(b'_\ell - \sum_{i=1}^{n}a'_{\ell i}x_i) + \sum_{i=1}^{n}\tilde{s}_i x_i$

maintenant, pour toute solution admissible x de (P), comme $b_k - \sum_{i=1}^{n}a_{ki}x_i = 0$, on obtient

$$f(x) = g(\tilde{u},\tilde{v}) + \sum_{\ell=1}^{p}\tilde{v}_\ell(b'_\ell - \sum_{i=1}^{n}a'_{\ell i}x_i) + \sum_{i=1}^{n}\tilde{s}_i x_i$$

Observons que pour toute solution admissible x, $b'_k - \sum_{i=1}^{n}a'_{\ell i}x_i \geq 0$, $x_i \geq 0$ et les coefficients $\tilde{v}_\ell$ et $\tilde{s}_i$ sont positifs. □

Grâce à la propriété 2, nous allons pouvoir montrer que les fonctions linéaires $L(x)$, $f_i(x)$ et $g_i(x)$ et l'objectif de (RL_{comp}) peuvent être construits à partir des solutions admissibles du dual de (PL_p).

Proposition 11 Pour toute solution admissible du dual de (PL_p) de valeur V, il existe $L(x)$, $f_i(x)$ et $g_i(x)$ tels que, $\forall x \in X$

$$q(x) \;=\; V + L(x) + \sum_{i=1}^{n} x_i f_i(x) + \sum_{i=1}^{n}(1 - x_i)g_i(x)$$

où $L(x)$, $f_i(x)$ et $g_i(x)$ sont des fonctions linéaires, positives, sur le domaine $\overline{X}$.

Preuve 14 *(en utilisant la propriété 2) Soient s_i (resp. s_{ij}) les variables d'écart des contraintes du problème dual de (PL_p) associées à x_i (resp. y_{ij}). Soit une solution optimale du problème (PL_p) (ou de son dual) de valeur V. Dans cette solution, on récupère $\tilde{s}_i$ et $\tilde{s}_{ij}$, les valeurs des variables s_i et s_{ij}, et les valeurs des variables duales $\tilde{v}^1$ (resp. $\tilde{v}^2$, $\tilde{v}^3$, $\tilde{v}^4$, $\tilde{v}^5$, $\tilde{v}^6$) associées aux inégalités (5.2) (resp. (5.3), (5.4), (5.5), (5.6), (5.7)). On obtient,*

Pour tout (x,y) solution de (PL_p) :
$q(x,y) = V + \sum_{i=1}^{n}\tilde{s}_i x_i + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\tilde{s}_{ij}y_{ij} + \sum_{\ell=1}^{p}\tilde{v}^1_\ell(b'_\ell - \sum_{i=1}^{n}a'_{\ell i}x_i) + \sum_{\ell=1}^{p}\sum_{j=1}^{n}\tilde{v}^2_{\ell j}(b'_\ell x_j -$

$$\sum_{i=1}^{n} a'_{\ell i} y_{ij}) + \sum_{\ell=1}^{p} \sum_{j=1}^{n} \tilde{v}^3_{\ell j}(b'_\ell(1-x_j) - \sum_{i=1}^{n} a'_{\ell i}(x_i - y_{ij})) + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \tilde{v}^4_{ij}(x_i - y_{ij}) +$$
$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \tilde{v}^5_{ij}(1 + y_{ij} - x_i - x_j) + \sum_{i=1}^{n} \tilde{v}^6_i(1 - x_i)$$

Pour tout $x \in X$, il existe un seul y tel que (x, y) solution admissible de (PL_p) et cet y est défini par $y_{ij} = x_i x_j$, ce qui donne, pour tout $x \in X$:

$$q(x) = V + \sum_{i=1}^{n} \tilde{s}_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} \tilde{s}_{ij} x_i x_j + \sum_{\ell=1}^{p} \tilde{v}^1_\ell(b'_\ell - \sum_{i=1}^{n} a'_{\ell i} x_i) + \sum_{\ell=1}^{p} \sum_{j=1}^{n} \tilde{v}^2_{\ell j} x_j (b'_\ell -$$
$$\sum_{i=1}^{n} a'_{\ell i} x_i) + \sum_{\ell=1}^{p} \sum_{j=1}^{n} \tilde{v}^3_{\ell j}(1-x_j)(b'_\ell - \sum_{i=1}^{n} a'_{\ell i} x_i) + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \tilde{v}^4_{ij} x_i (1-x_j) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \tilde{v}^5_{ij}(1-$$
$$x_i)(1 - x_j) + \sum_{i=1}^{n} \tilde{v}^6_i(1 - x_i)$$

En posant :

$$L(x) = \sum_{i=1}^{n} \tilde{s}_i x_i + \sum_{i=1}^{n} \tilde{v}^6_i(1 - x_i) + \sum_{\ell=1}^{p} \tilde{v}^1_\ell(b'_\ell - \sum_{i=1}^{n} a'_{\ell i} x_i)$$

$$f_i(x) = \sum_{j=1}^{n} \tilde{s}_{ij} x_j + \sum_{j=1, j\neq i}^{n} \tilde{v}^4_{ij}(1 - x_j) + \sum_{\ell=1}^{p} \tilde{v}^2_{\ell i}(b'_\ell - \sum_{j=1}^{n} a'_{\ell j} x_j)$$

$$g_i(x) = \sum_{j=i+1}^{n} \tilde{v}^5_{ij}(1 - x_j) + \sum_{\ell=1}^{p} \tilde{v}^3_{\ell i}(b'_\ell - \sum_{j=1}^{n} a'_{\ell j} x_j)$$

et en observant que pour tout $x \in \overline{X}$, les fonctions ci-dessus sont positives ou nulles, on obtient une écriture de $F(x)$ sous la forme (5.1), soit :

$$\forall x \in X, \quad q(x) = V + L(x) + \sum_{i=1}^{n} x_i f_i(x) + \sum_{i=1}^{n} (1 - x_i) g_i(x)$$

□

Finalement, on détermine des bornes supérieures $\overline{f}_i = \sum_{j=1}^{n} \tilde{s}_{ij} + \sum_{j=1, j\neq i}^{n} \tilde{v}^4_{ij} + \sum_{\ell=1}^{p} \tilde{v}^2_{\ell i}(b'_\ell - \sum_{j=1: a'_{\ell j}<0}^{n} a'_{\ell j})$ et $\overline{g}_i = \sum_{j=i+1}^{n} \tilde{v}^5_{ij} + \sum_{\ell=1}^{p} \tilde{v}^3_{\ell i}(b'_\ell - \sum_{j=1: a'_{\ell j}<0}^{n} a'_{\ell j})$ pour obtenir la reformulation linéaire compacte associée à la relaxation (PL_p).

La proposition 11 est vraie en particulier pour la solution optimale de (PL_p). On peut alors réécrire $F(x) = V^* + L^*(x) + \sum_{i=1}^{n} x_i f_i^*(x) + \sum_{i=1}^{n}(1 - x_i)g_i^*(x)$ afin de contruire la linéarisation compacte positive associée :

$$
\begin{aligned}
(RL^*_{comp}): \quad \min \quad & q_L(x) = V^* + L^*(x) + \sum_{i=1}^{n} h_i + \sum_{i=1}^{n} h'_i \\
\text{s.c.} \quad & \\
& \sum_{i=1}^{n} a_{ki} x_i = b_k \qquad k = 1, \ldots, m \\
& \sum_{i=1}^{n} a'_{\ell i} x_i \leq b'_\ell \qquad \ell = 1, \ldots, p \\
& h_i \geq f_i^*(x) - \overline{f}_i^*(1 - x_i) \qquad i = 1, \ldots, n \\
& h'_i \geq g_i^*(x) - \overline{g}_i^* x_i \qquad i = 1, \ldots, n \\
& x \in \{0,1\}^n \\
& h_i \geq 0 \quad h'_i \geq 0 \quad i = 1, \ldots, n
\end{aligned}
$$

et montrer que la valeur optimale de $(\overline{RL^*_{comp}})$ est au moins aussi bonne que celle de (PL_p). La proposition suivante le montre.

Proposition 12 Soit (x^*, y^*) la solution optimale de (PL_p) associée à la solution duale qui a permis de construire (RL^*_{comp}). Alors $(x^*, 0, 0)$ est une solution admissible de $(\overline{RL^*_{comp}})$, de valeur V^*. Il s'agit donc d'une solution optimale de $(\overline{RL^*_{comp}})$.

Preuve 15 *Il suffit de prouver que,*

1. $L^*(x^*) = 0$ *et*
2. *pour tout* i, $0 \geq f_i^*(x^*) - \overline{f^*}_i(1 - x_i^*)$
3. *pour tout* i, $0 \geq g_i^*(x^*) - \overline{g^*}_i x_i^*$.

1. $L^*(x^*) = 0$ *découle des relations des écarts complémentaires.*

2. *Pour* $i = 1, ...n$,
   $f_i^*(x^*) - \overline{f^*}_i(1 - x_i^*) =$
   $\sum_{j=1}^{n} \tilde{s}_{ij} x_j^* + \sum_{j=1, j\neq i}^{n} \tilde{v}_{ij}^4 (1 - x_j^*) + \sum_{\ell=1}^{p} \tilde{v}_{\ell i}^2 (b'_\ell - \sum_{j=1}^{n} a'_{\ell j} x_j^*)$

$$- \left(\sum_{j=1}^{n} \tilde{s}_{ij} + \sum_{j=1,j\neq i}^{n} \tilde{v}^4_{ij} + \sum_{\ell=1}^{p} \tilde{v}^2_{\ell i}(b'_\ell - \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j}) \right)(1-x^*_i) =$$
$$\sum_{j=1}^{n} \tilde{s}_{ij}(x^*_j + x^*_i - 1) + \sum_{j=1,j\neq i}^{n} \tilde{v}^4_{ij}(x^*_i - x^*_j)$$
$$+ \sum_{\ell=1}^{p} \tilde{v}^2_{\ell i} \left(b'_\ell x^*_i - \sum_{j=1}^{n} a'_{\ell j} x^*_j + \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j}(1 - x^*_i) \right)$$

Chacune des composantes de la somme ci-dessus est négative ou nulle. En effet,

- $\tilde{s}_{ij}(x^*_j + x^*_i - 1) \leq 0$ *car si* $\tilde{s}_{ij} > 0$ *alors* $y^*_{ij} = 0$ *et vérifierait* $y^*_{ij} \geq x^*_j + x^*_i - 1$
- $\tilde{v}^4_{ij}(x^*_i - x^*_j) \leq 0$ *car si* $\tilde{v}^4_{ij} > 0$ *alors* $y^*_{ij} = x^*_i$ *ainsi,* $x^*_i - x^*_j = y^*_{ij} - x^*_j \leq 0$
- $\tilde{v}^2_{\ell i}(b'_\ell x^*_i - \sum_{j=1}^{n} a'_{\ell j} x^*_j) \leq 0$ *car si* $\tilde{v}^2_{\ell i} > 0$ *alors* $\sum_{j=1}^{n} a'_{\ell j} y^*_{ji} = b'_\ell x^*_i$ *ainsi,*

  $b'_\ell x^*_i - \sum_{j=1}^{n} a'_{\ell j} x^*_j + \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j}(1 - x^*_i)$

  $= \sum_{j=1}^{n} a'_{\ell j} y^*_{ji} - \sum_{j=1}^{n} a'_{\ell j} x^*_j + \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j}(1 - x^*_i)$

  $= \sum_{j=1:a'_{\ell j}>0}^{n} a'_{\ell j}(y^*_{ji} - x^*_j) + \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j}(y^*_{ji} - x^*_j + 1 - x^*_i) \leq 0$ *puisque* $y^*_{ji} - x^*_j \leq 0$ *et* $y^*_{ji} - x^*_j + 1 - x^*_i \geq 0$

3. *Pour* $i = 1, ...n$,

   $g^*_i(x^*) - \overline{g^*}_i x^*_i =$

   $\sum_{j=i+1}^{n} \tilde{v}^5_{ij}(1 - x^*_j) + \sum_{\ell=1}^{p} \tilde{v}^3_{\ell i}(b'_\ell - \sum_{j=1}^{n} a'_{\ell j} x^*_j) - \left(\sum_{j=i+1}^{n} \tilde{v}^5_{ij} + \sum_{\ell=1}^{p} \tilde{v}^3_{\ell i}(b'_\ell - \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j}) \right) x^*_i$

   $= \sum_{j=i+1}^{n} \tilde{v}^5_{ij}(1 - x^*_j - x^*_i) + \sum_{\ell=1}^{p} \tilde{v}^3_{\ell i} \left(b'_\ell(1 - x^*_i) - \sum_{j=1}^{n} a'_{\ell j} x^*_j + \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j} x^*_i \right)$

   $= \sum_{j=i+1}^{n} \tilde{v}^5_{ij}(1 - x^*_j - x^*_i) + \sum_{\ell=1}^{p} \tilde{v}^3_{\ell i} \left(b'_\ell(1 - x^*_i) - \sum_{j=1}^{n} a'_{\ell j} x^*_j + \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j} x^*_i \right)$

   Ici encore, chacune des composantes de la somme ci-dessus est négative ou nulle. En effet,

   - $\tilde{v}^5_{ij}(1 - x^*_j - x^*_i) \leq 0$ *car si* $\tilde{v}^5_{ij} > 0$ *alors* $1 - x^*_j - x^*_i = -y^*_{ij} \leq 0$
   - $\tilde{v}^3_{\ell i} \left(b'_\ell(1 - x^*_i) - \sum_{j=1}^{n} a'_{\ell j} x^*_j + \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j} x^*_i \right) \leq 0$ *car si* $\tilde{v}^3_{\ell i} > 0$

$$
\begin{aligned}
&\textit{alors } b'_\ell(1-x_i^*) = \sum_{j=1}^{n} a'_{\ell j}(x_j^* - y_{ji}^*) \textit{ ainsi,}\\
&b'_\ell(1-x_i^*) - \sum_{j=1}^{n} a'_{\ell j}x_j^* + \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j}x_i^*\\
&= \sum_{j=1}^{n} a'_{\ell j}(x_j^* - y_{ji}^*) - \sum_{j=1}^{n} a'_{\ell j}x_j^* + \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j}x_i^*\\
&= \sum_{j=1:a'_{\ell j}>0}^{n} a'_{\ell j}(-y_{ji}^*) + \sum_{j=1:a'_{\ell j}<0}^{n} a'_{\ell j}(x_i^* - y_{ji}^*) \leq 0\\
&\textit{puisque } (-y_{ji}^*) \leq 0 \textit{ et } (x_i^* - y_{ji}^*) \geq 0
\end{aligned}
$$

□

Corollaire 6 Pour toute solution admissible du dual de (PL_p) de valeur V, on peut construire une reformulation linéaire compacte positive dont la valeur de la relaxation continue est supérieure à V.

Corollaire 7 Pour toute solution optimale du dual de (PL_p) de valeur V^*, on peut construire une reformulation linéaire compacte positive dont la valeur de la relaxation continue est supérieure à V^*.

Exemple 2 :

Réécrivons la fonction objectif $\phi(x)$ de (E) en fonction des variables x, après résolution de la relaxation produit :

$$
\begin{aligned}
\phi(x) = \quad & -67.52\\
& +x_1\left(109.83x_4 + 30.07x_5\right)\\
& +x_2\left(129.90x_4 + 3.52\left(1-x_3\right)\right)\\
& +x_3\left(1.26\left[-2-\left(-x_1+2x_2-5x_3-2x_4+2x_5\right)\right]\right)\\
& +x_4\left(6.95\left[-2-\left(-x_1+2x_2-5x_3-2x_4+2x_5\right)\right]\right)\\
& +x_5\left(52.34\left(1-x_3\right)+11.22\left[-2-\left(-x_1+2x_2-5x_3-2x_4+2x_5\right)\right]\right)\\
& +\left(1-x_2\right)\left(0.74\left[-2-\left(-x_1+2x_2-5x_3-2x_4+2x_5\right)\right]\right)
\end{aligned}
$$

On peut donc maintenant définir les fonctions $f_i(x)$ et $g_i(x)$,

$$\begin{aligned}
f_1(x) = &\quad 109.83x_4 + 30.07x_5 \\
f_2(x) = &\quad 129.90x_4 + 3.52(1 - x_3) \\
f_3(x) = &\quad 1.26(-2 - (-x_1 + 2x_2 - 5x_3 - 2x_4 + 2x_5)) \\
f_4(x) = &\quad 6.95(-2 - (-x_1 + 2x_2 - 5x_3 - 2x_4 + 2x_5)) \\
f_5(x) = &\quad 52.34(1 - x_3) + 11.22(-2 - (-x_1 + 2x_2 - 5x_3 - 2x_4 + 2x_5)) \\
g_2(x) = &\quad 0.74(-2 - (-x_1 + 2x_2 - 5x_3 - 2x_4 + 2x_5))
\end{aligned}$$

en déduire leurs bornes supérieures,

$$\begin{aligned}
\overline{f}_1 = &\quad 109.83 + 30.07 = 139.90 \\
\overline{f}_2 = &\quad 129.90 + 3.52 = 133.42 \\
\overline{f}_3 = &\quad 1.26(-2 - (-1 - 5 - 2)) = 7.56 \\
\overline{f}_4 = &\quad 6.95(-2 - (-1 - 5 - 2)) = 41.7 \\
\overline{f}_5 = &\quad 52.34 + 11.22(-2 - (-1 - 5 - 2)) = 67.32 \\
\overline{g}_2 = &\quad 0.74(-2 - (-1 - 5 - 2)) = 4.44
\end{aligned}$$

et enfin, construire le programme (RL_{comp_E}).

$$\begin{aligned}
(RL_{comp_E}) \quad \min \quad & \phi(x) = -67.52 + \sum_{i=1}^{5} h_i + h'_2 \\
\text{s.c.} \quad & \\
& -x_1 + 2x_2 - 5x_3 - 2x_4 + 2x_5 \leq -2 \\
& x_1 + x_2 + x_4 + x_5 = 2 \\
& h_i \geq f_i(x) - \overline{f}_i(1 - x_i) \qquad i = 1, \ldots, 5 \\
& h'_2 \geq g_2(x) - \overline{g}_2 x_2 \\
& x \in \{0, 1\}^n \\
& h_i \geq 0 \qquad\qquad\qquad\qquad i = 1, \ldots, 5 \\
& h'_2 \geq 0
\end{aligned}$$

La valeur optimale de la relaxation continue associée à la linéarisation compacte (RL_{comp_E}) est -67.52. ♢

Un exemple de la reformulation linéaire compacte de Glover appliquée directement au problème inital (E), sans utilisation de la solution optimale

de (PL_p), est donné à la fin de ce chapitre : sa relaxation continue (valeur optimale = -110.78) est moins bonne que celle associée à la linéarisation compacte positive.

5.2 Mise en parrallèle avec QCR : utilisation de la borne trouvée par une bonne relaxation SDP

Soit une relaxation SDP $(SDQ01)$ du programme $(Q01)$, déjà définie en section 3.2 :

$$
\begin{aligned}
(SDQ01): \quad \min \quad & c^t x + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} q_{ij} X_{ij} \\
\text{s.c.} \quad & \\
& X_{ii} = x_i \qquad\qquad i = 1, .., n \qquad (5.9) \\
& -b_k x_i + \sum_{j=1}^{n} a_{kj} X_{ij} = 0 \quad i = 1, .., n; k = 1, .., m (5.10) \\
& Ax = b \\
& A'x \leq b' \\
& \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\
& x \in \mathbb{R}^n, \; X \in S_n
\end{aligned}
$$

Utilisation de la solution optimale de SDP pour construire une reformulation quadratique

On rappelle que la méthode QCR consiste à reformuler le problème $(Q01)$ en ajoutant à la fonction objectif $q(x)$ deux fonctions nulles sur tout le domaine X et dépendant de deux paramètres $\alpha \in \mathbb{R}^{m \times n}$ et $u \in \mathbb{R}^n$:

$$
q_{\alpha,u}(x) = \sum_{i=1}^{n} q_i x_i + \sum_{i=1}^{n-1} \sum_{j=i, j\neq i}^{n} q_{ij} x_i x_j + \sum_{k=1}^{m} \left(\sum_{i=1}^{n} \alpha_{ki} x_i \right) \left(\sum_{j=1}^{n} a_{kj} x_j - b_k \right) + \sum_{i=1}^{n} u_i \left(x_i^2 - x_i \right)
$$

Il a été montré dans [19] et dans le chapitre 3 que trouver les valeurs optimales des paramètres α et u (notées respectivement α^* et u^*) peut se faire par la résolution de la relaxation semidéfinie $(SDQ01)$ ci-dessus.

La reformulation quadratique convexe associée est donc :

$$
\begin{array}{lll}
(RQ_{conv}) : & \min & q_{\alpha^*,u^*}(x) \\
& \text{s.c.} & \\
& & \displaystyle\sum_{i=1}^{n} a_{ki}x_i = b_k \quad k = 1,\ldots,m \\
& & \displaystyle\sum_{i=1}^{n} a'_{\ell i}x_i \leq b'_\ell \quad \ell = 1,\ldots,p \\
& & x \in \{0,1\}^n
\end{array}
$$

On sait également que la valeur optimale du programme $(SDQ01)$ est égale à la valeur optimale de la relaxation continue de (RQ_{conv}).

On a ainsi réécrit $q(x)$ comme une constante à laquelle on ajoute une fonction quadratique convexe positive ou nulle pour tout $x \in \overline{X}$. En effet, en posant $g(x) = q_{\alpha^*,u^*}(x) - v\,(SDQ01)$, on a l'égalité $q(x) = v\,(SDQ01) + g(x)$.

On peut donc construire, tout comme pour la reformulation linéaire compacte positive présentée dans les sections précédentes, une reformulation quadratique convexe grâce à la résolution d'une relaxation SDP : QCR "capture" la borne trouvée par $(SDQ01)$.

Exemple 2 :
La relaxation SDP de (E) est :

$$
\begin{array}{lll}
(SDP_E): & \min & -9x_1 - 7x_2 + 2x_3 + 23x_4 + 12x_5 - 48X_{12} + 4X_{13} + 36X_{14} \\
 & & -24X_{15} - 7X_{23} + 36X_{24} - 84X_{25} + 40X_{34} + 4X_{35} - 88X_{45} \\
 & \text{s.c.} & \\
 & & x_1 - 2x_2 + 5x_3 + 2x_4 - 2x_5 \geq 2 \\
 & & x_1 + x_2 + x_4 + x_5 = 2 \\
 & & X_{12} + X_{14} + X_{15} = x_1 \qquad \leftarrow \alpha_1^* \\
 & & X_{12} + X_{24} + X_{25} = x_2 \qquad \leftarrow \alpha_2^* \\
 & & X_{13} + X_{23} + X_{34} + X_{35} = 2x_3 \qquad \leftarrow \alpha_3^* \\
 & & X_{14} + X_{24} + X_{45} = x_4 \qquad \leftarrow \alpha_4^* \\
 & & X_{15} + X_{25} + X_{45} = x_5 \qquad \leftarrow \alpha_5^* \\
 & & X_{11} = x_1 \qquad \leftarrow u_1^* \\
 & & X_{22} = x_2 \qquad \leftarrow u_2^* \\
 & & X_{33} = x_3 \qquad \leftarrow u_3^* \\
 & & X_{44} = x_4 \qquad \leftarrow u_4^* \\
 & & X_{55} = x_5 \qquad \leftarrow u_5^* \\
 & & \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\
 & & x \in \mathbb{R}^n,\ X \in \mathbb{R}^{n \times n}
\end{array}
$$

La valeur optimale de (SDP_E) est égale à -81.32, également valeur optimale de la relaxation continue de (RQ_{conv_E}) où les paramètres u^* et α^* sont calculés à partir de la solution optimale de (SDP_E) :

$$
\begin{array}{lll}
(RQ_{conv_E}): & \min & \phi(x) + (8066.79x_1 + 8076.64x_2 - 8.79x_3 + 8040.33x_4 + 8088.93x_5) \\
 & & (x_1 + x_2 + x_4 + x_5 - 2) + 8.63(x_1^2 - x_1) + 0.17(x_2^2 - x_2) \\
 & & +12.44(x_3^2 - x_3) + 66.36(x_4^2 - x_4) - 7.70(x_5^2 - x_5) \\
 & \text{s.c.} & \\
 & & x_1 - 2x_2 + 5x_3 + 2x_4 - 2x_5 \geq 2 \\
 & & x_1 + x_2 + x_4 + x_5 = 2 \\
 & & x \in \{0,1\}^n
\end{array}
$$

♢

La reformulation linéaire compacte positive et la reformulation QCR utilisent toutes deux la solution d'une relaxation particulière (produit pour la

reformulation linéaire et SDP pour QCR). Ainsi, on a :

$$v(\overline{RL_{comp}}) = v(PL_p)$$

$$v(\overline{RL_{conv}}) = v(SDP)$$

Remarque : complément sur l'Exemple 2

Soit le programme quadratique en variables 0-1 (E) de l'exemple 2. Appliquons la linéarisation classique *:*

$$\begin{array}{lll}
(RL_E): & \min & -9x_1 - 7x_2 + 2x_3 + 23x_4 + 12x_5 - 48y_{12} + 4y_{13} + 36y_{14} \\
 & & -24y_{15} - 7y_{23} + 36y_{24} - 84y_{25} + 40y_{34} + 4y_{35} - 88y_{45} \\
 & s.c. & \\
 & & -x_1 + 2x_2 - 5x_3 - 2x_4 + 2x_5 \leq -2 \\
 & & x_1 + x_2 + x_4 + x_5 = 2 \\
 & & y_{ij} \leq x_i \qquad\qquad i < j; c_{ij} < 0 \\
 & & y_{ij} \leq x_j \qquad\qquad i < j; c_{ij} < 0 \\
 & & 1 - x_i - x_j + y_{ij} \geq 0 \quad i < j; c_{ij} < 0 \\
 & & y_{ij} \geq 0 \qquad\qquad i < j; c_{ij} < 0 \\
 & & x_1, \ldots, x_5 \in \{0, 1\}
\end{array}$$

La valeur optimale associée à la relaxation de la linéarisation classique est -115.

Si on applique directement la linéarisation de Glover à (E), sans passer

par la reformulation produit, on obtient le programme suivant :

$$
\begin{array}{lll}
(RL_{comp_dE}): & \min & -9x_1 - 7x_2 + 2x_3 + 23x_4 + 12x_5 + z_1 + z_2 + z_3 + z_4 + z_5 \\
& s.c. & \\
& & -x_1 + 2x_2 - 5x_3 - 2x_4 + 2x_5 \leq -2 \\
& & x_1 + x_2 + x_4 + x_5 = 2 \\
& & z_1 \geq -30x_1 \\
& & z_1 \geq -24x_2 + 2x_3 + 18x_4 - 12x_5 - 20(1 - x_1) \\
& & z_2 \geq -69.5x_2 \\
& & z_2 \geq -24x_1 - 3.5x_3 + 18x_4 - 42x_5 - 16.6(1 - x_2) \\
& & z_3 \geq -1.5x_3 \\
& & z_3 \geq 2x_1 - 3.5x_2 + 20x_4 + 2x_5 - 22(1 - x_3) \\
& & z_4 \geq -36x_4 \\
& & z_4 \geq 18x_1 + 18x_2 + 20x_3 - 44x_5 - 56(1 - x_4) \\
& & z_5 \geq -85.2x_5 \\
& & z_5 \geq -12x_1 - 42x_2 + 2x_3 - 44x_4 + 10(1 - x_5) \\
& & x_1, \ldots, x_5 \in \{0, 1\}
\end{array}
$$

dont la valeur optimale de la relaxation continue est égale à -110.78.

En résumé, voici le tableau récapitulatif des valeurs optimales des relaxations des différentes reformulations de (E) :

Valeur opt. (E)	Relax.lin. classique (RL_E)	Relax.lin.compacte directe (RL_{comp_dE})	Relax.lin. produit (PL_{p_E})	Relax.lin.compacte après RLT (RL_{comp_E})	Relax.reformulation. quad. convexe (RQ_{conv_E})
-65	-115	-110.78	-67.52	-67.52	-81.39

$\diamondsuit$

Dans ce chapitre, nous avons proposé une technique de reformulation linéaire, que nous avons appelée *reformulation linéaire compacte positive*.

La reformulation linéaire développée ici incorpore la valeur d'une bonne relaxation linéaire (la relaxation produit) dans un schéma de branch-and-bound. Parallèlement, QCR intègre, quant à elle, une borne obtenue par relaxation SDP.

Des résultats expérimentaux sont donnés dans [2] par exemple pour la reformulation linéaire et dans les chapitres suivants pour la reformulation quadratique convexe. Plus particulièrement, le chapitre 6 présente trois applications numériques de la méthode QCR sur des problèmes non convexes.

Chapitre 6

Etude expérimentale de trois problèmes d'optimisation combinatoire avec un objectif non convexe

Ce chapitre présente des applications de la méthode QCR à trois problèmes d'optimisation combinatoire non convexe : le problème du k-cluster (section 6.2), de la bipartition de graphe (section 6.3) et de la minimisation d'échange d'outils (section 6.4). Les méthodes de la plus petite valeur propre, EQCR et IQCR vont également être appliquées au problème du k-cluster (sections 6.2.1 et 6.2.2) de manière à donner, en plus des résultats théoriques de la section 4.4, de nombreux résultats expérimentaux. Avant de définir chaque problème et de réaliser leur résolution exacte, la section 6.1 présente l'environnement expérimental dans lequel nous avons effectué les tests.

6.1 Environnement expérimental

Toutes les expérimentations présentées dans ce manuscrit ont été réalisées sur un Pentium IV 2.2 GHz avec 1 Go de RAM. Nous avons également utilisé les logiciels suivants :

- *Scilab* pour le calcul de valeurs propres.
- *SB* [77] ou encore *CSDP* [28] pour la résolution des programmes SDP.

– La version 9 de *CPLEX* [41] avec le modeleur *AMPL* pour la résolution des programmes quadratiques convexes en variables 0-1.

6.2 Le problème du k-cluster

Soit un graphe non orienté $G = (V, U)$ de n sommets $\{v_1, \ldots, v_n\}$, un ensemble d'arêtes non pondérées U et un entier positif $k \in \{3, \ldots, n-2\}$. Le problème du k-cluster consiste à sélectionner un sous-ensemble de k sommets $S \subseteq V$ tel que le sous-graphe induit par S maximise le nombre d'arêtes.

Par exemple, considérons le graphe $G = (V, U)$ suivant à 7 sommets :

FIG. 6.1 – Problème du k-cluster : $n = 7$

Pour $k = 4$, le problème du k-cluster consiste à sélectionner un sous-ensemble S de 4 sommets qui maximise le nombre d'arêtes contenues dans S. Une solution optimale est donnée par le sous-ensemble $S = \{v_3, v_4, v_6, v_7\}$ de 4 sommets et la valeur optimale associée est égale à 4 (autrement dit les arêtes $[v_3, v_4], [v_3, v_7], [v_4, v_6]$ et $[v_6, v_7]$).

Le problème du k-cluster ou encore du k-sous-graphe le plus dense ([15], [93], [117]) peut être facilement modélisé par un problème quadratique en variables 0-1 sous contraintes linéaires :

$$(KC') \quad : \quad \max \left\{ f'(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \delta_{ij} x_i x_j \ : \ \sum_{i=1}^{n} x_i = k, \ x \in \{0,1\}^n \right\}$$

où les coefficients binaires $\delta_{ij} = 1$ si et seulement si $[v_i, v_j]$ est une arête de G. La variable binaire x_i est égale à 1 si et seulement si le sommet v_i

FIG. 6.2 – Problème du k-cluster : $n = 7$ et $k = 3$

est dans le sous-ensemble S. Le programme (KC') ne contient qu'une seule contrainte d'égalité qui est une contrainte de cardinalité : on veut choisir les k sommets qui maximisent la fonction objectif.

Le problème du k-cluster peut aussi s'écrire comme un problème de minimisation :

$$(KC) \quad : \quad \min \left\{ f(x) = x^t \Delta x \ : \ \sum_{i=1}^{n} x_i = k, \ x \in \{0,1\}^n \right\}$$

où le terme général de la matrice Δ de dimension $n \times n$ est $\Delta_{ij} = -\frac{1}{2}\delta_{ij}, \ \forall i, j$. On notera que $v(KC) \ = \ -v(KC')$.

Ce problème a été étudié par Billionnet [15] qui propose six réécritures du problème (KC'). L'efficacité de ces différentes formulations dépend fortement des instances considérées. Par exemple, elles ne permettent pas de résoudre (KC') en moins de trente minutes pour des graphes à plus de 68 sommets quand la densité est inférieure ou égale à 50% et à plus de 116 sommets quand elle est égale à 75%. Pisinger [117] présente également une reformulation du problème de maximisation (KC'), mais cette fois-ci avec poids sur les arêtes.

Les sections 6.2.1 et 6.2.2 présentent nos expérimentations, autrement dit l'application de la méthode de la plus petite valeur propre, EQCR, IQCR et enfin QCR à (KC).

6.2.1 Application de la méthode de la plus petite valeur propre et de EQCR

La méthode EQCR est la première méthode de reformulation que nous avons élaborée. Elle est bien sûr moins efficace que la méthode QCR en terme de calcul de bornes, comme nous l'avons démontré dans la section 4.4. Cependant, nous allons présenter une application de la méthode EQCR dans le but de la comparer expérimentalement avec la méthode de la plus petite valeur propre (section 4.1) et d'étudier, pour le problème du k-cluster, l'effet apporté par l'ajout de la fonction nulle paramétrée $\beta\left(\sum_{i=1}^{n} x_i - k\right)^2$ à $f(x)$. Dans le cas de la méthode de la plus petite valeur propre, on rappelle que $\beta = 0$.

La reformulation EQCR

Reprenons le problème (KC). D'après la section 4.2, nous pouvons définir un nouveau problème, qui dépend d'un nouveau paramètre β :

$$(KC_\beta) \ : \ \min\left\{ f_\beta(x) \ : \ \sum_{i=1}^{n} x_i = k, \ x \in \{0,1\}^n \right\}$$

où

$$\begin{aligned} f_\beta(x) &= x^t \Delta x + \beta\left(\sum_{i=1}^{n} x_i - k\right)^2 - \lambda_{min}(\Delta_\beta)\sum_{i=1}^{n}(x_i^2 - x_i) \\ &= x^t \Delta_\beta x - \beta k^2 - \lambda_{min}(\Delta_\beta)\sum_{i=1}^{n}(x_i^2 - x_i) \end{aligned}$$

et où le terme général de Δ_β est $(\Delta_\beta)_{ij} = -\frac{1}{2}\delta_{ij} + \beta, \ \forall i, j$.

La méthode EQCR consiste tout d'abord à résoudre le programme semi-

défini suivant (*Phase I*) :

$$\begin{aligned}(SDKC_1) \ : \ \min \ & \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\delta_{ij}X_{ij} \\ \text{s.c.} \ & tr(X) \ = \ 1 \\ & \sum_{i=1}^{n}\sum_{j=1}^{n}X_{ij} \ = \ 0 \qquad (6.1) \\ & X \succeq 0\end{aligned}$$

La valeur optimale β^* est la variable duale optimale associée à la contrainte (6.1). Ainsi, le programme (KC_{β^*}) suivant est convexe :

$$(KC_{\beta^*}) \ : \ \min_{x\in X}\left\{x^t\Delta x + \beta^*\left(\sum_{i=1}^{n}x_i - k\right)^2 + \lambda_{min}(\Delta_{\beta^*})\sum_{i=1}^{n}\left(x_i^2 - x_i\right)\right\}$$

La reformulation de la plus petite valeur propre

La méthode de la plus petite valeur propre consiste, quant à elle, à reformuler (KC) en un nouveau programme quadratique convexe (KC_0) :

$$(KC_0) \ : \ \min_{x\in X}\left\{x^t\Delta x + \lambda_{min}(\Delta)\sum_{i=1}^{n}\left(x_i^2 - x_i\right)\right\}$$

Génération de graphes

La méthode de la plus petite valeur propre et EQCR sont appliquées sur des graphes générés aléatoirement : pour une densité donnée d et pour toute paire d'indices (i, j) tel que $i < j$, un nombre aléatoire ρ est généré dans l'intervalle $[0, 1]$. Si $\rho > d$ alors δ_{ij} est fixé à 0, sinon δ_{ij} est fixé à 1.

Résultats

Les tableaux suivants présentent la comparaison des deux algorithmes de

résolution (notés respectivement HR et EQCR pour la méthode d'Hammer et Rubin et la notre) pour différentes tailles de graphe ($n = 40, 80$), différentes densités ($d = 25\%, 50\%, 75\%$) et différentes valeurs de k ($k = \frac{n}{4}, \frac{n}{2}, \frac{3n}{4}$).

Légende des tableaux 6.1 à 6.6 :

- d, densité du graphe
- *opt*, valeur de la solution optimale ou de la meilleure solution connue
- CPU, temps de calcul requis par le logiciel $CPLEX9$. Le signe '-' indique que l'instance correspondante n'a pu être résolue à l'optimum en moins d'une heure
- *borne* représente la valeur optimale de la relaxation continue de $\left(\overline{KC_0}\right)$ (resp. $\left(\overline{KC_{\beta^*}}\right)$) si l'on applique la méthode de la plus petite valeur propre (resp. EQCR). *borne* n'est autre que la borne inférieure obtenue par relaxation continue à la racine de l'arborescence de recherche.
- β^*, paramètre β optimal, déjà défini dans la section 4.2
- gap $(HR) = \frac{\gamma(0) - opt}{opt} * 100$
  gap $(EQCR) = \frac{\gamma(\beta^*) - opt}{opt} * 100$, les écarts relatifs entre la meilleure solution connue et la borne inférieure calculée à la racine de l'arbre de recherche (saut d'intégrité)
- **moy.**, moyennes, pour chaque triplet (n, k, d), des temps CPU et des sauts d'intégrité des deux approches.

TAB. 6.1 – Résultats des 2 approches ($n = 40, k = \frac{n}{4}$)

		HR			*EQCR*			
d(%)	*opt*	CPU	*borne*	gap(%)	CPU	*borne*	β^*	gap(%)
25	29	1'1"	51.02	75.9	0.07"	32.30	9.46	11.3
	30	4'17"	56.99	90	0.24"	34.89	20.57	16.3
	27	11'16"	53.70	98.8	1.13"	32.58	1.15	20.6
	27	2'35"	50.02	85.2	0.47"	31.609	9.95	17.07
	26	2'55"	48.60	86.9	0.84"	31.69	8.03	21.8
	moy.	**4'25"**		**87.4**	**0.55"**			**17.4**
50	40	-	97.73	144	0.69"	46.46	19.52	16
	41	-	106.39	159	1.16"	47.13	44.34	14.9
	39	-	104.22	167	4.17"	46.55	0.48	19.3
	40	-	100.76	151.9	0.84"	46.05	6.94	15
	40	-	98.52	146.3	0.65"	46.15	0.63	15.3
	moy.	**-**		**153.6**	**1.5"**			**16.1**
75	45	-	145.280	222	28.66"	54.14	44.34	20.3
	45	-	147.38	227.5	1'3"	54.99	2.98	22.2
	45	-	148.34	229.7	44.27"	53.69	6.63	19.3
	45	-	147.22	227	46.1"	54.425	8.93	20.9
	45	-	146.23	224	58.11"	54.91	3.42	22
	moy.	**-**		**226**	**47.94"**			**20.9**

- : instance correspondante non résolue en moins d'1h

TAB. 6.2 – Résultats des 2 approches ($n = 40, k = \frac{n}{2}$)

		HR			*EQCR*			
d(%)	*opt*	CPU	*borne*	gap(%)	CPU	*borne*	β^*	gap (%)
25	77	3'23"	102.04	32.5	0.29"	81.59	9.46	6
	84	14'7"	113.99	35.7	0.21"	88.60	20.57	5.5
	76	59'11"	107.42	41.3	1.17"	81.74	1.15	7.6
	75	4'41"	100.06	33.4	0.19"	79.45	9.95	5.9
	73	4'33"	97.21	33.2	0.32"	77.92	8.03	6.7
	moy.	**17'11"**		**35.2**	**0.44"**			**6.3**
50	130	-	195.459	50	0.08"	134.42	19.52	3.4
	130	-	212.792	63.7	3.59"	137.33	44.34	5.6
	133	-	208.442	56.7	0.26"	137.92	0.48	3.7
	128	-	201.54	57.5	0.75"	133.943	6.94	4.6
	129	-	197.048	52.7	0.25"	134.249	0.63	4.1
	moy.	**-**		**56**	**0.99"**			**4.3**
75	168	-	290.56	73	0.55"	173.26	44.34	3.1
	173	-	294.77	70.4	0.16"	177.425	2.98	2.6
	171	-	296.69	73.5	0.48"	175.764	6.63	2.8
	171	-	294.45	72.2	0.33"	175.805	8.93	2.8
	171	-	292.475	71	0.88"	176.982	3.42	3.5
	moy.	**-**		**72**	**0.48"**			**2.96**

- : instance correspondante non résolue en moins d'1h

TAB. 6.3 – Résultats des 2 approches ($n = 40, k = \frac{3n}{4}$)

		HR			*EQCR*			
d(%)	*opt*	CPU	*borne*	gap(%)	CPU	*borne*	β^*	gap (%)
25	134	3.76"	151.91	13.4	0.06"	137.35	9.46	2.5
	148	19.36"	170.91	15.5	0.16"	152.95	20.57	3.3
	138	43.62"	161.01	16.7	0.3"	143.43	1.15	3.9
	134	2.11"	149.98	11.9	0.06"	137.784	9.95	2.8
	130	1.9"	145.26	11.3	0.05"	133.46	8.03	2.6
	moy.	**14.15"**		**13.76**	**0.12"**			**3**
50	247	3'29"	293.189	18.7	0.06"	250.95	19.52	1.6
	263	58'18"	319.18	21.3	0.07"	266.59	44.34	1.36
	261	9'6"	312.66	19.8	0.07"	265.0	0.48	1.5
	254	5'4"	302.30	19	0.11"	258.75	6.94	1.87
	250	2'30"	295.53	18.2	0.04"	254.20	0.63	1.6
	moy.	**10'2"**		**19.4**	**0.07"**			**1.58**
75	349	-	435.84	24.8	0.05"	352.49	44.34	1
	357	-	442.14	23.8	0.06"	360.87	2.98	1.08
	358	-	445.03	24.3	0.04"	361.03	6.63	0.8
	354	-	441.67	24.7	0.08"	357.91	8.93	1.1
	355	-	438.71	23.5	0.06"	358.92	3.42	1.1
	moy.	**-**		**24.22**	**0.05"**			**1.01**

- : instance correspondante non résolue en moins d'1h

TAB. 6.4 – Résultats des 2 approches ($n = 80, k = \frac{n}{4}$)

		HR			*EQCR*			
d(%)	*opt*	CPU	*borne*	gap(%)	CPU	*borne*	β^*	gap (%)
25	94	-	203.16	116	-	109.85	4.53	16.8
	93	-	199.58	114	36'48"	107.22	5.41	15.2
	100	-	203.22	103	5'31"	114.08	23.03	14.08
	96	-	207.91	116	26'9"	110.25	4.48	14.8
	97	-	212.5	119	-	111.88	2.9	15.3
	moy.	**-**		**113.6**	**-**			**15.2**
50	149	-	394.85	165	-	167.71	0.33	12.5
	148	-	394.30	166.4	-	166.21	3.458	12.3
	144	-	398.55	176	-	164.95	0.724	14.5
	144	-	395.09	174	-	162.05	4.739	12.5
	149	-	402.85	170	-	169.21	5.34	13.5
	moy.	**-**		**170**	**-**			**13**
75	182	-	594.10	226	-	202.188	0.545	11.09
	182	-	592.94	225	-	203.70	0.832	11.9
	183	-	600.12	228	-	205.95	1.503	12.5
	183	-	595.6	225	-	202.97	12.03	10.9
	183	-	590.14	222	-	201.59	5.159	10.1
	moy.	**-**		**225**	**-**			**11.29**

- : instance correspondante non résolue en moins d'1h

TAB. 6.5 – Résultats des 2 approches ($n = 80, k = \frac{n}{2}$)

		HR			*EQCR*			
d(%)	*opt*	CPU	*borne*	gap(%)	CPU	*borne*	β^*	gap (%)
25	280	-	406.33	45	5'47"	293.62	4.53	4.8
	273	-	399.16	46.2	32'55"	287.63	5.41	5.3
	285	-	406.44	42.6	10'39"	300.13	23.03	5.3
	284	-	415.83	46.4	4'9"	296.77	4.48	4.4
	292	-	425.04	45.6	8'7"	305.9	2.9	4.7
	moy.	**-**		**45**	**12'19"**			**4.9**
50	488	-	789.71	61.8	9'51"	504.89	0.33	3.4
	489	-	788.62	61.3	17'47"	506.08	3.458	3.6
	484	-	797.10	64.7	11'31"	500.02	0.724	3.3
	479	-	790.18	65	21'35"	494.99	4.739	3.3
	495	-	805.71	62.8	13'26"	512.14	5.34	3.46
	moy.	**-**		**63**	**14'50"**			**3.4**
75	671	-	1188.22	77	4'7"	683.11	0.545	1.8
	668	-	1185.9	77.5	19'42"	682.66	0.832	2.2
	665	-	1200.25	80.5	-	685.54	1.503	3
	669	-	1191.20	78.1	39'36"	683.86	12.03	2.2
	657	-	1180.28	79.6	-	674.3	5.159	2.6
	moy.	**-**		**78.5**	**-**			**2.36**

- : instance correspondante non résolue en moins d'1h

TAB. 6.6 – Résultats des 2 approches ($n = 80, k = \frac{3n}{4}$)

		HR			*EQCR*			
d(%)	*opt*	CPU	*borne*	gap(%)	CPU	*borne*	β^*	gap (%)
25	518	-	609.36	17.6	54.27"	530.701	4.53	2.4
	512	-	598.60	16.9	56.13"	524.5	5.41	2.4
	523	-	609.64	16.5	47.44"	536.43	23.03	2.5
	528	-	623.70	18.1	1'24"	541.02	4.48	2.46
	543	-	637.46	17.39	24.22"	554.56	2.9	2.1
	moy.	**-**		**17.29**	**53.32"**			**2.37**
50	968	-	1184.55	22.4	36.36"	982.85	0.33	1.5
	983	-	1182.91	20.3	2.32"	993.72	3.458	1.09
	972	-	1195.65	23	3'11"	988.51	0.724	1.7
	972	-	1185.275	21.9	4.05"	982.25	4.739	1.05
	989	-	1208.56	22.2	53.93"	1004.16	5.34	1.5
	moy.	**-**		**21.96**	**57.61"**			**1.36**
75	1413	-	1782.32	26.1	28.99"	1424.47	0.545	0.8
	1410	-	1778.84	26.1	7.75"	1420.72	0.832	0.76
	1417	-	1800.37	27.05	2'6"	1431.36	1.503	1.01
	1416	-	1786.80	26.2	5.88"	1425.73	12.03	0.68
	1394	-	1770.42	27	20.05"	1404.80	5.159	0.77
	moy.	**-**		**26.5**	**37.8"**			**0.8**

- : instance correspondante non résolue en moins d'1h

Les temps de calcul des valeurs $\lambda_{min}(\Delta)$, β^* et $\lambda_{min}(\Delta_{\beta^*})$ sont très rapides (moins de 1 seconde), tout comme ceux de $\gamma(0)$ et $\gamma(\beta^*)$ (section 4.2).

n=40 (Tableaux 6.1, 6.2 et 6.3)

- Pour $k = \frac{n}{4}$ (Tableau 6.1) et $d = 25\%$, la méthode EQCR permet de trouver la valeur optimale très rapidement (moins de 1.13") alors que la résolution par la méthode de la plus petite valeur propre peut mettre jusqu'à 11'16". Pour les densités de 50% et 75%, la méthode de la plus petite valeur propre ne permet de résoudre aucune instance à l'optimum en moins d'une heure alors que EQCR trouve toujours la solution optimale en moins de 4.17" si $d = 50\%$ et 1'3" si $d = 75\%$.
  Les sauts d'intégrité, quelles que soient les densités, sont en moyenne égaux à 18% pour EQCR alors que pour la seconde approche, ils varient entre 75.9% et 229.7%
- Pour $k = \frac{n}{2}$ (Tableau 6.2), comme pour $k = \frac{n}{4}$, la reformulation de la plus petite valeur propre ne permet pas de résoudre toutes les instances en moins d'une heure de temps CPU. Alors que, quelles que soient les densités, EQCR met moins de 0.44" et les sauts d'intégrité associés varient entre 2.6% et 7.6%.
- Pour $k = \frac{3n}{4}$ (Tableau 6.3), avec la méthode EQCR, chaque instance est résolue en moins de 0.3" et le saut d'intégrité varie entre 0.08 et 3.9%. La méthode de la plus petite valeur propre est ici aussi nettement moins efficace (sur le calcul de la borne inférieure du problème reformulé et donc sur les temps CPU).

n=80 (Tableaux 6.4, 6.5 et 6.6)

- Pour $k = \frac{n}{4}$ (Tableau 6.4), la méthode EQCR permet de résoudre seulement 3 instances sur 15 en moins d'une heure contre aucune pour la méthode de la plus petite valeur propre. Cependant, le saut d'intégrité est jusqu'à 20 fois inférieur.
- Pour $k = \frac{n}{2}$ (Tableau 6.5), 12 instances sur 15 sont résolues en moins d'une heure grâce à la reformulation EQCR. Le plus grand temps CPU est de 39'36" et en moyenne, la méthode trouve la solution optimale en 13'. La méthode de la plus petite valeur propre ne résout aucune instance en moins d'une heure et les sauts d'intégrité associés varient

entre 42.6% et 80.5% contre 1.8% et 5.3% pour EQCR.

- Pour $k = \frac{3n}{4}$ (Tableau 6.6), EQCR permet de résoudre toutes les instances à l'optimum (aucune n'est résolue avec la méthode de la plus petite valeur propre). Les temps CPU ne dépassent jamais 3'11" et les sauts d'intégrité sont relativement faibles.

Quelle que soit l'instance considérée, la borne obtenue par la reformulation EQCR ($\beta = \beta^*$) est supérieure à celle de la reformulation de la plus petite valeur propre ($\beta = 0$), ce qui illustre bien la proposition 7 de la section 4.4. Par conséquent, les temps de résolution sont nettement inférieurs.

De plus, les résultats expérimentaux montrent que EQCR améliore les méthodes de la littérature pour ce problème combinatoire puisqu'elle permet d'obtenir des solutions exactes pour la majorité des instances de taille au plus égale à 80 sommets et k égal à $\frac{n}{2}$ et $\frac{3n}{4}$.

6.2.2 Application des méthodes IQCR et QCR

Nous allons maintenant appliquer la méthode la plus générale au problème du k-cluster : la reformulation QCR.

Appliquer la méthode QCR requiert, dans la *Phase I*, de résoudre le programme semidéfini suivant :

$$
\begin{aligned}
(SDKC_2) \;:\; \min \quad & \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \delta_{ij} X_{ij} \\
\text{s.c.} \quad & X_{ii} = x_i \qquad\qquad i = 1, \ldots, n \qquad (6.2) \\
& -k x_i + \sum_{j=1}^{n} X_{ij} = 0 \quad i = 1, \ldots, n \qquad (6.3) \\
& \sum_{i=1}^{n} x_i = k \\
& \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\
& x \in \mathbb{R}^n,\ X \in S_n
\end{aligned}
$$

Ensuite, il suffit d'extraire les valeurs u^* et α^*, qui sont les variables duales optimales associées aux contraintes (6.2) et (6.3) respectivement pour construire le nouveau problème (KC_{α^*,u^*}), équivalent à (KC), et dont la relaxation continue est convexe :

$$(KC_{\alpha^*,u^*}) \quad : \quad \min \{q_{\alpha^*,u^*}(x) = x^t \Delta x + \sum_{i=1}^{n} u_i^*(x_i^2 - x_i) + \sum_{i=1}^{n} \alpha_i^* x_i \left(\sum_{j=1}^{n} x_j - k \right) : \sum_{i=1}^{n} x_i = k,\ x \in \{0,1\}^n\}.$$

La *Phase II* consiste tout simplement à soumettre (KC_{α^*,u^*}) à un solveur MIQP. Nous savons qu'à la racine de l'arbre de recherche du branch-and-bound, la borne obtenue par relaxation continue est égale à la valeur optimale de $(SDKC_2)$.

Afin de tester l'efficacité de l'ajout simultané des deux fonctions nulles $q_\alpha(x)$ et $q_u(x)$ $(\alpha \in \mathbb{R}^n, u \in \mathbb{R})$ sur l'ensemble admissible, nous allons comparer la méthode QCR avec la formulation (KC_{u_2}), autrement dit IQCR, qui ne perturbe que la diagonale de Δ. Pour cette approche, le programme SDP suivant doit être résolu :

$$\begin{aligned} (SDKC_3): \quad \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} \delta_{ij} X_{ij} \\ \text{s.c.} \quad & X_{ii} = x_i \qquad\qquad i = 1, \ldots, n \qquad (6.4) \\ & \sum_{i=1}^{n} x_i = k \\ & \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\ & x \in \mathbb{R}^n,\ X \in S_n \end{aligned}$$

On utilise ici les mêmes instances que la section 6.2.1 et le même processeur que celui décrit dans la section 6.1. Les tableaux 6.7 à 6.15 présentent les résultats complets pour tous les graphes :

Légende des tableaux 6.7 à 6.15 :

- d, densité du graphe
- opt, valeur de la solution optimale ou de la meilleure solution connue
- CPU, temps CPU total requis par $CPLEX9$ pour résoudre $(Q01_{\alpha^*,u^*})$ et $(Q01_{u_{2^*}})$. Le signe '-' indique que l'instance correspondante n'a pu être résolue à l'optimum en moins d'une heure
- $borne$, borne à la racine de l'arbre de recherche du branch-and-bound, i.e. la valeur optimale de la relaxation continue de $(Q01_{\alpha^*,u^*})$ ou $(Q01_{u^{2^*}})$
- $gap = \frac{bound-opt}{opt} * 100$, l'écart relatif entre la meilleure solution obtenue et la borne inférieure calculée à la racine de l'arbre de recherche (saut d'intégrité)
- #nœuds, nombre de noeuds dans l'arborescence de recherche
- **moy.**, moyennes, pour chaque triplet (n, k, d), des temps CPU et des sauts d'intégrité de IQCR et QCR

Les temps de résolution correspondants aux calculs de (α^*, u^*) (méthode QCR) et de u_2^* (méthode IQCR) par le solveur SB ne sont pas indiqués dans les tableaux suivants : ils sont très petits, toujours inférieurs à 1 seconde. De même, le calcul de la borne, c'est-à-dire de la solution de la relaxation continue de $(Q01_{\alpha^*,u^*})$ ou $(Q01_{u_{2^*}})$, est très rapide.

TAB. 6.7 – Résultats des deux algorithmes ($n = 40$, $k = \frac{n}{4}$)

		QCR				*IQCR*			
$d(\%)$	*opt*	*CPU*	*borne*	*gap*	*#nœuds*	*CPU*	*borne*	*gap*	*#nœuds*
25	29	0.04"	30.63	5.6	43	15.33"	49.9	72	88848
	30	0.07"	32.08	6.9	150	1'46"	56.41	88	579430
	27	0.36"	30.42	12.6	1231	4'25"	53.18	96	1361551
	27	0.16"	29.95	10.9	508	59.06"	49.41	83	337578
	26	0.18"	29.12	12	656	1'16"	47.92	84	431958
	moy.	**0.16"**		**9.63**		**1'44"**		**84.6**	
50	40	0.19"	43.46	8.6	645	-	97.16	142.9	13733723
	41	0.15"	44.38	8.2	459	-	105.99	158.5	18043481
	39	0.92"	43.54	11.6	3765	-	103.67	165.8	18463871
	40	0.28"	43.64	9.1	999	-	100.25	150.6	19319151
	40	0.23"	43.65	9.1	820	-	97.96	144.9	16771867
	moy.	**0.36"**		**9.32**		**-**		**152.54**	
75	45	2.58"	50.69	12.6	12996	-	145.02	222.2	20629251
	45	5.25"	50.77	12.8	31820	-	147.05	226.7	20569711
	45	4.15"	50.36	11.9	21951	-	148.03	228.9	21253751
	45	3.39"	50.27	11.7	18557	-	146.92	226.5	20678313
	45	4.41"	50.58	12.4	27111	-	145.86	224.1	20828406
	moy.	**3.96"**		**12.28**		**-**		**185.68**	

- : instance correspondante non résolue en moins d'1h

TAB. 6.8 – Résultats des deux algorithmes ($n = 40$, $k = \frac{n}{2}$)

		QCR				*IQCR*			
$d(\%)$	*opt*	*CPU*	*borne*	*gap*	*#nœuds*	*CPU*	*borne*	*gap*	*#nœuds*
25	77	0.08"	79.44	3.2	212	29.57"	97.55	25.7	171754
	84	0.06"	86.08	2.5	110	2'8"	111.61	32.8	631749
	76	0.18"	79.15	4.1	526	19'	105.36	38.6	4353901
	75	0.04"	76.84	2.4	54	53.07"	97.46	29.9	304227
	73	0.08"	75.28	3.1	173	59.12"	94.54	29.5	325885
	moy.	**0.08"**		**3.06**		**4'42"**		**31.3**	
50	130	0.03"	131.48	1.1	32	-	193.17	48.66	18387481
	130	0.62"	134.53	3.5	2500	-	211.21	62.4	21678192
	133	0.12"	136.16	2.4	349	-	206.23	55	19876820
	128	0.21"	131.75	2.9	728	-	199.38	55.8	20606526
	129	0.13"	132.37	2.6	435	-	194.78	50.9	20366412
	moy.	**0.22"**		**2.5**		**-**		**54.55**	
75	168	0.1"	170.80	1.7	249	-	289.54	72.3	22372032
	173	0.06"	175.15	1.2	111	-	293.43	69.6	22551071
	171	0.13"	173.61	1.5	370	-	295.42	72.8	21920075
	171	0.09"	173.43	1.4	251	-	293.23	71.4	22343940
	171	0.13"	173.86	1.7	419	-	290.97	70	22470031
	moy.	**0.102**		**1.5**		**-**		**71.22**	

- : instance correspondante non résolue en moins d'1h

TAB. 6.9 – Résultats des deux algorithmes ($n = 40$, $k = \frac{3n}{4}$)

		QCR				*IQCR*			
d(%)	*opt*	*CPU*	*borne*	*gap*	*#nœuds*	*CPU*	*borne*	*gap*	*#nœuds*
25	134	0.03"	135.11	0.8	3	0.15"	142.52	6.3	954
	148	0.03"	149.66	1.1	34	7.62"	165.54	11.8	51514
	138	0.06"	140.57	1.9	207	15.53"	156.51	13.4	108802
	134	0.03"	135.20	0.9	8	0.22"	143.76	7.2	1438
	130	0"	130.97	0.7	0	0.31"	139.65	7.4	1953
	moy.	**0.12"**		**1.08**		**4.77"**		**9.22**	
50	247	0.03"	248.34	0.5	22	2'25"	287.90	16.5	714591
	263	0.03"	264.86	0.7	43	31'6"	315.59	19.9	3444769
	261	0.05"	263.48	0.9	92	4'4"	307.69	17.8	1177113
	254	0.05"	256.63	1	99	2'29"	297.19	17	842617
	250	0.04"	251.96	0.8	41	1'34"	290.27	16.1	568437
	moy.	**0.04"**		**0.78**		**8'19"**		**17.46**	
75	349	0.03"	350.67	0.5	33	-	433.54	24.2	18757433
	357	0.04"	359.02	0.6	49	-	439.10	22.9	13360731
	358	0.03"	359.39	0.4	23	-	442.1	23.5	16029881
	354	0.03"	355.58	0.4	41	-	438.93	23.9	17658571
	355	0.04"	356.93	0.5	41	-	435.36	22.6	8502981
	moy.	**0.034"**		**0.48**		**-**		**23.42**	

- : instance correspondante non résolue en moins d'1h

TAB. 6.10 – Résultats de la méthode QCR ($n = 80$, $k = \frac{n}{4}$)

d(%)	*opt*	*CPU*	*borne*	*gap*	*#nœuds*
25	94	1'56"	102.71	9.2	190016
	93	2'5"	102.26	9.9	203159
	100	8.21"	107.11	7.1	13739
	96	1'21"	105.12	9.5	129227
	97	5'2"	106.84	10.1	501668
	moy.	**2'6"**		**9.16**	
50	149	1'22"	160.34	7.6	129556
	148	1'51"	158.92	7.3	181519
	144	16'25"	157.99	9.7	1592333
	144	6'3"	155.28	7.8	592656
	149	1'27"	160.36	7.6	140435
	moy.	**5'26"**		**8**	
75	182	20'40"	193.63	6.4	2112340
	182	53'57"	194.99	7.1	5291081
	184	17'8"	196.80	6.9	1662192
	183	15'6"	194.36	6.2	1481003
	183	3'22"	193.30	5.6	305878
	moy.	**22'3"**		**6.44**	

TAB. 6.11 – Résultats de la méthode QCR ($n = 80$, $k = \frac{n}{2}$)

$d(\%)$	*opt*	*CPU*	*borne*	*gap*	*#nœuds*
25	280	12.21"	287.09	3.9	20379
	273	51.77"	281.32	3	82008
	285	9.48"	292.17	2.5	15982
	284	12.27"	291.85	2.8	18736
	292	12.17"	299.52	2.6	20104
	moy.	**19.6"**		**2.96**	
50	488	22.33"	497.58	2	37078
	489	27.21"	497.53	1.7	45882
	484	27.26"	493.19	1.9	41950
	479	26.57"	487.25	1.7	40501
	495	11.31"	503.54	1.7	17308
	moy.	**18.54"**		**1.8**	
75	671	10.09"	677.87	1	15691
	668	51.42"	676.83	1.3	82276
	665	9'57"	676.63	1.7	924400
	669	1'8"	677.37	1.2	107320
	657	2'38"	666.33	1.4	251302
	moy.	**2'57"**		**1.32**	

TAB. 6.12 – Résultats de la méthode QCR ($n = 80$, $k = \frac{3n}{4}$)

$d(\%)$	*opt*	*CPU*	*borne*	*gap*	*#nœuds*
25	518	1.09"	523.07	0.9	2080
	512	0.8"	516.71	0.9	1462
	523	0.43"	526.81	0.7	587
	528	1.88"	533.57	1.1	3763
	543	0.38"	546.7	0.7	605
	moy.	**0.92"**		**0.86**	
50	968	1.26"	974.26	0.5	2583
	983	0.31"	987.11	0.4	357
	972	6.19"	979.72	0.8	13097
	972	0.39"	976.14	0.4	597
	989	2.29"	996.28	0.7	5156
	moy.	**2.09"**		**0.56**	
75	1413	7.02"	1420.53	0.5	15638
	1410	1.36"	1415.44	0.4	2728
	1417	7.94"	1424.40	0.5	16644
	1416	0.61"	1420.38	0.3	1040
	1394	1.06"	1398.94	0.4	1912
	moy.	**3.6"**		**0.42**	

TAB. 6.13 – Résultats de la méthode QCR ($n = 100$, $k = \frac{n}{4}$)

$d(\%)$	*opt*	*CPU*	*borne*	*gap*	*#nœuds*
25	139	-	151.58	9	3879506
	138	-	153.46	11.2	3977611
	142	27'26"	153.99	8	1713934
	140	15'46"	152.02	8.6	979105
	145	30'23"	157.66	8.7	1948478
	moy.	**-**		**9.1**	
50	218	-	236.97	8.7	3874652
	221	-	241.23	9	3798921
	216	-	232.92	7.8	4005381
	221	32'58"	236.68	7	2026697
	221	-	236.28	6.9	3846491
	moy.	**-**		**7.88**	
75	282	-	297.81	5.6	3987021
	282	-	298.92	6	4200221
	285	-	301.41	5.7	3908591
	288	-	303.43	5.3	3983371
	286	-	301.38	5.4	4105875
	moy.	**-**		**5.6**	

- : instance correspondante non résolue en moins d'1h

TAB. 6.14 – Résultats de la méthode QCR ($n = 100$, $k = \frac{n}{2}$)

$d(\%)$	*opt*	*CPU*	*borne*	*gap*	*#nœuds*
25	417	21'17"	428.54	2.8	1385156
	417	7'24"	429.01	2.9	464713
	429	1'32"	437.90	2.1	102571
	413	3'3"	423.03	2.4	191163
	435	6'12"	446.43	2.6	379615
	moy.	**7'54"**		**2.56**	
50	729	-	745.07	2.2	3983441
	740	-	757.57	2.4	3916567
	729	6'32"	740.64	1.6	396780
	729	-	747.09	2.5	3950406
	733	-	749.10	2.2	4002221
	moy.	**-**		**2.18**	
75	1029	3'46"	1039.01	0.9	250625
	1035	15'42"	1046.58	1.1	1006513
	1047	14'34"	1058.77	1.1	888108
	1055	7'51"	1066.04	1	512979
	1049	3'49"	1058.75	0.9	244333
	moy.	**9'8"**		**1**	

- : instance correspondante non résolue en moins d'1h

TAB. 6.15 – Résultats de la méthode QCR ($n = 100$, $k = \frac{3n}{4}$)

$d(\%)$	opt	CPU	$borne$	gap	$\#n\text{œ}uds$
25	796	2.03"	801.97	0.7	2564
	787	4.3"	793.48	0.8	6029
	809	3.49"	815.01	0.7	5167
	776	29.53"	783.88	1	43107
	828	5.64"	834.69	0.8	8115
	moy.	**9"**		**0.8**	
50	1483	2'43"	1494.76	0.8	240649
	1518	42.77"	1529.28	0.7	60137
	1476	6.48"	1483.33	0.5	9226
	1487	34.4"	1497.96	0.7	50457
	1495	7.4"	1502.75	0.5	10512
	moy.	**50.81"**		**0.64**	
75	2175	14.11"	2182.72	0.3	22351
	2192	25.11"	2200.64	0.4	36487
	2230	17.32"	2237.96	0.4	25226
	2235	4.7"	2242.07	0.3	7066
	2229	10.37"	2235.86	0.3	14093
	moy.	**14.32"**		**0.34**	

n=40 (Tableaux 6.7, 6.8 et 6.9)

- Pour $k = \frac{n}{4}$, la reformulation QCR trouve la valeur optimale en moins de 5.25", toutes densités confondues, alors que IQCR ne permet pas de résoudre les instances à l'optimum en moins d'une heure si la densité du graphe est de 50% ou 75%. Cependant, pour une densité de 25% le temps CPU de IQCR est relativement faible (jusqu'à 4'25") mais toujours supérieur à celui requis par l'approche QCR. Les sauts d'intégrité suivent la même évolution : ils varient entre 5.6 et 12.6% pour QCR contre 72 et 228.9% pour IQCR.
- Pour $k = \frac{n}{2}$, QCR permet de résoudre chaque instance en moins de 0.62" et le saut d'intégrité associé est de 2% en moyenne. Les conclusions sur les résultats obtenus par la reformulation IQCR sont quasiment identiques que pour le cas où $k = \frac{n}{4}$, les sauts d'intégrité étant cependant un peu moins élevés.
- Pour $k = \frac{3n}{4}$, les sauts d'intégrité pour QCR sont meilleurs que pour les autres valeurs de k (de 0.4% à 1.9%) et par conséquent, les temps CPU sont faibles. On observe encore une fois les moins bonnes performances

de la méthode IQCR.

Compte tenu de ces premiers résultats pour les graphes de plus petite taille, nous avons choisi de ne pas continuer la comparaison des deux méthodes pour les graphes à plus de 40 sommets.

n=80 (Tableaux 6.10, 6.11 et 6.12)

Comme pour les graphes de taille 40, les meilleurs résultats sont obtenus pour $k = \frac{3n}{4}$. Dans ce cas, la solution optimale est donnée en moins de 7.02". Pour $k = \frac{n}{2}$, chaque instance est résolue en moins de 10'. Les plus grands temps CPU sont obtenus pour $k = \frac{n}{4}$ (de 8.21" à 53'57").

n=100 (Tableaux 6.13, 6.14 et 6.15)

Pour $n = 100$ et $k = \frac{3n}{4}$, toutes les instances sont résolues en moins d'une heure. C'est également le cas pour $k = \frac{n}{2}$ quand la densité prend respectivement les valeurs 25% et 75%. Pour toutes les instances, la plus difficile à traiter pour notre approche semble être celle avec une valeur de k égale à $\frac{n}{4}$. Nous pouvons noter que dans ce cas, même si la solution optimale est rarement trouvée en moins d'une heure, le saut d'intégrité à la racine de l'arbre de recherche est relativement bas, généralement inférieur à 10%.

Si l'on compare les résultats obtenus avec les méthodes QCR et IQCR, on voit clairement que la meilleure méthode est la formulation la plus générale (démontré théoriquement dans la section 4.4). En effet, la valeur de la borne est améliorée et par conséquent, le temps de résolution en 0-1 est en moyenne 2 fois plus rapide avec la méthode QCR. De plus, elle permet aussi de résoudre toutes les instances pour des graphes allant jusqu'à 80 sommets en moins d'une heure.

6.2.3 Synthèse des résultats

Nous avons testé les quatre approches présentées dans le chapitre 4 sur le problème du k-cluster. Si l'on compare les résultats obtenus pour ces 4 méthodes, il apparait clairement que la reformulation QCR est la plus performante. Le tableau suivant présente un petit récapitulatif sur le nombre d'instances résolues en moins d'une heure, pour chaque reformulation et chaque taille des graphes. On rappelle que pour chaque valeur de n (=40,80,100),

45 instances sont considérées.

TAB. 6.16 – Comparaison des quatre approches sur le nombre d'instances résolues en moins d'1h pour 45 instances testées $\forall n$

n	HR	EQCR	IQCR	QCR
40	20	**45**	20	**45**
80	0	30	0	**45**
100	0	0	0	**30**

Si l'on étudie maintenant les différents sauts d'intégrité à la racine de l'arborescence de recherche pour chaque couple (n, d) et pour chacune des quatre approches, on voit clairement que les bornes obtenues par relaxation continue des problèmes transformés par QCR sont nettement supérieures (on note que le signe '-' correspond aux instances non résolues par les méthodes correspondantes) :

TAB. 6.17 – Comparaison des sauts d'intégrité obtenus avec les quatre approches

n	d	HR	EQCR	IQCR	QCR
40	25	45.44	26.7	41.71	**4.75**
	50	76.35	7.33	74.85	**4.2**
	75	107.42	8.30	93.44	**4.56**
80	25	58.63	7.49	-	**4.33**
	50	84.99	5.92	-	**3.45**
	75	110	4.82	-	**2.72**
100	25	-	-	-	**4.15**
	50	-	-	-	**3.57**
	75	-	-	-	**2.31**

D'après les résultats du tableau 6.2.3, la méthode de la plus petite valeur propre est clairement la plus faible car les moyennes des sauts d'intégrité obtenus à la racine de l'arborescence de recherche sont les plus importants (entre 45.44% et 110%). Par ailleurs, les sauts d'intégrité obtenus avec EQCR et IQCR sont très différents : c'est avec EQCR que l'on obtient une plus grande borne inférieure. Plus précisément, pour les graphes à 40 sommets, on passe d'un saut d'intégrité de 93.44% quand $d = 75\%$ avec IQCR à un

saut de 8.30% avec EQCR.

Comparaison avec la méthode de Billionnet [15]

Comparé à l'approche développée par Billionnet, les résultats obtenus avec QCR sont meilleurs en terme de taille d'instances et de temps de résolution. Nous sommes capable de résoudre de manière efficace des graphes allant jusqu'à 100 sommets. Lorsque $n = 80$, QCR permet de résoudre toutes les instances à l'optimum en moins de 20 minutes, excepté pour une instance (53') alors que les formulations de Billionnet ne permettent pas de résoudre les graphes à 68 sommets.

6.3 Le problème de la bipartition de graphe

Soit un graphe non orienté $G = (V, U)$ de n sommets $\{v_1, \ldots, v_n\}$ et un ensemble d'arêtes pondérées E. Le problème de la bipartition consiste à partitionner les sommets de G en deux sous-ensembles V_1 et V_2 tels que :

1. $V_1 \cup V_2 = V$ et $V_1 \cap V_2 = \varnothing$
2. $\mid V_1 \mid = p$ et $\mid V_2 \mid = n - p$
3. le poids de la coupe soit minimale.

Par exemple, considérons le graphe $G = (V, U)$ suivant à 6 sommets et 9 arêtes,

FIG. 6.3 – Problème de la bipartition de graphe : $n = 6$

et posons $p = 3$. Dans ce cas précis, $\mid V_1 \mid = \mid V_2 \mid = \frac{n}{2}$. C'est un cas particulier du problème de bipartition : l'équipartition de graphe.

FIG. 6.4 – Problème de la bipartition de graphe : $n = 6$ et $p = 3$

Dans cet exemple, une solution optimale consiste à sélectionner dans V_1 les 3 sommets v_1, v_2 et v_6 et dans V_2 les 3 autres sommets v_3, v_4 et v_5. La valeur optimale associée est égale à la somme des poids des arêtes ayant une extrémité dans V_1 et l'autre dans V_2, c'est-à-dire 4 ($[v_2, v_3], [v_3, v_6], [v_5, v_6]$).

Ce problème se modélise comme un programme quadratique en variables 0-1 sous contrainte linéaire :

$$(BP) \quad : \quad \min \left\{ g(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left(x_i(1 - x_j) + (1 - x_i)x_j \right) \quad : \sum_{i=1}^{n} x_i = p, \ x \in \{0,1\}^n \right\}$$

où c_{ij} est le poids de l'arête $[v_i, v_j]$. La variable binaire x_i est égale à 1 si et seulement si le sommet v_i est dans V_1.

Ce problème a de nombreuses applications dans des domaines très variés tels que la programmation scientifique [44], la physique [10], les algorithmes parallèles [48].

6.3.1 Un état de l'art

De nombreuses heuristiques ont été proposées ([11], [12], [32], [48], [87], [91], [115], [121]) ainsi que des calculs de borne inférieure ([26], [50], [52], [105], [56]).

Nous nous intéressons tout particulièrement aux méthodes exactes. Pour le problème de l'équipartition de graphe (i.e. $p = \frac{n}{2}$), Roucairol et Hansen

[122] suggèrent une approche basée sur la dualisation de la contrainte de cardinalité. Brunetta, Conforti et Rinaldi [30] proposent également une méthode utilisant un branch-and-cut et une relaxation de la programmation linéaire. Ferreira et al. [55] décrivent une approche similaire pour le partitionnement de graphes avec capacité sur les sommets tandis que Johnson et al. [88] présentent une approche basée sur la génération de colonnes.

Lorsque p est quelconque, Christofides et Brooker [39] proposent un algorithme basé sur le calcul d'une borne inférieure obtenue par la résolution de problèmes de flot maximal. Un algorithme de branch-and-bound parallèle est implémenté par Clausen et Träff [40] tandis que Michelon, Ripeau et Maculan [111] suggèrent une méthode de branch-and-bound utilisant une approximation de l'ensemble admissible par une ellipsoïde.

Toutes ces méthodes de résolution exacte permettent de résoudre des graphes avec au plus 60 sommets. De meilleurs résultats sont obtenus par Karisch, Rendl et Clausen [90]. Ils décrivent une approche combinant la programmation semidéfinie et la méthode des plans coupants. Ils peuvent résoudre des instances avec 80-90 sommets et obtiennent de fines approximations pour les graphes de plus grande taille.

La section 6.3.2 donne un bref aperçu de la reformulation QCR de (BP). La section 6.3.3 est consacrée aux expérimentations et aux comparaisons de nos résultats avec ceux de Karisch, Rendl et Clausen.

6.3.2 Reformulation convexe du problème de la bipartition de graphe

Le problème (BP) est équivalent au problème suivant, dans lequel, dans la fonction objectif, les termes linéaires sont séparés des termes quadratiques :

$$(BP) \quad : \quad \min \left\{ g(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} -2c_{ij}x_ix_j + \sum_{i=1}^{n} C_ix_i \quad : \sum_{i=1}^{n} x_i = p,\ x \in \{0,1\}^n \right\}$$

où $C_i = \sum_{j=1}^{i-1} c_{ji} + \sum_{j=i+1}^{n} c_{ij}$.

Si l'on applique QCR à (BP), on obtient donc un nouveau problème :

$$(BP_{\alpha,u}) \quad : \quad \min \left\{ g_{\alpha,u}(x) : \sum_{i=1}^{n} x_i = p,\ x \in \{0,1\}^n \right\}$$

avec :

$$g_{\alpha,u}(x) = g(x) + \sum_{i=1}^{n} \alpha_i x_i \left(\sum_{j=1}^{n} x_j - p \right) + \sum_{i=1}^{n} u_i \left(x_i^2 - x_i \right)$$

Puisque (BP) a seulement une contrainte, α est un vecteur à n composantes. Soit $(\overline{BP_{\alpha,u}})$ la relaxation continue de $(BP_{\alpha,u})$ i.e. :

$$(\overline{BP_{\alpha,u}}) \quad : \quad \min \left\{ g_{\alpha,u}(x) : \sum_{i=1}^{n} x_i = p,\ x \in [0,1]^n \right\}$$

Les paramètres optimaux (α^*, u^*) sont obtenus en résolvant la relaxation semidéfinie de (BP) suivante, dont la valeur optimale est égale à celle de $(\overline{BP_{\alpha^*,u^*}})$:

$$\begin{aligned}
(SDBP) \quad \min \quad & \sum_{i=1}^{n} \sum_{j=i+1}^{n} -2c_{ij}X_{ij} + \sum_{i=1}^{n} C_i X_{ii} & & \\
\text{s.c.} \quad & X_{ii} = x_i & i = 1, \ldots, n \qquad & (6.5) \\
& -px_i + \sum_{j=1}^{n} X_{ij} = 0 & i = 1, \ldots, n \qquad & (6.6) \\
& \sum_{i=1}^{n} x_i = p & & \\
& \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 & & \\
& x \in \mathbb{R}^n, X \in S_n & &
\end{aligned}$$

u^* et α^* sont respectivement les valeurs optimales des variables duales associées aux contraintes d'égalité (6.5) et (6.6).

6.3.3 Résultats expérimentaux sur des graphes non pondérés générés aléatoirement

L'environnement expérimental est celui détaillé dans la section 6.1. Notons qu'une contrainte est ajouté à (BP) quand $p = \frac{n}{2}$ dans le but d'accélerer

le branch-and-bound : elle consiste à fixer à 1 la variable correspondant au sommet de plus grand degré.

Nous appliquons tout d'abord la méthode QCR à des graphes générés aléatoirement, non pondérés. Ils sont générés de la même manière que pour le problème du k-cluster : pour une densité donnée d et un couple d'indices (i, j) tel que $i < j$, un nombre aléatoire $\rho \in [0, 1]$ est généré. Si $\rho > d$, alors c_{ij} est fixé à 0 ; sinon, c_{ij} est fixé à 1. Les tableaux 6.18 à 6.21 présentent les résultats pour 5 tailles de graphe ($n = 40, 60, 80, 90, 100$), 3 densités ($d = 25\%, 50\%, 75\%$) et 3 valeurs de p ($p = \lfloor \frac{n}{8} \rfloor, \lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{2} \rfloor$). Pour chaque triplet (n, p, d), nous résolvons 5 instances.

Légende des tableaux 6.18 à 6.21 :

- d, densité du graphe.
- opt, valeur de la solution optimale ou de la meilleure solution connue.
- CPU_{QCR}, temps CPU total requis par le solveur MIQP de CPLEX9 pour résoudre $(BP_{\alpha,u})$ et par conséquent (BP).
- CPU_{QCR} *moy.* est la moyenne des temps CPU pour 5 instances. *best* (resp. *worst*) CPU_{QCR}, le meilleur (resp. le moins bon) temps CPU des 5 instances pour une densité donnée.
- $borne_{QCR}$, borne à la racine de l'arbre de recherche, i.e. la valeur optimale de la relaxation continue de $(BP_{\alpha,u})$ qui est aussi la valeur optimale de $(SDBP)$.
- $gap_{QCR} = \frac{opt - borne_{QCR}}{opt} * 100$.
- $\#n\text{œ}uds$, nombre de nœuds dans l'arborescence de recherche.

Le tableau 6.18 présente les résultats pour 4 valeurs de n (i.e. $40, 60, 80, 90$). Pour chaque densité, les valeurs moyennes de CPU_{QCR} et gap_{QCR} sont reportées ainsi que le meilleur et le moins bon CPU_{QCR} relevé sur 5 instances. Les tableaux 6.19 à 6.21 (i.e. $n = 100$ sommets) présentent les résultats complets.

TAB. 6.18 – Résultats de graphes aléatoires non pondérés ($n = 40, 60, 80, 90$)

n	p	$d(\%)$	CPU_{QCR} *moy.*	*best* CPU_{QCR}	*worst* CPU_{QCR}	$gap_{QCR}(\%)$
40	$\lfloor \frac{n}{8} \rfloor$	25	0.04"	0.02"	0.07"	12.45
		50	0.07"	0.03"	0.12"	7.22
		75	0.08"	0.04"	0.14"	4.32
	$\lfloor \frac{n}{4} \rfloor$	25	0.15"	0.1"	0.29"	10.2
		50	0.25"	0.15"	0.45"	6.33
		75	0.36"	0.09"	1.01"	3.36
	$\lfloor \frac{n}{2} \rfloor$	25	0.54"	0.22"	1.37"	8.05
		50	0.64"	0.11"	0.85"	3.08
		75	0.53"	0.32"	1.02"	2.02
60	$\lfloor \frac{n}{8} \rfloor$	25	2.29"	0.5"	7.01"	13.68
		50	9.8"	1.8"	19.5"	5.62
		75	7.8"	1.8"	5.02"	2.8
	$\lfloor \frac{n}{4} \rfloor$	25	2"	1"	4"	9.86
		50	9.5"	1.6"	18"	5.64
		75	7.6"	1.7"	16.2"	2.82
	$\lfloor \frac{n}{2} \rfloor$	25	16.2"	4.5"	41"	7.16
		50	29"	9"	66"	3.72
		75	14.6"	8"	22"	1.78
80	$\lfloor \frac{n}{8} \rfloor$	25	5.7"	1.7"	8.6"	8.48
		50	3.8"	1.8"	8.7"	4.98
		75	14.8"	3.4"	36.8"	3.26
	$\lfloor \frac{n}{4} \rfloor$	25	2'18"	35.5"	7'	8.4
		50	4'04"	14.6"	11'05"	4.54
		75	5'09"	31.5"	9'46"	2.6
	$\lfloor \frac{n}{2} \rfloor$	25	24'21"	4'36"	50'38"	6.52
		50	10'51"	2'56"	24'13"	3.26
		75	6'49"	5'44"	10'30"	1.62
90	$\lfloor \frac{n}{8} \rfloor$	25	41.2"	1.5"	2'11"	10.04
		50	25.3"	9.11"	1'07"	5.14
		75	29.6"	10.38"	58.43"	3.08
	$\lfloor \frac{n}{4} \rfloor$	25	46'25"	16'31"	2h32'	8.76
		50	24'10"	46.6"	1h06'	3.48
		75	2h23'11"	27.1"	6h21'30"	2.56
	$\lfloor \frac{n}{2} \rfloor$	25	1h03'43"	16'58"	2h27'	5.92
		50	3h36'	9'48"	8h59'30"	3.14
		75	1h10'31"	17'28"	2h22'16"	1.6

TAB. 6.19 – Résultats de graphes aléatoires non pondérés ($n = 100, p = \lfloor \frac{n}{8} \rfloor$)

$d(\%)$	num	opt	CPU_{QCR}	$borne_{QCR}$	$gap_{QCR}(\%)$	$\#n\text{œ}uds$
25	1	178	46.8"	161.12	9.5	62548
	2	181	2'28"	161.33	10.9	192259
	3	182	29"	166.42	8.6	37599
	4	183	2'45"	163.31	10.7	211066
	5	189	20.7"	173.79	8	26134
50	1	420	1'4"	398.22	5.2	133224
	2	444	3'	418.02	5.8	226563
	3	426	1'24"	405.23	4.9	106652
	4	420	26.8"	397.83	5.3	33134
	5	424	4'48"	400.30	5.6	368080
75	1	695	51.3"	677.00	2.6	68699
	2	708	3'19"	686.15	3.1	274763
	3	723	3'24"	700.90	3.1	267130
	4	722	5'46"	698.38	3.3	474981
	5	719	2'	698.49	2.9	173249

TAB. 6.20 – Résultats de graphes aléatoires non pondérés ($n = 100, p = \lfloor \frac{n}{4} \rfloor$)

$d(\%)$	num	opt	CPU_{QCR}	$borne_{QCR}$	$gap_{QCR}(\%)$	$\#n\text{œ}uds$
25	1	335	4h36'32"	306.35	8.6	16548632
	2	326	27'13"	301.48	7.5	1613543
	3	340	1h17'	314.43	7.5	4473166
	4	331	2h08'36"	305.04	7.8	6651234
	5	356	2h27'	327.59	8	8589197
50	1	769	5h13'	734.73	4.5	17672160
	2	803	12h30'28"	761.04	5.2	43245303
	3	771	5h11'	738.89	4.2	18252528
	4	764	26'34"	733.02	4.1	1617654
	5	771	5h55'46"	738.40	4.2	20642738
75	1	1244	2'26"	1224.09	1.6	150053
	2	1271	2h07'22"	1242.62	2.2	6703433
	3	1301	4h34'45"	1271.36	2.3	16241715
	4	1292	1h23'11"	1265.53	2.0	4805338
	5	1292	42'55"	1266.43	2	2635526

TAB. 6.21 – Résultats de graphes aléatoires non pondérés ($n = 100, p = \lfloor \frac{n}{2} \rfloor$)

$d(\%)$	num	opt	CPU_{QCR}	$borne_{QCR}$	$gap_{QCR}(\%)$	$\#n\text{œ}uds$
25	1	459	13h51'24"	432.16	5.8	45887303
	2	452	14h22'47"	422.44	6.5	45334466
	3	473	29h30'03"	445.15	5.8	92752492
	4	451	7h06'19"	425.96	5.5	22557572
	5	487	20h04'	457.9	6	64184090
50	1	1039	14h51'11"	1005.47	3.2	47593207
	2	1067	1h42'31"	1036.35	2.8	4825501
	3	1044	43h41'31"	1010.35	3.2	140500675
	4	1036	2h15'51"	1008.25	2.6	6275713
	5	1048	49h39'47"	1012.82	3.3	157422406
75	1	1696	43h42'58"	1665.36	1.8	142751199
	2	1716	15h18'11"	1687.60	1.6	50102940
	3	1755	40h52'6"	1726.39	1.6	130657537
	4	1754	39h48'27"	1721.93	1.8	130988772
	5	1749	15h03'06"	1720.51	1.6	48732055

Pour tous les tableaux, les temps de calcul de (α^*, u^*) par la programmation semidéfinie ne sont pas reportés : ils sont toujours inférieurs à 1 minute.

n=40,60 (Tableau 6.18)
Pour chaque instance, la valeur optimale est déterminée en moins de 2 secondes pour $n = 40$ et en moins d'une minute pour $n = 60$.

n=80 (Tableau 6.18)

- Pour $p = \lfloor \frac{n}{8} \rfloor$, le temps de résolution est très rapide (toujours inférieur à 37 secondes).
- Pour $p = \lfloor \frac{n}{4} \rfloor$, chaque instance est résolue en moins de 11 minutes.
- Les plus mauvais résultats sont obtenus pour $p = \lfloor \frac{n}{2} \rfloor$ et $d = 25\%$. Cependant, toutes les instances sont résolues en moins d'une heure.

n=90 (Tableau 6.18)

- Pour $p = \lfloor \frac{n}{8} \rfloor$, le plus mauvais temps CPU est d'environ 2 minutes.

- Pour $p = \lfloor \frac{n}{4} \rfloor$, 11 instances sur 15 sont résolues en moins d'une heure. Cependant, une instance (avec $d = 75\%$) requiert plus de 6 heures.
- Pour $p = \lfloor \frac{n}{2} \rfloor$, 7 instances sur 15 sont résolues en moins d'une heure. Le plus mauvais temps CPU est de 9 heures pour une instance avec une densité de 50%.

n=100 (Tableaux 6.19 - 6.21)

- Pour $p = \lfloor \frac{n}{8} \rfloor$, le plus mauvais temps CPU est d'environ 6 minutes.
- Pour $p = \lfloor \frac{n}{4} \rfloor$, le plus mauvais temps CPU, égal à 12h30', est obtenu pour $d = 50\%$. Cependant, la moyenne des temps CPU pour les 15 instances est de 3h16'.
- Pour $p = \lfloor \frac{n}{2} \rfloor$, le plus mauvais temps CPU est égal à environ 50 heures pour une instance avec une densité de 50%. En moyenne, chacune des instances est résolue en 24 heures.

En remarque générale, nous pouvons observer que, quelle que soit la taille des graphes, les meilleures valeurs de saut d'intégrité sont obtenues pour les plus grandes densités. Les sauts d'intégrité moyens sont égaux à 9% pour $d = 25\%$, 4% pour $d = 50\%$ et 3% pour $d = 75\%$.

De plus, les plus longs temps CPU sont obtenus pour le problème de l'équipartition de graphes. (i.e. $p = \lfloor \frac{n}{2} \rfloor$) : intuitivement, nous pouvons penser que la taille de partition p augmente la difficulté du problème quand elle tend vers la valeur $\frac{n}{2}$ puisque le nombre de solutions réalisables est égal au nombre de combinaisons de p sommets choisis parmi n.

Comparaison avec une convexification directe

Il existe une convexification simple de la fonction objectif de (BP). En effet, dans le cas où tous les coefficients c_{ij} sont positifs, pour tout $x \in \{0,1\}^n$, on a :

$$g(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij}(x_i - x_j)^2 = g_{conv}(x)$$

Ainsi, on obtient un nouveau problème, noté (BP_{conv}) :

$$(BP_{conv}) \quad : \quad \text{Min} \quad \left\{ g_{conv}(x) \quad : \sum_{i=1}^{n} x_i = p, \ x \in \{0,1\}^n \right\}$$

dont la fonction objectif est convexe. Par conséquent, le problème (BP_{conv}) peut être directement soumis à un solveur MIQP.

Cette convexification est moins efficace que celle déterminée par la méthode QCR. En effet, la valeur optimale de la relaxation continue de (BP_{conv}) est toujours égale à 0 puisque $x_i = \frac{p}{n}$ est une solution réalisable de valeur 0. Par conséquent, nous avons choisi de ne tester cette approche que sur deux graphes de taille 40 du tableau 6.18 avec $p = \frac{n}{4}$ et deux densités différentes ($d = 25\%$ et 75%). Pour la première instance (resp. la seconde), résoudre (BP_{conv}) requiert 3'02" (resp. plus de 2h30') alors que le temps de résolution de QCR est de 0.29" (resp. 0.45").

6.3.4 Comparaison avec les résultats de Karisch, Rendl et Clausen (KRC)

Rappel de la méthode KRC [90]

Dans cette section, nous comparons notre méthode avec celle de Karisch, Rendl et Clausen (KRC), qui ont développé une méthode de résolution exacte pour le problème de bipartition de graphe. Leur approche se base sur un branch-and-bound spécifique fondé sur la programmation semidéfinie et sur les relaxations polyhédrales.

Plus précisément, ils considèrent le problème de bipartition modélisé sous la forme suivante :

$$(BIS) \ : \ \min \left\{ \frac{1}{4} x^t L x : x^t e = d, x \in \{-1, +1\}^n \right\}$$

où L représente la matrice de *Laplace* définie à partir de la matrice d'adjacence A du graphe G : $L = Diag(Ae) - A$. Si l'on note $n_1 =\mid V_1 \mid$ et $n_2 =\mid V_2 \mid$ et que l'on suppose $n_1 \geq n_2$ alors $d = n_1 - n_2 \geq 0$. La variable bivalente x_i est égale à -1 si le sommet v_i appartient au sous-ensemble V_1 et $+1$ s'il appartient au sous-ensemble V_2.

A partir de cette formulation particulière, Karisch et al. définissent une relaxation semidéfinie via l'introduction d'une nouvelle variable $X = xx^t$:

$$(BIS_{SDP}) \ : \min \left\{ \frac{1}{4} TrLX : Diag(X) = e, e^t X e = d^2, X \succeq 0 \right\}$$

Afin d'obtenir une relaxation plus fine, des inéquations spécifiques aux problèmes en (-1,1) peuvent être ajoutées à (BIS_{SDP}) : les *inégalités triangulaires*.

$$x_{ij} + x_{ik} + x_{jk} \geq -1 \tag{6.7}$$

$$x_{ij} - x_{ik} - x_{jk} \geq -1 \tag{6.8}$$

$$-x_{ij} + x_{ik} - x_{jk} \geq -1 \tag{6.9}$$

$$-x_{ij} - x_{ik} + x_{jk} \geq -1 \tag{6.10}$$

et puisque $\mathcal{O}(n^3)$ inégalités peuvent être violées, les inégalités triangulaires sont réécrites en supposant que l'on n'en considère qu'une partie (par exemple m_b). Ainsi, (6.7) à (6.10) peuvent se simplifier par :

$$B(X) + b \geq 0$$

où B est un opérateur linéaire défini dans $\mathbb{R}^{n \times n}$ et à valeurs dans $\mathbb{R}^{m_b}$ représentant les inégalités triangulaires de type (6.7) - (6.10). Enfin, $b \in \mathbb{R}^{m_b}$ avec $b_i = 1$.

Ainsi, une relaxation semidéfinie plus fine et incluant quelques inégalités triangulaires peut être définie :

$$(P) \ : \min \left\{ \frac{1}{4} TrLX : Diag(X) = e, e^t X e = d^2, B(X) + b \geq 0, X \succeq 0 \right\}$$

C'est à partir de cette relaxation que KRC développent leur algorithme de branch-and-bound. Pour chaque nœud de l'arborescence de recherche, la séparation considère le fait que deux sommets v_p et v_q appartiennent ou non au même ensemble. Ainsi, pour chacun des deux cas, le problème (P) est reformulé en un nouveau problème de dimension inférieure (si v_p et v_q sont dans le même sous-ensemble) ou avec une contrainte d'égalité supplémentaire (si v_p et v_q ne sont pas dans le même sous-ensemble) moyennant une redéfinition de la matrice L. En ce qui concerne l'évaluation en chaque nœud, KRC développent un algorithme primal-dual basé sur les points intérieurs.

Comparaison avec la méthode KRC testée sur des instances de Brunetta, Conforti et Rinaldi

Karisch, Rendl and Clausen testent leur approche sur une librairie créée par Brunetta, Conforti et Rinaldi (BCR) [30]. Nous avons choisi également de résoudre ces instances, tirées de l'adresse internet

ftp ://ftp.math.unipd.it/pub.Misc.equicut

et de comparer nos résultats avec ceux de KRC. Les instances correspondent au problème de l'équipartition ($p = \lfloor \frac{n}{2} \rfloor$) et sont divisées en 4 classes :

- *Instances Aléatoires* (Tableau 6.22) où la densité des graphes est fixée avant de créer des graphes dont le poids des arêtes (entier) est généré aléatoirement, uniformément dans l'intervalle $[1, 10]$.
- *Grilles Toroïdales* (Tableau 6.23), notées $'h \times kt'$. Elles représentent une grille toroïdale pondérée de dimension $h \times k$ dont les arêtes sont générées uniformément dans l'intervalle $[1, 10]$, comme précédemment. Ces graphes contiennent hk sommets et $2hk$ arêtes.
- *Grilles Mixtes* (Tableau 6.24) sont des graphes complets, notés $'h \times km'$. Les arêtes de la grille planaire de dimension $h \times k$ ont des poids aléatoires uniformes dans $[1, 100]$, et tous les poids des arêtes restantes sont uniformément générés dans l'intervalle $[1, 10]$.
- *Instances avec Poids Négatifs* (Tableau 6.25). Ils sont générés de la même manière que les instances de la première classe excepté que les poids des arêtes sont contenus dans $[-10, -1] \cup [1, 10]$.

Nous avons réalisé les expérimentations sur un ordinateur estimé 15 fois plus rapide que celui de Karisch, Rendl et Clausen, qui est un HP 9000/735.

Légende des tableaux 6.22 à 6.25 :

- CPU_{KRC}, le temps CPU requis par la méthode KRC sur leur machine. Les autres noms de colonnes sont définis pour les tableaux précédents.

TAB. 6.22 – Equipartition sur des graphes alétoires de la librairie BCR

pb	*n*	*d*(%)	*opt*	CPU_{KRC}	CPU_{QCR}	$borne_{QCR}$	gap_{QCR}(%)
v0.90	20	10	21	1"	0.01"	18.33	12.7
v0.00	20	100	401	1"	0.02"	392	2.2
t0.90	30	10	24	1"	0.01"	20.7	13.75
t0.50	30	50	397	22"	0.11"	370	6.8
t0.00	30	100	900	6"	0.09"	877.94	2.45
q0.90	40	10	63	4"	0.16"	50.72	19.4
q0.80	40	20	199	1'09"	0.54"	168.74	15.2
q0.30	40	70	1056	1'02"	0.98"	1016.35	17.9
q0.20	40	80	1238	25"	0.52"	1202.74	27
q0.10	40	90	1425	41"	0.64"	1387	2.67
q0.00	40	100	1606	4"	0.12"	1578.18	1.7
c0.90	50	10	122	10"	0.28"	102.06	16.3
c0.80	50	20	368	3'04"	2.66"	328.10	10.8
c0.70	50	30	603	4'02"	3.4"	555.9	7.8
c0.30	50	70	1658	2'44"	3.35"	1593.19	3.9
c0.10	50	90	2226	2'39"	2.03"	2166.95	2.65
c0.00	50	100	2520	2'20"	1.24"	2472.68	1.88
c2.90	52	10	123	12"	0.75"	96.58	21.4
c4.90	54	10	160	1'39"	1.28"	135.28	15.45
c6.90	56	10	177	30"	0.96"	149.41	15.6
c8.90	58	10	226	8'46"	23.61"	185.60	17.8
s0.90	60	10	238	4'57"	5.13"	200.82	15
	Valeurs moyennes			**1'35"**	**2"**		**11.4**

TAB. 6.23 – Equipartition des grilles toroïdales de la librairie BCR

pb	*n*	*d*(%)	*opt*	CPU_{KRC}	CPU_{QCR}	$borne_{QCR}$	gap_{QCR}(%)
4x5t	20	21	28	1"	0.01"	24.46	12.6
6x5t	30	14	31	3"	0.02"	22.65	26.9
8x5t	40	10	33	6"	0.09"	20.27	38.5
12x2t	42	10	9	5"	0.05"	3.008	66.5
23x2t	46	9	9	2'05"	0.2"	1.98	78
4x12t	48	9	24	17"	0.14"	10.99	54.2
5x10t	50	8	33	6"	0.25"	16.52	50
10x6t	60	7	42	5'50"	2.22"	19.52	53.5
7x10t	70	6	45	9'32"	3.69"	20.57	54.2
10x8t	80	5	43	15'44"	2.29"	23.51	47.7
	Valeurs moyennes			**3'22"**	**0.8"**		**48.21**

TAB. 6.24 – Equipartition des grilles mixtes de la librairie BCR

pb	*n*	*d*(%)	*opt*	CPU_{KRC}	CPU_{QCR}	$borne_{QCR}$	gap_{QCR}(%)
2x10m	20	100	118	1"	0.01"	112	5.1
6x5m	30	100	270	1"	0.03"	258.4	4.3
2x17m	34	100	316	29"	0.04"	294.98	6.6
10x4m	40	100	436	2"	0.04"	424	2.7
5x10m	50	100	670	2"	0.01"	654.126	2.4
4x13m	52	100	721	34"	0.23"	694.86	3.6
13x4m	52	100	721	34"	0.22"	694.82	3.6
9x6m	54	100	792	12"	0.48"	760.53	3.97
10x6m	60	100	954	8"	0.22"	932.29	2.3
10x7m	70	100	1288	14"	0.81"	1259.1	2.2
	Valeurs moyennes			**13.7"**	**0.2"**		**3.7**

TAB. 6.25 – Equipartition des graphes avec poids négatifs de la librairie BCR

pb	*n*	*d*(%)	*opt*	CPU_{KRC}	CPU_{QCR}	$borne_{QCR}$	gap_{QCR}(%)
t0.n.10	30	90	-301	7"	0.18"	-351.96	16.9
t0.n.00	30	100	-337	3"	0.09"	-382.63	11.9
q0.n.70	40	30	-298	50"	0.98"	-343.36	15.2
q0.n.50	40	40	-389	55"	0.37"	-435.46	11.9
q0.n.40	40	30	-450	6"	0.12"	-482.29	7.1
q0.n.00	40	100	-471	1'06"	0.56"	-544.37	15.58
c0.n.00	50	100	-829	4'17"	10.81"	-953.24	15.6
s0.n.80	60	20	-465	40"	3.34"	-513.65	10.46
o0.n.80	80	20	-690	31'48"	4'18"	-780.44	13.1
	Valeurs moyennes			**4'25"**	**30"**		**13.1**

La méthode QCR permet de résoudre en moyenne 50 fois plus vite toutes les instances considérées. Pour chaque classe, le temps CPU requis pour trouver la solution optimale est fortement amélioré comparé aux résultats de KRC : CPU_{QCR} peut être jusqu'à 30 fois plus petit que CPU_{KRC} si l'on tient compte des différences de machines. Plus précisément, en moyenne, tous

les temps de résolution sont inférieurs à 4" et 37 instances sur 51 sont résolues en moins d'une seconde (49 en moins de 5 secondes). Le plus long temps de résolution est d'environ 4'18" pour les plus grandes instances avec poids négatif sur les arêtes ($n = 80$). Le temps de résolution de la méthode KRC pour cette instance n'est en outre pas amélioré. Pour les autres graphes, si nous prenons toujours en compte les différences d'ordinateurs, notre méthode est entre 3 à 16 fois plus rapide. Les classes les plus faciles à résoudre sont les grilles mixtes et les grilles toroidales où les plus longs temps de résolution sont de 3.69" et 0.81" respectivement. Malgré les temps de résolution faible, le saut d'intégrité pour les graphes toroidaux est assez élevé, ce qui est dû aux petites densités des graphes considérés.

Comparaison avec les résultats de KRC obtenus sur leurs instances aléatoires

Pour finir, nous appliquons notre méthode sur des instances générées aléatoirement par Karisch, Rendl et Clausen. Les graphes sont pondérés avec une probabilité uniforme de $\frac{1}{2}$ sur le poids des arêtes. Les instances ont un nombre de sommets variant de 36 à 84 et les valeurs de p sont $\lfloor \frac{n}{2} \rfloor$, $\lfloor \frac{3n}{4} \rfloor$, $\lfloor \frac{7n}{12} \rfloor$ et $\lfloor \frac{13n}{24} + \frac{1}{2} \rfloor$. Le tableau 6.26 présente les différents résultats.

TAB. 6.26 – Bipartition sur des graphes générés aléatoirement

graph	n	p	*opt*	CPU_{KRC}	CPU_{QCR}	$borne_{QCR}$	$gap_{QCR}(\%)$	$\#nœuds_{QCR}$
ex36a	36	$\lfloor \frac{n}{2} \rfloor$	117	3"	0.22"	111.76	4.5	894
ex60a	60		367	5'	11.39"	354.45	3.4	26517
ex84a	84		742	2h02'55"	40'06"	716.36	3.5	2772955
ex36a	36	$\lfloor \frac{3n}{4} \rfloor$	85	31"	0.13"	78.41	7.8	584
ex60a	60		268	9'56"	8.95"	252.02	5.96	24569
ex84a	84		548	2h20'25"	12'24"	521.33	4.9	1193153
ex36a	36	$\lfloor \frac{7n}{12} \rfloor$	112	10"	0.21"	106.43	5	819
ex60a	60		351	5'38"	3.56"	340.64	2.9	8013
ex84a	84		721	9h16'51"	2h17'24"	691.56	4.08	9015554
ex36a	36	$\lfloor \frac{13n}{24} + \frac{1}{2} \rfloor$	114	3"	0.13"	108.88	4.5	794
ex60a	60		360	5'18"	8.72"	348.88	3.1	19349
ex84a	84		735	5h03'53"	1h52'48"	707.33	3.8	6437203
	Valeurs moyennes			**1h35'24"**	**25'16"**		**4.45**	

Comme Karisch et al. l'ont notifié, les instances générées aléatoirement de ce type représentent la classe la plus dure. On remarque aussi que les valeurs de p sont, pour chaque instance, très proches de $\frac{n}{2}$. Ainsi, on se rapproche du problème de l'équipartition qui est le problème le plus difficile.

Toutes les instances sont résolues à l'optimum. Plus précisément, les temps de résolution pour les graphes à 36 et 60 sommets sont très petits (moins de 0.22" quand $n = 36$ et moins de 11.39" quand $n = 60$). Les sauts d'intégrité sont également faibles (moins de 7.8% quand $n = 36$ et 6% quand $n = 60$). Quand $n = 84$, ils sont inférieurs à 4.9%. Cependant, quelques minutes et même quelques heures sont requises (entre 12' et 2h17').

Maintenant, si nous comparons nos résultats avec ceux de KRC (et évidemment en tenant toujours compte des différences de machines), notre méthode n'est pas toujours meilleure. Par exemple, pour le graphe de 84 sommets et pour une valeur de p égale à $\lfloor \frac{7n}{12} \rfloor$, la valeur optimale est seulement déterminée 4,5 fois plus rapidement qu'avec la méthode de KRC alors que leur ordinateur est 15 fois plus lent. Cependant, nous pouvons dire que Karisch et al. appliquent une méthode spécifique au problème de bipartition de graphe alors que l'approche QCR est très générale : nous n'avons pas pris en

compte les caractéristiques spécifiques au problème (BP).

Dans cette section, nous avons présenté une application de la méthode QCR au problème de bipartition de graphe. Nous avons résolu 300 instances de différentes classes : graphes non pondérés, (positivement ou non positivement) pondérés, grilles. Nous avons également considéré différentes valeurs de densités et de taille de partition. Nos expérimentations montrent que le saut d'intégrité est généralement inférieur à 20% et diminue avec les densités : pour $d = 25\%$, il est en moyenne égal à 9% ; pour $d = 50\%$, le saut est de 4% et pour $d = 75\%$ 3%. Le plus mauvais saut d'intégrité est obtenu pour les grilles toroïdales (petites densités des graphes).

L'approche QCR résout efficacement le problème de bipartition sur des graphes généraux avec 90-100 sommets mais on remarque que, quelle que soit la taille du graphe, le problème de l'équipartition est le plus difficile (spécialement pour $n = 100$). Si l'on compare nos résultats avec les méthodes existantes ([122], [30], [55], [88], [39],[40], [111]) qui se limitent à la résolution de graphes avec environ 60 sommets, nous pouvons dire que QCR est une méthode très efficace.

Si l'on se compare maintenant à la méthode KRC, tout comme Karisch et al., nous sommes capables de résoudre des graphes avec 90-100 sommets et nous obtenons de bonnes relaxations pour chaque classe de problèmes. Cependant, leurs temps de résolution sont quelquefois meilleurs que les notres (environ 7 instances sur 63) . Ceci est sans doute dû en partie au fait qu'ils utilisent les spécificités du problème de bipartition contrairement à QCR.

6.4 Le problème de la minimisation d'échange d'outils

On suppose que l'on dispose de n pièces à usiner sur une machine contenant un magasin d'outils de capacité limitée. Chaque pièce a besoin d'un ensemble d'outils pour être usinée, ce qui signifie qu'au moment où elle passe sur la machine, ces outils doivent être présents dans le magasin. En raison de la capacité limitée du magasin, il est parfois nécessaire de procéder à des échanges d'outils entre l'usinage de deux pièces. Le problème de la minimisation d'échange d'outils est de fournir un ordre de passage des pièces sur

la machine et l'évolution de l'état du magasin à chaque étape, de manière à minimiser le nombre total d'échanges d'outils.

Quelques contraintes doivent être prises en compte :
- Une seule machine usine les pièces
- La machine utilise exclusivement les outils présents dans le magasin
- La machine ne peut entamer une tâche que si tous les outils nécessaires à cette tâche sont présents dans le magasin
- Si un outil est manquant, il faut le mettre en magasin quitte à retirer un autre outil, inutile pour cette tâche.

Les données sont les suivantes :
- c : la capacité du magasin de la machine
- n : le nombre de pièces à usiner
- m : le nombre total d'outils
- A : la matrice représentant les outils nécessaires à chaque pièce

Les coefficients de la matrice A sont notés a_{ji}, $\forall j \in \{1, \ldots, m\}$, $\forall i \in \{1, \ldots, n\}$ et $a_{ji} = 1$ si et seulement si l'outil j est nécessaire à la tâche i.

Prenons par exemple une machine dont le magasin est de capacité maximale 4. On souhaite usiner 5 pièces qui utilisent 8 outils différents. La matrice A est donnée par :

$$\begin{array}{c} \\ \\ \\ outils \rightarrow \end{array} \begin{array}{c} \text{pièces} \\ \downarrow \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

La valeur optimale est égale à 4. Une des solutions optimales est donnée dans le tableau suivant :

périodes	1	2	3	4	5
pièces	1	4	3	2	5
outils	**1**	**1**	**2**	2	**4**
	2	**2**	**4**	**3**	5
	4	4	**5**	4	**6**
	7	7	7	**5**	**8**
échanges d'outils	0	0	1	1	2

Chaque colonne représente une période et une pièce. On indique, pour chaque pièce, les outils présents dans le magasin au moment où elle est usinée. Les chiffres en gras correspondent aux outils nécessaires à l'usinage de la pièce. Les autres outils sont donc présents dans le magasin parce qu'ils ont déjà servi, et que les retirer du magasin "coûte" plus cher que de les y laisser.

Le problème de la minimisation du nombre d'échanges d'outils est un problème NP-difficile dès que la capacité du magasin est supérieure ou égale à 2 [43]. Avec des méthodes exactes, des instances comprenant jusqu'à 25 tâches sont résolues par Laporte et al. [96]. Les modélisations les plus courantes sont linéaires en variables entières sous contraintes linéaires ([96], [132], [43]). Celle de Tang et Denardo [132] sera présentée plus en détail dans la section suivante : ils arrivent à résoudre polynomialement un cas particulier du problème où la séquence d'usinage des pièces est connue.

La section 6.4.1 présente la formulation linéaire de Tang et Denardo. Ensuite, dans le but d'appliquer la méthode QCR (section 6.4.4) ainsi qu'une méthode de convexification directe (section 6.4.3), le problème de minimisation d'échange d'outils est modélisé sous forme d'un programme quadratique en variables 0-1 sous contraintes linéaires (section 6.4.2).

Ce travail a été réalisé par Aurélie Le Maître de février à août 2005, pendant son stage de Master2 recherche, au laboratoire CEDRIC du CNAM, sous ma direction ainsi que celle d'Alain Billionnet et Sourour Elloumi.

6.4.1 Formulation de Tang et Denardo

Pour leur modélisation, Tang et Denardo utilisent trois familles de variables bivalentes :

- $x_{jk} = 1$ si l'outil j est présent dans le magasin à l'étape k
- $y_{ik} = 1$ si la pièce i est usinée à l'étape k
- $z_{jk} = 1$ si l'outil j est ajouté dans le magasin à l'étape k. Cette variable correspond donc à un changement d'outils.

Le programme linéaire en 0-1 sous contraintes linéaires est le suivant :

$$\min \quad \sum_{j=1}^{m}\sum_{k=2}^{m} z_{jk}$$

s.c.

$$\sum_{j=1}^{m} x_{jk} \leq c \qquad \forall k \in \{1,\ldots,n\} \tag{6.11}$$

$$\sum_{i=1}^{n} y_{ik} = 1 \qquad \forall k \in \{1,\ldots,n\} \tag{6.12}$$

$$\sum_{k=1}^{n} y_{ik} = 1 \qquad \forall i \in \{1,\ldots,n\} \tag{6.13}$$

$$\sum_{i=1}^{n} a_{ji} y_{ik} \leq x_{jk} \qquad \forall k \in \{1,\ldots,n\}, \forall j \in \{1,\ldots,m\} \tag{6.14}$$

$$x_{jk} - x_{j,k-1} \leq z_{jk} \qquad \forall k \in \{1,\ldots,n\}, \forall j \in \{1,\ldots,m\} \tag{6.15}$$

$$x_{jk} \in \{0,1\} \qquad \forall j \in \{1,\ldots,m\}, \forall k \in \{1,\ldots,n\}$$

$$y_{ik} \in \{0,1\} \qquad \forall i \in \{1,\ldots,n\}, \forall k \in \{1,\ldots,n\}$$

$$z_{jk} \in \{0,1\} \qquad \forall j \in \{1,\ldots,m\}, \forall k \in \{1,\ldots,n\}$$

La contrainte (6.11) exprime le fait qu'à chaque étape, le nombre d'outils dans le magasin ne doit pas dépasser sa capacité c. Les contraintes (6.12) et (6.13) sont des contraintes d'affectation des pièces : à chaque étape, une seule pièce est usinée une seule fois. La contrainte (6.14) traduit le fait que si l'on usine une pièce i à une étape k ($y_{ik} = 1$) et que cette pièce a besoin de l'outil j ($a_{ji} = 1$), alors l'outil j devra être présent dans le magasin à cette étape (x_{jk}). La contrainte (6.15) permet de définir la variable z : on compte un échange d'outils lorsque l'outil est présent à une étape k et qu'il n'était pas présent à l'étape précédente.

Les expériences menées par Tang et Denardo à partir de cette formulation ont donné des résultats plutôt décevants. En effet, la valeur de la relaxation continue est toujours égale à 0, la solution optimale de la relaxation continue étant :

$$\begin{cases} y_{ik} = \frac{1}{n} & \forall i \in \{1,\dots,n\}, \forall k \in \{1,\dots,n\} \\ x_{jk} = \frac{n_j}{n} & \forall j \in \{1,\dots,m\}, \forall k \in \{1,\dots,n\} \\ z_{jk} = 0 & \forall j \in \{1,\dots,m\} \forall k \in \{1,\dots,n\} \end{cases}$$

où n_j représente le nombre de pièces utilisant l'outil j. Cette valeur de relaxation ne permet donc pas de développer un algorithme de résolution par un branch-and-bound efficace.

Nous allons donc tenter de modéliser ce problème avec une fonction objectif quadratique, en partant de la formulation de Tang et Denardo, de manière à appliquer la méthode QCR.

6.4.2 Une formulation quadratique

On peut voir la formulation de Tang et Denardo comme une linéarisation. On rappelle que la variable z_{kj} représente l'insertion d'un outil j dans le magasin à l'étape k. Or, z_{kj} peut s'exprimer directement à partir de la variable x car $z_{kj} = x_{jk}(1 - x_{j,k-1})$. En effet, $x_{jk}(1 - x_{j,k-1})$ vaut 1 quand l'outil j est présent à l'étape k mais pas à l'étape $k-1$ ce qui implique que l'outil a bien été ajouté à l'étape k. Ainsi, dénombrer l'ensemble des outils ajoutés dans le magasin à chaque étape revient à compter le nombre total de changements d'outils.

Soient n le nombre de pièces et m le nombre d'outils. Les variables y_{ik} et x_{jk} sont les mêmes que celles de la formulation de Tang et Denardo. Le programme quadratique en variables 0-1 s'écrit :

$$
\begin{aligned}
(EC) \quad \min \quad & \sum_{j=1}^{m}\sum_{k=2}^{m} x_{jk}(1 - x_{j,k-1}) & & \\
s.c. \quad & \sum_{j=1}^{m} x_{jk} = c & \forall k \in \{1,\dots,n\} & \quad (6.16)\\
& \sum_{i=1}^{n} y_{ik} = 1 & \forall k \in \{1,\dots,n\} & \\
& \sum_{k=1}^{n} y_{ik} = 1 & \forall i \in \{1,\dots,n\} & \\
& \sum_{i=1}^{n} a_{ji} y_{ik} \leq x_{jk} & \forall k \in \{1,\dots,n\}, \forall j \in \{1,\dots,m\} & \\
& x_{jk} \in \{0,1\} & \forall j \in \{1,\dots,m\}, \forall k \in \{1,\dots,n\} & \\
& y_{ik} \in \{0,1\} & \forall i \in \{1,\dots,n\}, \forall k \in \{1,\dots,n\} &
\end{aligned}
$$

La contrainte (6.16) assure que la capacité du magasin est bien respectée. Pour des raisons pratiques, la contrainte (6.11) a été transformée en une contrainte d'égalité, ce qui ne limite pas la résolution du problème : on peut toujours laisser dans le magasin des outils non nécessaires, sans pour cela augmenter le coût total. Les autres contraintes sont les mêmes que celles de la modélisation de Tang et Denardo.

C'est à partir de (EC) que nous allons appliquer la méthode QCR mais aussi, dans un premier temps, une convexification directe consistant à réécrire la fonction économique.

6.4.3 Une méthode de convexification directe

En utilisant de manière astucieuse le caractère binaire des variables et la contrainte de capacité saturée, on parvient à exprimer l'objectif sous la forme d'une fonction convexe :

$$q_{conv}(x) = \sum_{j=1}^{m}\sum_{k=2}^{m} \frac{1}{2}\left(x_{jk} - x_{j,(k-1)}\right)^2$$

Cette fonction est bien équivalente à celle de départ puisque :

$$\begin{aligned} q_{conv}(x) &= \sum_{j=1}^{m}\sum_{k=2}^{m} - x_{jk}x_{j,k-1} + \tfrac{1}{2}\sum_{j=1}^{m}\sum_{k=2}^{m} x_{jk}^2 + \tfrac{1}{2}\sum_{j=1}^{m}\sum_{k=2}^{m} x_{j,k-1}^2 \\ &= \sum_{j=1}^{m}\sum_{k=2}^{m} - x_{jk}x_{j,k-1} + \tfrac{1}{2}\sum_{j=1}^{m}\sum_{k=2}^{m} x_{jk} + \tfrac{1}{2}\sum_{j=1}^{m}\sum_{k=2}^{m} x_{j,k-1} \end{aligned}$$

Or, d'après la contrainte (6.16), à chacune des $m-1$ étapes du processus de fabrication, exactement c outils sont présents dans le magasin, d'où :

$$\sum_{j=1}^{m}\sum_{k=2}^{m} x_{jk} = c(m-1)$$

De la même façon, $\sum_{j=1}^{m}\sum_{k=2}^{m} x_{j,(k-1)} = c(m-1)$ et donc :

$$\begin{aligned} q_{conv}(x) &= \sum_{j=1}^{m}\sum_{k=2}^{m} - x_{jk}x_{j,k-1} + c(m-1) \\ &= \sum_{j=1}^{m}\sum_{k=2}^{m} - x_{jk}x_{j,k-1} + \sum_{j=1}^{m}\sum_{k=2}^{m} x_{jk} \\ &= q(x) \end{aligned}$$

En remplaçant $q(x)$ par $q_{conv}(x)$, on peut résoudre directement le problème par un solveur quadratique convexe.

Les instances utilisées pour la résolution nous ont été fournies par les auteurs de [96]. Les valeurs des solutions optimales pour ces instances particulières n'étant pas connues, les résultats obtenus sont comparés avec la valeur de la solution admissible fournie par l'algorithme gourmand de Laporte, Salazar et Semet.

Le tableau suivant présente les résultats expérimentaux, qui ont été réalisés sur un processeur Pentium II, avec un ordinateur disposant de 128Mo de RAM.

Légende du tableau 6.4.3 :

- *sol. continue*, les valeurs optimales de la relaxation continue du nouveau problème convexe

- *sol. gourmand*, les valeurs admissibles obtenues par l'algorithme gourmand développé par Laporte, Salazar et Semet [96]
- *adm. entière* : dans les tests suivants, la limite de temps a été fixée à 5 minutes. Ainsi, la solution entière obtenue est une solution admissible mais non nécessairement optimale.
- Le signe '-' correspond aux instances n'ayant pas de solution admissible entière en moins de 5 minutes

pièces	*outils*	*capacité*	*sol. continue*	*adm. entière*	*sol. gourmand*
10	10	4	0	10	10
15	15	5	0	28	5
15	20	6	0	23	21
15	25	15	0	36	34
20	15	10	0	30	24
20	20	10	0	-	42
20	25	10	0	49	51
25	25	20	0	36	38

On constate que pour l'instance $(20, 20, 10)$, aucune solution entière n'a été trouvée en moins de 5 minutes. Pour les deux dernières instances, on parvient à obtenir une meilleure solution admissible que l'algorithme de Laporte et Salazar.

Maintenant, en ce qui concerne les solutions entières admissibles obtenues, on remarque que la valeur de la relaxation continue est toujours égale à 0, ce qui peut se montrer facilement. En effet, l'objectif est positif ou nul pour toutes valeurs de x ou y. De plus, on peut exhiber une solution pour laquelle l'objectif associé est nul : il s'agit de la solution fournie par la relaxation continue du programme linéaire de Tang et Denardo :

$$\begin{cases} y^*_{ik} = \frac{1}{n} & \forall i \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, n\} \\ x^*_{jk} = \frac{n_j}{n} & \forall j \in \{1, \ldots, m\}, \forall k \in \{A, \ldots, n\} \end{cases}$$

où n_j représente le nombre de pièces utilisant l'outil j.

La relaxation continue donnant toujours une valeur optimale nulle, cette méthode peut se révéler très inefficace pour la résolution exacte. Essayons maintenant d'appliquer la méthode QCR au problème (EC).

6.4.4 Application de la méthode QCR : les limites des logiciels *SB* et *CSDP*

La reformulation QCR requiert, dans la Phase I, la résolution d'une relaxation SDP où chaque contrainte d'égalité est multipliée par chaque variable. Le problème (EC) contient deux ensembles de variables de décisions x_{jk} et y_{ik} ayant deux significations différentes. En effet, on rappelle que x_{jk} prend la valeur 1 si l'outil k est dans le magasin quand la pièce j est usinée et y_{ik} prend la valeur 1 si la pièce i est usinée juste avant la pièce k.

On doit résoudre le programme SDP où les contraintes en x et y sont multipliées par chaque variable x et y.

$$
\begin{aligned}
(SDEC) \quad \min \quad & \sum_{j=1}^{m}\sum_{k=2}^{m} X_{jkj(k-1)} + \sum_{j=1}^{m}\sum_{k=2}^{m} x_{jk} \\
\text{s.c.} \quad & X_{jkjk} = x_{jk} && \forall j \in \{1,\dots,m\}, \forall k \in \{1,\dots,n\} && (6.17)\\
& Y_{ilil} = y_{il} && \forall i \in \{1,\dots,n\}, \forall l \in \{1,\dots,n\} && (6.18)\\
& \sum_{j=1}^{m} x_{jk} = c && \forall k \in \{1,\dots,n\} \\
& -cx_{il} + \sum_{j=1}^{m} X_{jkil} = 0 && \forall k,l \in \{1,\dots,n\}, \forall i \in \{1,\dots,m\} && (6.19)\\
& \sum_{i=1}^{n} y_{ik} = 1 && \forall k \in \{1,\dots,n\} \\
& \sum_{k=1}^{n} y_{ik} = 1 && \forall i \in \{1,\dots,n\} \\
& \sum_{i=1}^{n} a_{ji} y_{ik} \leq x_{jk} && \forall k \in \{1,\dots,n\}, \forall j \in \{1,\dots,m\} \\
& \sum_{i=1}^{n} Y_{ikhl} - y_{hl} = 0 && \forall k,l \in \{1,\dots,n\}, \forall i \in \{1,\dots,m\} && (6.20)\\
& \sum_{k=1}^{n} Y_{ikhl} = 0 && \forall h,l \in \{1,\dots,n\}, \forall i \in \{1,\dots,n\} && (6.21)\\
& \sum_{i=1}^{n} Z_{ikjl} - x_{jl} = 0 && \forall j \in \{1,\dots,m\}, \forall k,l \in \{1,\dots,n\} && (6.22)\\
& \sum_{k=1}^{n} Z_{ikjl} - x_{jl} = 0 && \forall j \in \{1,\dots,m\}, \forall i,l \in \{1,\dots,n\} && (6.23)\\
& \sum_{j=1}^{m} Z_{jkhl} - c.y_{jhl} = 0 && \forall h,l,k \in \{1,\dots,n\} && (6.24)\\
& Z \succeq 0
\end{aligned}
$$

Si l'on note z le vecteur constitué des variables x_{jk} suivis des variables y_{ik} :

$z = (x_{11}, x_{12}, \ldots, x_{1n}, x_{21}, \ldots, x_{mn}, y_{11}, y_{12}, \ldots, y_{1n}, y_{21}, \ldots, y_{nn})$
alors Z a la forme suivante, où p est la dimension de la matrice (soit $nm + nn$) :

$$Z = \begin{pmatrix} 1 & z_1 & z_2 & . & . & . & z_p \\ z_1 & z_1.z_1 & z_1.z_2 & . & . & . & z_1.z_p \\ z_2 & z_2.z_1 & z_2.z_2 & . & . & . & z_2, z_p \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ z_p & z_p.z_1 & z_p z_2 & . & . & . & z_p.z_p \end{pmatrix}$$

Une fois le programme $(SDEC)$ résolu, les variables duales associées aux contraintes (6.17), (6.18), (6.19), (6.20), (6.21), (6.22), (6.23), (6.24), correspondent respectivement aux paramètres u^*, v^*, α^*, β^*, γ^*, ϕ^*, ψ^*, ω^*, qui sont ensuite intégrés dans la fonction objectif de (EC) :

$$\sum_{j=1}^{m}\sum_{k=2}^{n} x_{jk}(1 - x_{j,k-1}) + \sum_{i=1}^{m}\sum_{l=1}^{n}\sum_{k=1}^{n} x_{il}\alpha^*_{kil}(\sum_{j=1}^{m} x_{jk} - c) + \sum_{j=1}^{m}\sum_{k=1}^{m} u^*_{jk}(x^2_{jk} - x_{jk})$$
$$+ \sum_{j=1}^{m}\sum_{k=1}^{n} v^*_{ik}((y^2_{ik} - y_{ik}) + \sum_{h=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n} y_{hl}\beta^*_{hlk}(\sum_{i=1}^{n} y_{ik} - 1) + \sum_{h=1}^{n}\sum_{l=1}^{n}\sum_{i=1}^{n} y_{hl}\gamma^*_{hli}(\sum_{k=1}^{n} y_{ik} - 1)$$
$$+ \sum_{j=1}^{m}\sum_{l=1}^{n}\sum_{k=1}^{n} x_{jl}\phi^*_{jlk}(\sum_{i=1}^{n} y_{ik} - 1) + \sum_{j=1}^{m}\sum_{l=1}^{n}\sum_{i=1}^{n} x_{jl}\psi^*_{jli}(\sum_{k=1}^{n} y_{ik} - 1)$$
$$+ \sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n} y_{il}\omega^*_{ilk}(\sum_{j=1}^{m} x_{jk} - c)$$

Le tableau 6.4.4 présente les différents résultats obtenus. Le programme $(SDEC)$ a été résolu avec le logiciel SB. De nombreuses difficultés ont été rencontrées lors de cette résolution. En effet, puisque la taille des instances à résoudre est grande, le logiciel SB ne permet pas de trouver une valeur optimale en moins de 10 minutes. Ainsi, afin de pouvoir être compétitif avec les meilleurs résultats trouvés jusqu'à maintenant par Tang et Denardo ou encore Salazar et Semet, on a limité le temps de calcul de SB de 10' à 2h50 selon les instances. En effet, pour les plus petites instances, à partir de 10', les valeurs duales optimales calculées par SB permettent de rendre le nouvel objectif convexe alors que pour d'autres instances, ce n'est malheureusement pas le cas.

Légende du tableau 6.4.4 :

- cpu_{SB}, temps de résolution du logiciel SB pour trouver la valeur admissible adm_{SB}
- $borne_{cplex}$, valeur optimale de la relaxation continue de (CE)
- adm_{cplex}, valeur entière admissible trouvée en 5' par le branch-and-bound de CPLEX
- *sol. gourmand*, les valeurs admissibles obtenues par l'algorithme gourmand développé par Laporte, Salazar et Semet [96]
- Le signe '-' signifie qu'aucune solution entière n'a pu être déterminée en moins de 5'.
- Le signe '!' signifie que les valeurs duales obtenues par le logiciel SB ne permettent pas de rendre la fonction objectif reformulée par QCR convexe.

n	m	c	cpu_{SB}	adm_{SB}	$borne_{cplex}$	adm_{cplex}	$sol.gourmand$
10	10	4	17'	-0.29	-1.67	10	10
15	15	5	17'	-11.61	-3.20	29	31
15	20	6	33'	-16.55	-5.13	28	21
15	25	15	50'	-12.51	-5.35	37	34
20	15	10	50'	-40.94	-5.14	34	24
20	20	10	1h7'	-31.13	-6.39	-	32
20	25	10	1h15'	-65.03	-7.16	-	46
25	25	20	2h46'	-480.08	!	-	38

On observe ici deux phénomènes assez gênants : il n'y a pas égalité entre la valeur optimale continue du problème (CE) et la valeur optimale du programme $(SDEC)$ et pire, cette dernière est négative.

L'inégalité entre les bornes s'explique par le fait que l'on arrête le logiciel SB avant qu'il ne trouve la solution optimale. On observe que même après plus de 2h30' de résolution, les valeurs duales admissibles de $(SDEC)$ ne permettent pas toujours de rendre l'objectif convexe. Le second phénomène (i.e. une valeur optimale du programme SDP toujours négative) s'explique également par notre arrêt forcé de la résolution par SB.

Les mêmes instances ont été résolues avec le logiciel CSDP et les temps

de calcul sont également très longs. Les approches de résolution SDP sont donc ici trop coûteuses en temps de calcul et donc difficilement exploitables pour ce problème. De ce fait, nous ne pouvons malheureusement pas être compétitifs par rapport aux méthodes existantes.

6.5 Conclusion

Sur les trois problèmes d'optimisation non convexes que nous avons considérés, QCR s'est révélée très efficace pour deux d'entre eux : le problème du k-cluster et de bipartition. En effet, pour le problème du k-cluster, nous sommes capables de résoudre des instances encore jamais résolues dans la littérature et pour le problème de bipartition de graphe, notre méthode générale permet également la résolution d'instances de grandes tailles et ses performances sont comparables à des méthodes très récentes, spécifiques à ce problème.

Malheureusement, pour le dernier problème que nous avons étudié, i.e. la minimisation d'échanges d'outils, QCR n'est pas efficace à cause de la difficulté de convergenre des logiciels SDP.

Chapitre 7

Etude expérimentale de trois problèmes d'optimisation avec un objectif convexe ou préalablement convexifié

Comme nous l'avons vu dans le chapitre précédent, toutes les applications de la méthode QCR ont été réalisées sur des problèmes quadratiques avec un objectif non convexe. La reformulation QCR a, dans ce cas, joué un double rôle : celui de convexifier l'objectif mais aussi de trouver la plus grande borne inférieure possible.

Dans les expérimentations qui vont nous intéresser dans ce chapitre, les problèmes sont déjà convexes. Nous avons donc tenté d'appliquer la méthode QCR dans le but, non plus de convexifier l'objectif, mais de savoir si elle permet d'améliorer la borne inférieure obtenue par relaxation continue. Nous verrons que les résultats sont très intéressants.

On considère dans un premier temps un problème de plus court chemin en univers incertain (section 7.1) puis un problème d'investissements (section 7.2). Ces deux problèmes combinatoires se formulent comme des programmes quadratiques convexes en variables 0-1 sous containtes linéaires. La fonction objectif étant convexe, on peut les soumettre directement et simplement à un solveur quadratique convexe. Ils vont donc être résolus par deux approches différentes :

- celle qui parait la plus naturelle, c'est-à-dire le soumettre directement aux logiciels MIQP standards.
- la méthode QCR : le problème est reformulé avant d'être résolu en entier.

Enfin, nous présentons des résultats expérimentaux sur un problème d'ordonnancement (section 7.3), qui se modèlise comme un programme en variables bivalentes et dont l'objectif n'est pas convexe. Nous l'étudions dans cette section car nous allons lui appliquer non seulement QCR et une reformulations spécifique, présentée par Skutella [129] mais aussi deux reconvexifications basées sur la reformulation de Skutella.

7.1 Le problème du plus court chemin probabiliste

On considère le problème du plus court chemin dans le cadre probabiliste. Il consiste à déterminer un chemin "optimal" d'un sommet origine v_1 à un sommet destination v_n dans un graphe orienté $G = (V, U)$. Les temps de parcours associés aux arcs sont des variables aléatoires positives dont on connait l'espérance et la variance-covariance.

Beaucoup de problèmes de cheminements dans les graphes possèdent ce caractère aléatoire. On cherchera ici le chemin dont l'espérance du temps de parcours total est inférieure à une valeur t donnée et, parmi ces chemins, celui dont la variance de la longueur est minimale. On notera les similitudes entre ce problème et les modèles classiques de gestion de portefeuille type Markovitz [107] : l'approche est identique puisque l'on souhaite respecter une espérance tout en minimisant un risque.

Données du problème :
$V = \{v_1, v_2, \ldots, v_n\}$
$U = \{u_1, u_2, \ldots, u_m\}$
t_i variable aléatoire représentant le temps de parcours de l'arc u_i, de moyenne $\bar{t_i}$ $(i, 1, \ldots, m)$
σij covariance associée aux temps de parcours des 2 arcs u_i et u_j $(i = 1, \ldots, m; j = 1, \ldots, m)$
t l'espérance du temps de parcours ne doit pas dépasser la valeur t
$\omega^+(v_i)$ ensemble des arcs ayant pour extrémité initiale le sommet v_i
$\omega^-(v_i)$ ensemble des arcs ayant pour extrémité terminale le sommet v_i

Le problème peut se formuler par le programme quadratique en variables bivalentes suivant :

$$
\begin{aligned}
(CP): \quad \min \quad & p(x) = \sum_{i=1}^{m}\sum_{j=1}^{m} \sigma_{ij} x_i x_j \\
\text{s.c.} \quad & \sum_{i=1}^{m} \bar{t_i} x_i \leq t \\
& \sum_{j:u_j \in \omega^+(v_i)} x_j - \sum_{j:u_j \in \omega^-(v_i)} x_j = 0 \quad i = 2, \ldots, n-1 \quad (7.1) \\
& \sum_{j:u_j \in \omega^+(v_1)} x_j = 1 \quad (7.2) \\
& \sum_{j:u_j \in \omega^-(v_n)} x_j = 1 \quad (7.3) \\
& x \in \{0,1\}^n
\end{aligned}
$$

où la variable bivalente x_i est égale à 1 si et seulement si l'arc u_i est retenu dans le chemin.

Les égalités (7.2) et (7.3) définissent la source et le puits du chemin. La contrainte (7.1) est la contrainte de conservation en chaque nœud.

Ici, le hessien de (CP) n'est autre que la matrice de variance-covariance, qui est, par définition, semidéfinie positive. Ainsi, le problème (CP) est un programme quadratique en 0-1 avec un objectif convexe.

Exemple 3 :

Considérons un graphe à 6 sommets et 9 arcs. Les paramètres du modèle sont estimés à partir d'un historique sur les 5 dernières périodes. Cet historique est présenté dans le tableau 7.1 (les six premières colonnes).

TAB. 7.1 – Historique et espérance des temps de parcours

arcs	temps de parcours					esp.
	1	2	3	4	5	math.
1 :(1,2)	40	60	70	100	10	56
2 :(1,4)	40	30	20	10	40	28
3 :(1,5)	10	10	20	20	20	16
4 :(2,3)	10	10	20	10	10	12
5 :(2,4)	30	20	20	20	30	20
6 :(3,6)	40	60	70	10	10	38
7 :(4,6)	10	30	20	10	30	20
8 :(5,4)	50	30	60	50	40	46
9 :(5,6)	50	30	20	10	50	32

On en déduit l'espérance du temps de parcours sur chaque arc (la dernière colonne). On peut ainsi contruire le graphe associé :

FIG. 7.1 – Graphe avec espérance mathématique du temps de parcours sur chaque arc

Sur la figure ci-dessus, le temps de parcours moyen apparaît sur chaque arc du graphe, notés u_i $\forall i = 1, \ldots, 9$.

A partir de l'historique 7.1, on déduit également la variance entre chaque paire d'arcs. Le tableau 7.2 donne la matrice de variance-covariance. Si l'on note t_{ij} le temps de parcours sur l'arc u_i à la période j, on rappelle que les coefficients de la matrice de variance-covariance sont donnés par :

$$\sigma_{ij} = \frac{1}{p} \sum_{k=1}^{p} (t_{ki} - \bar{t}_i)\,(t_{kj} - \bar{t}_j)$$

où p représente le nombre de périodes.

TAB. 7.2 – Matrice de variance-covariance

	1	2	3	4	5	6	7	8	9
1	1130	-410	30	35	-300	140	-175	130	-565
2	-410	170	-35	-20	125	-5	50	-60	230
3	30	-35	30	10	-25	-60	0	30	-40
4	35	-20	10	20	-25	80	0	35	-30
5	-300	125	-25	-25	100	-75	25	-50	175
6	140	-5	-60	80	-75	770	50	40	-70
7	-175	50	0	0	25	50	100	-75	50
8	130	-60	30	35	-50	40	-75	130	-65
9	-565	230	-40	-30	175	-70	50	-65	320

On a maintenant toutes les données du problème, que l'on peut soumettre directement à un solveur MIQP.

Le tableau 7.3 donne les résultats obtenus pour différentes valeurs de la longueur t.

TAB. 7.3 – Détails des résultats obtenus

t	Espérance du temps de parcours	Chemin optimal	Risque (variance)	Risque (Ecart-type)	Relax. continue
50	48	3,9	270	16.43	206.35
55	48	3,9	270	16.43	98.45
60	48	3,9	270	16.43	39.56
62	48	3,9	270	16.43	23.69
65	48	3,9	270	16.43	7.73
70	**48**	**3,9**	**270**	**16.43**	**0.41**
80	48	3,9	270	16.43	0.41
90	82	3,7,8	170	13.04	0.41
100	82	3,7,8	170	13.04	0.41
110	82	3,7,8	170	13.04	0.41
120	82	3,7,8	170	13.04	0.41
130	82	3,7,8	170	13.04	0.41
140	82	3,7,8	170	13.04	0.41

Par exemple, pour $t = 70$, la meilleure solution consiste à retenir le chemin formé des arcs 3 et 9. L'espérance du temps de parcours correspondant est égale à 48 avec un écart type de 16.43 environ. La valeur de la solution optimale est égale à la variance qui, dans ce cas, vaut 270. La valeur optimale de la relaxation continue est égale à 0.405. ♢

7.1.1 La reformulation QCR

Si l'on applique la reformulation QCR au problème (CP), la fonction économique convexe $p(x)$ est remplacée par :

$$\begin{aligned} p_{\alpha,u}(x) = & \sum_{i=1}^{m}\sum_{j=1}^{m}\sigma_{ij}x_ix_j + \sum_{k=1}^{n-2}\left(\sum_{i=1}^{n}\alpha_{ki}x_i\right)\left(\sum_{j:u_j\in\omega^+(v_i)}x_j - \sum_{j:u_j\in\omega^-(v_i)}x_j\right) \\ & + \left(\sum_{i=1}^{n}\alpha_{n-1,i}x_i\right)\left(\sum_{j:u_j\in\omega^+(v_1)}x_j - 1\right) + \left(\sum_{i=1}^{n}\alpha_{ni}x_i\right)\left(\sum_{j:u_j\in\omega^-(v_n)}x_j - 1\right) \\ & + \sum_{i=1}^{n}u_i(x_i^2 - x_i) \end{aligned}$$

les valeurs optimales de α et u sont calculées grâce au programme semidéfini suivant :

$$\begin{aligned} (SDCP): \quad \min \quad & \sum_{i=1}^{m}\sum_{j=1}^{m}\sigma_{ij}X_{ij} & & \\ \text{s.c.} \quad & & & \\ & \sum_{i=1}^{m}\bar{t_i}x_i \le t & & \\ & \sum_{j:u_j\in\omega^+(v_i)}x_j - \sum_{j:u_j\in\omega^-(v_i)}x_j = 0 & i = 2,\dots,n-1 & \\ & \sum_{j:u_j\in\omega^+(v_i)}X_{kj} - \sum_{j:u_j\in\omega^-(v_i)}X_{ij} = 0 & i = 2,\dots,n-1 & \leftarrow \alpha^*_{ij} \\ & & k = 1,\dots,m & \\ & \sum_{j:u_j\in\omega^+(v_1)}x_j = 1 & & \\ & \sum_{j:u_j\in\omega^+(v_1)}X_{ij} = x_i & i = 1,\dots,m & \leftarrow \alpha^*_{n-1,j} \\ & \sum_{j:u_j\in\omega^-(v_n)}x_j = 1 & & \\ & \sum_{j:u_j\in\omega^-(v_n)}X_{ij} = x_i & i = 1,\dots,m & \leftarrow \alpha^*_{nj} \\ & X_{ii} = x_i & i = 1,\dots,n & \leftarrow u^*_i \\ & x\in\{0,1\}^n, X\in S_n & & \end{aligned}$$

et on résout le nouveau problème reformulé, toujours convexe :

$$
\begin{array}{lll}
(CP): & \min & p_{\alpha^*,u^*}(x) \\
& \text{s.c.} & \sum_{i=1}^{m} \bar{t}_i x_i \leq t \\
& & \sum_{j:u_j \in \omega^+(v_i)} x_j - \sum_{j:u_j \in \omega^-(v_i)} x_j = 0 \quad i = 2,\ldots,n-1 \\
& & \sum_{j:u_j \in \omega^+(v_1)} x_j = 1 \\
& & \sum_{j:u_j \in \omega^-(v_n)} x_j = 1 \\
& & x \in \{0,1\}^n
\end{array}
$$

Exemple 3 :

Revenons à l'exemple présenté en début de section. Si l'on applique la reformulation QCR au problème initial déjà convexe, on obtient les résultats du tableau 7.4. Les colonnes en gras représentent les résultats obtenus avec la méthode QCR (les valeurs optimales obtenues par relaxation continue pour différentes valeurs de t et les sauts d'intégrité associés) :

TAB. 7.4 – Matrice de variance-covariance

t	Risque	**QCR**		pb.init.	
	(variance)	**relax. cont.**	**$gap_{QCR}(\%)$**	relax. cont.	$gap_{init}(\%)$
50	270	**249.97**	**7.41**	206.35	23
55	270	**204.88**	**24**	98.45	63
60	270	**169.29**	**37**	39.56	85
62	270	**158.40**	**41**	23.69	91
65	270	**146.12**	**45**	7.73	97
70	270	**137.03**	**49**	0.41	99
80	270	**136.90**	**49**	0.41	99
90	170	**136.90**	**19**	0.41	99
100	170	**136.90**	**19**	0.41	99
110	170	**136.90**	**19**	0.41	99
120	170	**136.90**	**19**	0.41	99
130	170	**136.90**	**19**	0.41	99
140	170	**136.90**	**19**	0.41	99

Les temps de résolution du programme $(SDCP)$ sont très rapides, dûs à la petite taille des instances considérées.

Clairement, la borne obtenue par reformulation est meilleure que celle du programme non modifié. Ainsi, pour chaque valeur de t, les sauts d'intégrité sont nettement inférieurs lorsque le problème est reformulé par QCR. Par exemple, lorsque t est égal à 140, on passe d'un écart relatif de 99% à 19%. ♢

Pour complèter ces premiers résultats, QCR va être appliqué à des graphes générés aléatoirement.

7.1.2 Comparaison entre QCR et la résolution directe par un solveur MIQP

Pour appliquer QCR au problème (CP), nous avons créé des graphes aléatoires $G = (X, U)$ possédant une source v_1 et un puits v_n. Le problème consistera donc à trouver le plus court chemin probabiliste entre v_1 et v_n. Pour déterminer G, un premier graphe $G' = (X', U')$ est créé via l'utilisation du générateur de graphe *rudy*

http ://www-user.tu-chemnitz.de/∽helmberg/sdp_software.html.

On rappelle que *rudy* prend en paramètre le nombre de sommet $\mid X' \mid$ que l'on désire avoir, la densité (et donc $\mid U' \mid$) ainsi qu'une "graine" qui permet de créer différents graphes à partir d'un même nombre de sommets et d'une même densité. Par exemple, si l'on entre en ligne de commande :

rudy -rnd_graph 10 25 100 et *rudy -rnd_graph 10 25 101*

on obtiendra deux graphes différents à 10 sommets et de densité 25%.

Le graphe G' est un graphe par niveau car le générateur crée les arêtes (i, j) tel que $i < j$. De plus, il ne possède pas forcément une unique source et un unique puits. Nous avons donc décidé d'en créer, au cas où il n'en existerait pas. Voici l'algorithme simple que nous avons développé :

1. Pour tout sommet sans prédecesseur, on lui ajoute une arête entrante dont l'extrémité initiale est la source,
2. Pour tout sommet sans successeur, on lui ajoute une arête sortante dont l'extrémité terminale est le puits.

Le graphe G ainsi créé possède deux sommets supplémentaires comparé au graphe G' ainsi qu'un nombre d'arêtes supplémentaires égal au nombre

de sommets de G' sans successeurs et sans prédécesseurs.

A partir de G, on peut maintenant déterminer l'historique des temps de parcours. Nous avons choisi de créer l'historique sur 5 périodes et les données sont générées uniformément sur l'intervalle $[1, 10]$.

Avec ces instances ainsi estimées, nous avons comparé QCR avec la résolution directe du problème (CP). Nous avons résolu 5 instances différentes pour 3 types de densité ($25\%, 50\%$ et 75%) et trois valeurs de t. Pour déterminer à partir de quelle valeur de t le programme (CP) admet une solution admissible, nous avons tout d'abord résolu le programme linéaire suivant pour chaque instance :

$$\min \left\{ \sum_{i=1}^{n} \bar{t}_i x_i : x \in \{0,1\}^n \right\}$$

Les tableaux 7.5 à 7.13 présentent les résultats complets.

Légende des tableaux 7.5 à 7.13 :

- *num*, numéro de l'instance considérée
- n, nombre de sommets (premier paramètre + 2 du générateur *rudy*)
- d, densité du graphe G'
- *opt*, valeur optimale de (CP)
- *borne*, borne à l'arborescence de recherche du branch-and-bound, i.e. la valeur optimale de $(\overline{CP})$ ou de $(\overline{CP_{\alpha^*,u^*}})$
- *gap*, saut d'intégrité entre la valeur optimale et la borne calculée à la racine de l'arborescence de recherche
- CPU_{CSDP}, temps CPU requis par le logiciel $CSDP$ pour résoudre $(SDCP)$.

TAB. 7.5 – Comparaison entre l'approche directe et QCR ($n = 10, d = 25\%$)

			pb.init.		QCR		
num	*t*	*opt*	*borne*	*gap*(%)	*borne*	CPU_{CSDP}	*gap*(%)
1	18	17.5	8.15	53	15.65	0.37"	11
	20	17.5	2.42	86	6.66	0.17"	62
	22	2.2	1.95	11	2.2	0.36"	0
2	15	7.7	3.26	58	6.14	0.36"	20
	18	7.7	3.26	58	5.74	0.29"	25
	20	7.7	3.26	58	5.74	0.16"	25
3	15	3.5	1.20	66	3.50	0.35"	0
	18	3.5	1.15	67	3.5	0.28"	0
	20	3.5	1.15	67	3.50	0.14"	0
4	15	4.8	1.09	77	2.14	0.53"	55
	18	4.8	1.06	78	2.14	0.52"	55
	20	4.8	1.06	78	2.14	0.23"	55
5	15	10.3	0.09	99	4.95	0.41"	52
	18	10.3	0.09	99	4.95	0.41"	52
	20	10.3	0.09	99	4.95	0.20"	52

TAB. 7.6 – Comparaison entre l'approche directe et QCR ($n = 10, d = 50\%$)

			pb.init.		QCR		
num	*t*	*opt*	*borne*	*gap*(%)	*borne*	CPU_{CSDP}	*gap*(%)
1	15	27.7	27.7	0	27.7	0.58"	0
	18	4.7	3.32	29	4.67	0.74"	0
	20	4.7	0.75	84	4.14	0.35"	12
2	15	16.8	4.01	76	9.98	0.75"	41
	18	20.3	4.17	79	12.68	0.75"	38
	20	20.3	3.06	85	8.30	0.34"	59
3	15	11.3	2.57	77	6.14	1.1"	46
	18	16.3	1.11	93	5.50	0.75"	66
	20	16.3	0.22	99	3.01	0.35"	82
4	13	13.5	8.93	34	13.5	0.21"	0
	14	13.5	0.41	97	5.84	0.24"	57
	15	13.5	0.15	99	3.10	1.03"	77
5	13	24.7	13.74	44	22.26	0.14"	0
	14	24.7	10.86	56	19.44	0.14"	21
	15	24.7	9.37	62	17.63	0.72"	29

TAB. 7.7 – Comparaison entre l'approche directe et QCR ($n = 10, d = 75\%$)

			pb.init.		QCR		
num	*t*	*opt*	*borne*	*gap*(%)	*borne*	CPU_{CSDP}	*gap*(%)
1	15	18.3	13.80	25	15.29	1.88"	16
	18	18.3	10.6	42	12.89	1.80"	30
	20	18.3	9.9	46	12.55	0.83"	31
2	14	48.5	29.93	38	38.2	0.49"	21
	15	2.3	1.48	36	2.3	2.20"	0
	16	2.3	0.31	86	0.79	0.52"	66
3	15	28.7	3.61	87	14.83	2.13"	48
	18	10.3	0.04	99	1.07	2.45"	90
	22	3.5	0.034	99	0.61	2.23"	83
4	15	24.5	18.26	25	20.08	1.95"	18
	18	14.7	6.96	53	11.64	1.90"	21
	20	14.7	6.79	54	9.15	0.82"	38
5	15	3.2	1.53	52	2.17	2.53"	32
	18	3.2	0.74	77	1.34	2.74"	58
	20	3.2	0.38	88	0.89	1.16"	72

TAB. 7.8 – Comparaison entre l'approche directe et QCR ($n = 15, d = 25\%$)

			pb.init.		QCR		
num	*t*	*opt*	*borne*	*gap*(%)	*borne*	CPU_{CSDP}	*gap*(%)
1	11	21.8	21.8	0	21.8	5.11"	0
	12	21.8	9.12	58	14.74	6.32"	32
	13	21.8	3.05	86	8.48	6.33"	61
2	13	22.7	22.7	0	22.7	3.85"	0
	15	28.8	13.24	54	16.68	4.65"	42
	17	10.7	0.32	97	4.07	4.43"	62
3	13	17.5	2.79	84	6.50	4.46"	63
	15	28.8	28.8	0	28.8	4.02"	0
	17	28.8	5.68	80	9.40	4.68"	67
4	14	11.5	6.53	43	10.11	4.45"	12
	15	11.5	1.85	84	6.53	4.48"	43
	16	11.5	0.96	92	4.20	4.45"	63
5	13	20.8	20.8	0	20.8	4.10"	0
	15	8.7	2.87	67	5.15	5.11"	41
	17	8.7	0.07	99	1.30	5.11"	85

TAB. 7.9 – Comparaison entre l'approche directe et QCR ($n = 15, d = 50\%$)

			pb.init.		QCR		
num	*t*	*opt*	*borne*	*gap*(%)	*borne*	CPU_{CSDP}	*gap*(%)
1	17	51.7	51.7	0	51.7	18.23"	0
	18	36.7	16.94	54	23.97	22.37"	35
	20	6.8	1.01	85	1.61	25.31"	76
2	13	7.8	7.8	0	7.8	19.68"	0
	15	7.8	1.60	79	2.30	23.88"	71
	17	6.7	0.25	96	0.56	25.33"	92
3	17	9.2	9.2	0	9.2	18.21"	0
	18	9.2	5.08	45	5.78	23.65"	37
	20	9.2	3.00	67	3.46	23.46"	62
4	13	3.3	3.3	0	3.3	19.66"	0
	15	3.3	0.76	77	1.02	27.40"	69
	17	3.3	0.031	99	0.10	28.89"	97
5	13	28.5	28.5	0	28.5	21.09"	0
	15	9.8	6.70	32	8.03	27.57"	18
	17	9.8	0.67	93	1.90	29.37"	81

TAB. 7.10 – Comparaison entre l'approche directe et QCR ($n = 15, d = 75\%$)

			pb.init.		QCR		
num	*t*	*opt*	*borne*	*gap*(%)	*borne*	CPU_{CSDP}	*gap*(%)
1	16	33.3	30.68	0	33.3	2'15"	0
	17	27.7	12.67	54	14.18	1'15"	49
	18	18.7	7.98	57	8.81	1'3"	53
2	14	20.3	19.03	55	19.19	1'15"	5
	16	20.3	9.61	53	10.54	1'10"	48
	18	14.7	3.24	78	3.91	1'15"	73
3	13	46.2	46.2	0	46.2	1'6"	0
	14	20.3	5.99	70	13.48	1'10"	34
	16	4.3	0.20	95	0.31	1'98"	93
4	12	7.7	7.7	0	7.7	1'10"	0
	13	7.7	4.65	40	4.83	1'35"	37
	15	6.7	1.42	79	1.59	1'35"	76
5	15	11.5	11.44	0	11.5	1'39"	0
	16	11.5	5.48	52	6.23	1'35"	46
	17	11.5	1.72	86	2.56	1'40"	78

TAB. 7.11 – Comparaison entre l'approche directe et QCR ($n = 20, d = 25\%$)

			pb.init.		QCR		
num	*t*	*opt*	*borne*	*gap*(%)	*borne*	CPU_{CSDP}	*gap*(%)
1	11	19.7	19.7	0	19.7	38.11"	0
	12	19.7	7.44	37	12.46	52.86"	37
	13	19.7	3.58	82	7.23	49.86"	63
2	15	5.7	5.7	0	5.7	35.89"	0
	16	5.7	3.53	38	3.95	43.66"	31
	17	5.7	2.39	58	2.81	43.90"	51
3	14	13.2	10.47	21	11.31	48.14"	14
	15	13.2	3.13	76	5.24	46.63"	60
	16	5.7	0.65	86	1.54	49.94"	73
4	15	31.7	31.7	0	31.7	34.37"	0
	16	24.5	13.86	43	17.92	43.87"	27
	17	13.3	5.91	56	9.20	41.45"	31
5	14	13.3	12.44	6	12.57	38.01"	26
	15	13.3	9.30	30	9.83	39.41"	26
	17	13.3	3.57	73	4.30	38.25"	68

TAB. 7.12 – Comparaison entre l'approche directe et QCR ($n = 20, d = 50\%$)

			pb.init.		QCR		
num	*t*	*opt*	*borne*	*gap*(%)	*borne*	CPU_{CSDP}	*gap*(%)
1	16	39.3	39.3	0	39.3	4'8"	0
	17	17	9.95	41	12.85	20'11"	24
	18	17	4.68	72	6	20'53"	65
2	19	24.3	24.3	0	24.3	3'53"	0
	20	21.7	2.56	88	3.59	18'38"	83
	21	9.3	1.03	81	1.30	18'29"	0
3	16	18	18	0	18	4'	40
	17	11.3	6.23	45	6.76	19'14"	40
	19	6.8	1.23	82	1.57	18'29"	77
4	15	17.5	12.41	8	17.5	4'19"	0
	16	17.5	5.46	69	6.90	18'44"	61
	17	17.5	1.41	92	2.50	19'25"	86
5	14	62.5	41.45	34	49.68	6'8"	21
	15	23.8	7.61	68	14.04	7'52"	41
	16	19.3	0.007	99	0.30	20'38"	98

TAB. 7.13 – Comparaison entre l'approche directe et QCR ($n = 20, d = 75\%$)

			pb.init.		QCR		
num	*t*	*opt*	*borne*	*gap*(%)	*borne*	CPU_{CSDP}	*gap*(%)
1	19	17	17	0	17	16'18"	0
	21	2.5	1.75	30	1.79	1h4'	28
	22	2.5	1.16	53	1.23	1h3'	51
2	20	18.7	11.32	39	12.09	1h5'	35
	21	11.3	6.72	41	7.17	1h6'	36
	22	11.3	3.42	70	3.74	1h5'	67
3	17	9	6.47	28	6.55	1h4'	27
	18	9	4.60	49	4.68	1h4'	48
	20	9	2.26	75	2.37	1h4'	74
4	11	52.3	52.3	0	52.3	15'11"	0
	12	17.3	10.16	41	14.08	1h9'	19
	14	14.3	0.88	94	1.09	1h8'	92
5	14	5.7	4.15	27	4.46	1h6'	22
	15	5.7	2.42	57	2.59	1h1'	55
	16	5.3	1.01	81	1.13	1h4'	79

Deux raisons nous ont incités à ne pas résoudre des instances de plus grande taille. La première est la plus contraignante puisqu'elle concerne les limitations mémoire du logiciel $CSDP$. En effet, imaginons que l'on génère une instance à 50 sommets avec une densité de 50%, alors le nombre d'arêtes peut atteindre 600 et le programme $(SDCE)$ peut contenir jusqu'à 30000 contraintes. Or, l'espace mémoire requis par $CSDP$ doit au moins contenir $8 \times r^2$ bits de mémoire où r représente le nombre de contraintes du programme $(SDCE)$. Ainsi, il faut plus de 7.2 gigabits pour résoudre une instance à 50 sommets, ce qu'un ordinateur standard ne possède pas.

La seconde raison prend en compte les résultats que nous avons obtenus pour les graphes à 10, 15 et 20 sommets. En effet, au vue de ces premiers résultats, nous verrons que notre approche peut ralentir significativement le temps de résolution dit "direct", c'est-à-dire le temps CPU correspondant à la résolution entière de (CP).

Avant de commenter les différents tableaux, il faut noter que nous n'avons pas reporté les temps de résolution exacte par le logiciel $CPLEX9$ car ils sont toujours très faibles (tous inférieurs à une demi-seconde).

Pour les graphes à 10 sommets, quelle que soit la densité (Tableaux 7.5 à 7.7), la borne obtenue avec QCR (et donc par relaxation continue de (CP_{α^*,u^*})) est toujours supérieure à celle obtenue par relaxation continue de (CP). On remarque qu'on peut ainsi améliorer significativement les sauts d'intégrité grâce à notre reformulation. Par exemple, lorsque $d = 25\%$ et pour l'instance 3, on passe d'un saut d'intégrité de 66% à 0%.

De plus, pour une instance donnée, plus la valeur de t augmente et plus la borne obtenue par relaxation continue de (CP) et (CP_{α^*,u^*}) est mauvaise. Dans ce cas, l'amélioration par QCR est variable. On peut tout aussi bien atteindre un saut d'intégrité de 52% avec QCR ($d = 25\%$ et $num = 5$) au lieu de 99%, tout comme un saut d'intégrité de 72% avec QCR au lieu de 88% ($d = 75\%$ et $num = 5$). Dans ce dernier cas, l'amélioration de la relaxation n'est pas considérable, même si elle reste intéressante.

Cependant, appliquer QCR sur les instances à 10 sommets se révèle tout de même efficace car les temps de résolution de $CSDP$ sont très faibles et donc négligeables.

Les mêmes remarques sont à faire pour les graphes à 15 et 20 sommets (Tableaux 7.8 à 7.13), excepté sur les temps de résolution des programmes semidéfinis. En effet, pour les graphes à 15 sommets, $CSDP$ met environ 5" pour trouver l'optimum quand $d = 25\%$, 25" quand $d = 50\%$ et plus d'une minute quand $d = 75\%$. Dans ce cas, la reformulation QCR devient inefficace puisque malgré une amélioration de la borne inférieure obtenue par relaxation continue, les temps de résolution sont nettement supérieurs (on rappelle que la résolution du problème non modifié ne met pas plus d'une demi seconde).

On observe le même phénomène quand $n = 20$ mis à part que les temps CPU de $CSDP$ sont encore plus longs (entre 34" et 1h9').

Bien sûr, ces premiers résultats ne sont pas très significatifs car nous résolvons des instances convexes très faciles malgré la mauvaise qualité des bornes inférieures obtenues par relaxation continue. Cependant, cela nous donne un premier aperçu sur l'amélioration que pourrait apporter la reformulation QCR.

La section suivant est plus concluante et montre qu'appliquer QCR sur un problème d'investissements (convexe) se révèle très efficace.

7.2 Le problème d'investissements

Un investisseur souhaite retenir p projets parmi n. Chaque projet a le même coût et a un rendement représenté par une variable aléatoire. L'espérance de rendement e_i $(i = 1, \ldots, n)$, la variance σ_{ii} $(i = 1, \ldots, n)$, et la covariance de chaque paire de projets, σ_{ij} $(i = 1, \ldots, n-1; j = i+1, \ldots, n)$ sont estimés à partir d'un historique des rendements observés. Le problème consiste à choisir p projets de façon à obtenir une espérance de rendement supérieure ou égale à une valeur fixée ρ tout en minimisant le risque, c'est-à-dire la variance du rendement. Le programme mathématique convexe en variables bivalentes associé est :

$$(PI) \quad : \quad \min \left\{ r(x) = \frac{1}{p^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \sigma_{ij} x_i x_j : \sum_{i=1}^{n} x_i = p, \sum_{i=1}^{n} e_i x_i \geq p\rho, \ x \in \{0,1\}^n \right\}$$

Schématiquement, l'investisseur souhaite constituer un portefeuille de projets et arbitrer entre les gains et les risques. Pour notre problème d'investissements, il doit s'assurer le risque minimal à rendement donné.
Les études théoriques réalisées pour ce type de problème supposent que l'on connait le couple "espérance de rendement" et "risque de chaque titre", information qui ne se trouve pas évidemment telle quelle dans la presse économique et qui fait une large part à l'appréciation de l'investisseur.

Exemple 4 :

Afin de mieux comprendre ce problème, on présente une application numérique avec 3 projets à choisir parmi 6. Le tableau 7.14 représente l'historique des rendements de chaque projet sur 5 années. On note r_{ij} le rendement du projet i à l'année j. L'espérance e_i est égale à la quantité $0.2 \sum_{k=1}^{5} r_{ik}$.

A partir de ces données, on peut construire la matrice de variance covariance : $\sigma_{ij} = 0.25 \sum_{k=1}^{t} (r_{ik} - e_i)(r_{jk} - e_j)$. Elle est estimée dans le tableau 7.15.

TAB. 7.14 – Historique des rendements et estimation de l'espérance de rendement de chaque projet

Projets	Rendement année 1 $r_{i1}(\%)$	Rendement année 2 $r_{i2}(\%)$	Rendement année 3 $r_{i3}(\%)$	Rendement année 4 $r_{i4}(\%)$	Rendement année 5 $r_{i5}(\%)$	Espérance de rendement $e_i(\%)$
1	3	-1	6	7	-2	2.6
2	4	3	0	0	1	1.6
3	4	3	6	1	-3	2.2
4	5	2	-1	5	5	3.2
5	3	-1	0	2	-1	0.6
6	-3	0	5	3	5	2

TAB. 7.15 – Estimation de la matrice de variance covariance

	1	2	3	4	5	6
1	16.3	-3.45	7.35	-2.4	4.55	1.5
2	-3.45	3.3	1.1	1.35	0.8	-5.75
3	7.35	1.1	11.7	-6.3	1.85	-4
4	-2.4	1.35	-6.3	7.2	2.1	-3
5	4.55	0.8	1.85	2.1	3.3	-3.5
6	1.5	-5.75	-4	-3	-3.5	12

L'investisseur va prendre un risque minimal qui dépend du rendement moyen qu'il souhaite obtenir, autrement dit, qui dépend de la valeur de ρ qu'il va vouloir fixer. Pour notre exemple, on peut construire la fonction représentant le risque minimal à prendre en fonction des valeurs de ρ :

Le tableau associé est :

ρ	*opt*	*borne*	*gap*(%)
1.4	0.189	0.025	89
1.8	0.49	0.0205	96
2	0.49	0.0205	96
2.2	0.49	0.0205	89
2.3	0.49	0.049	80
2.4	0.49	0.103	79
2.5	3.07	0.6	86
2.6	3.07	1.81	41

La colonne *borne* représente la valeur optimale de $(\overline{PI})$. Si l'on pose $\rho = 2.5\%$, la solution optimale consiste à retenir les projets 1, 4 et 6, l'espérance de rendement est alors 2.6% avec une variance de 3.07 et la valeur de la solution optimale de la relaxation continue est égale à 0.6.

La colonne *gap* représente l'écart relatif entre la borne obtenue par relaxation continue et la valeur optimale de (PI). ♢

7.2.1 La reformulation QCR

On applique la reformulation QCR au problème (PI). La fonction économique convexe $r(x)$ est remplacée par :

$$r_{\alpha,u}(x) = \frac{1}{p^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_{ij}x_ix_j + \left(\sum_{i=1}^{n}\alpha_i x_i\right)\left(\sum_{j=1}^{n}x_j - p\right) + \sum_{i=1}^{n}u_i(x_i^2 - x_i)$$

Les valeurs optimales de α et u sont calculées grâce au programme semidéfini

suivant :

$$
\begin{array}{lll}
(SDCP): \quad \min & \dfrac{1}{p^2}\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_{ij}X_{ij} & \\
\text{s.c.} & X_{ii} = x_i \qquad i = 1,\ldots,n & \leftarrow u_i^* \\
& -px_i + \displaystyle\sum_{j=1}^{n} X_{ij} = 0 \qquad i = 1,\ldots,n & \leftarrow \alpha_i^* \\
& \displaystyle\sum_{i=1}^{n} x_i = p & \\
& \displaystyle\sum_{i=1}^{n} e_i x_i \geq p\rho & \\
& \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 & \\
& x \in \mathbb{R}^n, X \in S_n &
\end{array}
$$

et on résout le nouveau problème reformulé, toujours convexe :

$$
(PI_{\alpha^*,u^*}) \quad : \quad \min \left\{ r_{\alpha^*,u^*}(x) : \sum_{i=1}^{n} x_i = p, \sum_{i=1}^{n} e_i x_i \geq p\rho,\ x \in \{0,1\}^n \right\}
$$

Exemple 4 :

Pour l'exemple donné en début de section, les nouvelles valeurs des bornes calculées à partir de la relaxation continue du problème modifié sont reportées dans le tableau 7.16.

Les temps de résolution pour cet exemple sont bien sûr très rapides car la taille des instances est petite. Seulement, on peut déjà remarquer que la borne obtenue par reformulation est meilleure que celle obtenue pour le problème non modifié. Le saut d'intégrité est significativement diminué : il peut être jusqu'à 3,7 fois plus petit. La borne est donc bien sûr meilleure pour le problème réécrit même si pour certaines instances, elle n'est pas très fine comparée à la valeur optimale. ♢

D'après l'exemple 4, on peut se demander si, pour des instances plus difficiles à résoudre, la diminution du saut d'intégrité permet de diminuer significativement les temps de résolution des problèmes considérés.

TAB. 7.16 – Comparaison des résolutions de (PI) et (PI_{α^*,u^*})

		pb initial		QCR	
ρ	opt	bound	gap(%)	$bound_{QCR}$	$gap_{QCR}(\%)$
1.4	0.189	0.025	89	**0.14**	26
1.8	0.486	0.0205	96	**0.14**	71
2	0.486	0.0205	96	**0.14**	71
2.2	0.189	0.0205	89	**0.14**	26
2.3	0.48	0.049	80	**0.20**	58
2.4	0.48	0.103	79	**0.34**	29
2.5	3.07	0.6	86	**0.95**	69
2.6	3.07	1.81	41	**2.38**	22

7.2.2 Comparaison entre QCR et la résolution directe par un solveur MIQP

Pour répondre à la question précédente, nous avons testé la méthode QCR sur deux types d'instances générées aléatoirement (plus précisément, on a choisi de générer aléaroirement l'historique des rendements) :

- *Classe histo+10* : On considère que l'historique est sur $n+10$ périodes (donc par exemple sur 80 périodes si l'on considère des portefeuilles à 70 projets). Les rendements sur chaque période varient aléatoirement selon une loi uniforme dans l'intervalle $[-10, 10]$.
- *Classe histo10* : On considère maintenant que l'historique est toujours sur 10 périodes, quelque soit le nombre de projets considéré. Les rendements sur chaque période varient aléatoirement selon une loi uniforme dans l'intervalle $[-3, 20]$. Cette classe de problèmes représente des données plus réalistes que la première classe et est plus difficile à résoudre.

Avant de résoudre les différentes instances, il a fallu choisir les valeurs de ρ. Pour cela, les deux programmes linéaires en 0-1 suivants ont été résolus dans le but de connaitre les valeurs admissibles du paramètre ρ :

$$(PL_{\rho_{min}}) \quad \min \left\{ \sum_{i=1}^{n} \frac{1}{p} e_i x_i : \sum_{i=1}^{n} x_i = p, x \in \{0,1\} \right\}$$

$$(PL_{\rho_{max}}) \quad \max \left\{ \sum_{i=1}^{n} \frac{1}{p} e_i x_i : \sum_{i=1}^{n} x_i = p, x \in \{0,1\} \right\}$$

Les valeurs de ρ sont ensuite fixées dans l'intervalle $[v(PL_{\rho_{min}}), v(PL_{\rho_{max}})]$

Les tableaux 7.17 à 7.21 représentent les résultats obtenus pour les instances de la *Classe histo+10*.

Légende des tableaux 7.17 à 7.21 :

- $p^2 * opt$ correspond à la valeur optimale de l'instance considérée multipliée par p^2.
- *borne* représente la valeur optimale de la relaxation continue de (PI) ou de (PI_{α^*,u^*}). *borne* n'est autre que la borne inférieure obtenue par relaxation continue à la racine de l'arborescence de recherche.
- CPU, temps de calcul requis par le logiciel $CPLEX9$ pour résoudre (PI).
- CPU_{exact}, temps de calcul requis par le logiciel $CPLEX9$ pour résoudre (PI_{α^*,u^*}).
- $CPU_{CPLEX+SB}$, temps de calcul requis par le logiciel $CPLEX9$ et le logiciel SB pour calculer (α^*, u^*) et résoudre (PI_{α^*,u^*}).

Plusieurs remarques sont à faire concernant la résolution des programmes semidéfinis $(SDPI)$.

Tout d'abord, il faut noter que nous avons choisi le logiciel SB pour résoudre $(SDPI)$. Il aurait cependant été plus naturel d'utiliser $CSDP$ car, pour une même instance, le temps CPU pour la résolution de la relaxation semidéfinie est nettement inférieur. Mieux, la valeur optimale peut être légèrement supérieure. Cependant, lorsque les paramètres optimaux α^* et u^* sont déterminés par SB, le branch-and-bound se comporte beaucoup mieux par la suite. Par exemple, pour l'instance de la classe *histo*10 à 70 projets et où $p = 40$ et $\rho = 7$, le temps de résolution de $(SDPI)$ est de 3" avec $CSDP$ (la borne inférieure associée est égale à 3.97) contre $4'$ avec SB (la borne inférieure associée est égale à 3.89). Mais ensuite, la résolution par $CPLEX9$ se fait en 10h18' quand le problème est reformulé avec $CSDP$ (ce temps CPU est même plus grand que la résolution du programme non modifié : 6h12' !) et en 5h39' avec SB (amélioration du temps de résolution de (PI)).

Concernant la résolution des instances de la classe *histo+10*, nous avons été contraint de réaliser la résolution de $(SDPI)$ avec ou sans les contraintes

d'inégalité $\sum_{i=1}^{n} e_i x_i \geq p$ selon les valeurs de ρ. En effet, pour les valeurs de ρ telles que les contraintes d'inégalité sont toujours vérifiées (redondance), le logiciel SB ne permet pas de rendre le nouveau hessien semidéfini positif. C'est pourquoi nous avons choisi, dans ce cas, de résoudre $(SDPI)$ sans les contraintes d'inégalité. Bien sûr, quand les contraintes deviennent nécessaires, nous les laissons dans la relaxation et la résolution sous SB est cette fois-ci possible.

TAB. 7.17 – Comparaison des résolutions de (PI) et (PI_{α^*,u^*}) : $n = 70$

			pb initial		QCR		
p	ρ	$p^2 * opt$	*borne*	CPU	*borne*	CPU_{exact}	$CPU_{CPLEX+SB}$
25	5	962.44	500.51	1'45"	659.37	46.22"	**1'02"**
	10	962.44	500.51	1'39"	659.37	41.69"	**48.84''**
	11	1749.37	1260.99	**8.96"**	1462.2	3.74"	15.89"
	11.5	4127.20	3824.66	**0.04"**	3996.08	0.04"	1'
40	5	2209.87	1777.13	**16.17"**	1920.72	10.4"	1'10"
	10	2209.87	1777.13	**13.27"**	1920.72	7.5"	1'14"
	10.5	2789.81	2378.46	**1.55"**	2554.42	0.78"	4.74"
50	5	4033.15	3624.52	**3.19"**	3785.86	1.38"	15.67"
	10	4061.71	3637.83	**2.83"**	3789.33	2.23"	22.44"
	10.5	5516.46	5262.54	**0.05"**	5418.68	0.1"	4.7"

TAB. 7.18 – Comparaison des résolutions de (PI) et (PI_{α^*,u^*}) : $n = 100$

			pb initial		QCR		
p	ρ	$p^2 * opt$	*borne*	CPU	*borne*	CPU_{exact}	$CPU_{CPLEX+SB}$
45	5	2655.01	2041.496	1h25'	2206.97	31'48"	**33'58"**
	10	2655.01	2041.496	1h25'	2206.97	34'6"	**34'45"**
	11	5489.44	4982.39	**5.89"**	5203.61	2.56"	6"
	11.2	7693.94	7289.33	**0.44"**	7495.28	0.44"	15.75"
50	5	3350.33	2700.44	2h37'	2882.90	42'6"	**43'15"**
	10	3350.33	2700.44	2h36'	2882.92	1h2"	**1h3'32"**
	11	7511.93	7087.35	**1.29"**	7282.50	0.86"	8.70"
55	5	4125.85	3502.02	55'55"	3708.98	15'38"	**19'8"**
	10	4146.53	3520.25	33'37"	3712.33	17'41"	**18'17"**
	10.5	5306.29	4686.85	3'4"	4904.63	1'49"	**2'20"**
	11	11715.14	11341.09	**0.11"**	11472.64	0.41"	32.21"
60	5	5062.09	4461.16	20'1"	4687.58	3'53"	**9'1"**
	10	5106.93	4474.55	21'42"	4691.07	15'6"	**17'9"**
	10.5	6644.04	6061.44	1'2"	6297.8	23.77"	**1'13"**
	10.6	7488.04	6910.64	**27.65"**	7164.41	8.34"	1'10"
65	5	6233.04	5601.72	18'	5832.37	3'37"	**3'54"**
	10	6265.59	5601.70	13'46"	5838.54	7'6"	**8'13"**
	10.5	8457.68	7852.33	**40.73"**	8117.93	10.85"	1'33"
	10.6	3756.96	9114.53	**43.82"**	9376.75	16.34"	1'
70	5	7548.03	6946.63	3'33"	7178.39	47.93"	**1'26"**
	10	7578.95	6955.19	3'	7186.28	1'38"	**2'29"**
	10.5	10716.97	10192.70	**0.07"**	10425.59	3.6"	10.1"
	10.6	12819.53	12379.36	**0.54"**	12632.23	1.11"	31.80"

TAB. 7.19 – Comparaison des résolutions de (PI) et (PI_{α^*,u^*}) : $n = 120, 200, 250$

				pb initial		QCR		
n	p	ρ	$p^2 * opt$	*borne*	CPU	*borne*	CPU_{exact}	$CPU_{CPLEX+SB}$
120	65	10.5	3996.21	3256.45	7h30'	3461.77	5h4'	**5h5'11"**
	70	10.5	4775.20	4019.24	4h25'	4242.12	2h45'	**2h45'25"**
		10.6	5386.18	4625.96	1h19'	4898.53	32'33"	**33'4"**
	75	10.5	5725.11	4953.60	2h45'	5195.51	1h21'	**1h22'**
		10.6	65275.51	5790.68	22'7"	6054.40	9'35"	**9'42"**
	80	5	4826.65	4136.50	50'30"	4327.23	35'27"	**36'22"**
		10	4826.65	4136.50	46'30"	4327.23	29'43"	**30'35"**
		10.5	6848.83	6102.00	33'49"	6352.80	18'57"	**20'35"**
		10.6	7936.58	7230.90	4'31"	7510.88	1'41"	**1'6"**
	90	5	6693.38	6026.26	8'51"	6277.92	3'10"	**3'26"**
		10	6693.38	6026.26	8'45"	6277.92	3'18"	**3'41"**
		10.5	9811.24	9281.16	**6.69"**	9542.31	3.4"	1'3"
		10.6	12616.58	12101.94	**0.08"**	12255.93	2.98"	52"
200	150	10.2	12815.30	11928.39	4h33'	12305.17	1h1"	**1h35"**
		10.5	27124.64	26857.54	**0.26"**	27050.50	1"	2'24"
	160	10.2	16481.15	15657.03	38'	16050.22	4'43"	**5'39"**
		10.4	25807.85	25326.64	**3.35"**	25655.40	4.07"	1'33"
	180	10.2	27866.12	27317.44	**7.8"**	27635.48	26.03"	1'28"
250	210	10.1	22638.52	21742.27	4h2'	22143.32	30'32"	**33'50"**
		10.2	28633.82	27843.70	27'23"	28263.63	13'27"	**17'**
	220	10.1	27614.97	26899.17	4'21"	27264.56	1'19"	**4'**
		10.2	40583.17	40471.52	**0.12"**	40578	0.07"	10'
	230	10.1	34580.46	34016.05	**6.67"**	34341.64	8.06"	3'38"

Les temps CPU notés en gras constituent le meilleur temps entre la résolution exacte directe de (PI) ou celle du problème transformé par QCR, i.e. (PI_{α^*,u^*}).

De manière générale, plus la valeur de p est petite, plus le problème est difficile à résoudre. Ainsi, nous avons choisi de déterminer la valeur optimale de (PI) ou (PI_{α^*,u^*}) seulement pour quelques valeurs de p : si pour une valeur $\tilde{p}$ donnée, le temps de résolution de (PI) dépasse une heure, on ne considérera pas les valeurs de p inférieures à $\tilde{p}$.

Considérons les résultats obtenus lorsque l'on a un ensemble de 70 projets (Tableau 7.17). En remarque générale, nous pouvons dire que, pour ces instances de taille moyenne, il n'est pas nécessaire d'appliquer la reformulation QCR. En effet, les temps de résolution requis par $CPLEX9$ sont inférieurs lorsque l'on résoud le problème transformé. Cependant, la résolution par SB ajoute un temps de calcul non négligeable au temps requis pas $CPLEX9$.

Maintenant, considérons le cas où l'investisseur peut choisir entre 100 projets (Tableau 7.18). Lorsque la valeur de p est grande, comme pour le cas à 70 projets, il n'est pas nécessaire de reformuler le problème car les temps CPU requis par $CPLEX9$ pour résoudre (PI) directement sont relativement faibles. Si l'on considère maintenant les autres valeurs de p, QCR peut devenir très intéressante. En effet, nous sommes capables de diminuer significativement les temps de résolution : on peut trouver l'optimum jusqu'à 5 fois plus rapidement (quand $p = 65$ et $\rho = 5$).

Les mêmes remarques sont à noter pour le cas où l'investisseur choisit entre 120, 200 ou 250 projets (Tableau 7.21) : lorsque p est assez grand, la reformulation n'est pas nécessairement justifiée. Cependant, lorsque la valeur de p diminue et ρ devient petit, la méthode QCR permet de résoudre le problème d'investissements de manière très efficace comparé à la résolution directe. Par exemple, pour le triplet $n = 200, p = 150$ et $\rho = 10.2$, on passe d'un temps de résolution de 4h30 à une résolution d'un peu plus d'une heure. L'amélioration est ici considérable.

Considérons maintenant les résultats obtenus pour la classe d'instances *histo*10.

TAB. 7.20 – Comparaison des résolutions de (PI) et (PI_{α^*,u^*}) : $n = 70, 100, 150$

				pb initial		QCR		
n	p	ρ	$p^2 * opt$	*borne*	CPU	*borne*	CPU_{exact}	$CPU_{CPLEX+SB}$
70	25	7.8	3	1.57	40'30"	2.10	30'3"	**30'20"**
		8	4.75	1.79	**28.64"**	2.43	25.13"	46"
		8.1	7.82	1.94	**2.81"**	2.44	2.23"	22"
	40	7	4	2.91	6h12'	3.89	5h39'	**5h43'**
		7.3	9.68	3.51	**4.93"**	3.95	3.75"	28'4"
		7.5	97.09	67.09	**0.24"**	68.20	0.61"	9'47"
	50	5	11.75	6.18	**0.54"**	6.31	0.53"	54"
		7	129.26	99.33	**0.47"**	100.08	0.87"	4'9"
100	75	7.1	8.14	6.89	54'6"	7.41	**51'10"**	52'51"
		7.15	9.24	6.79	**1'14"**	7.42	53.14"	3'53"
		7.2	17.86	7.35	**2.54"**	7.52	2.26"	13.26"
	80	6.2	8.56	7.17	19'15"	7.92	15'14"	**17'50"**
		7	16.73	7.72	31.36"	7.98	4.26"	**10.26"**
		7.2	311.16	214.09	**0.09"**	243.31	0.6"	1'28"
150	100	7.5	9.99	9	8h49'	9.77	4h38'	**4h38'33''**
		7.55	17.28	10.37	**10.18"**	10.57	8.81"	22.8"
		7.6	65.08	50.31	**0.54"**	50.72	3.12"	6'43"
	110	7.3	16.22	10.85	**33.39"**	11.07	33.27"	55.27"
		7.5	524.44	509.92	**0.08"**	510.52	1.50"	6'20"
	120	6.2	15.49	12.34	**1.79"**	12.49	1.31	1'41"
		6.1	15.5	12.34	**1.72"**	12.29	1.46"	1'1"
		6.9	18.84	12.82	**4.73"**	12.98	4.14"	32"

TAB. 7.21 – Comparaison des résolutions de (PI) et (PI_{α^*,u^*}) : $n = 200, 250$

				pb initial		QCR		
n	p	ρ	$p^2 * opt$	*borne*	CPU	*borne*	CPU_{exact}	$CPU_{CPLEX+SB}$
200	150	7.5	52.08	35.98	**1.41"**	36.09	1.27"	1'36"
		7.7	10173.04	10029.59	**0.53"**	10031.96	1.76"	12'22"
	160	7.28	17.22	15.21	41'29"	15.91	37'30"	**39'30"**
		7.29	19.29	15.43	**3'52"**	11.94	3'25"	4'45"
		7.3	22.88	16.02	**26.04"**	16.22	24.72"	58"
250	210	7.29	60.29	51.1	**0.89"**	51.22	0.71"	51'
		7.3	102.13	80.89	**2.71"**	80.75	19.16"	1h35'20"
		7.4	3080.78	3075.88	**0.12"**	1076.04	0.63"	14'40"
	220	7.09	23.34	21.42	2'19"	21.91	1'36"	**2'13"**
		7.1	58.96	42.52	**6.2"**	42.81	55.89"	52'54"
		7.2	665.88	630.57	**0.6"**	631.83	10.49"	22'11"

Comme pour les tableaux présentant les résultats de la classe $histo + 10$, les temps CPU notés en gras constituent le meilleur temps entre la résolution exacte directe de (PI) ou celle du problème transformé par QCR, i.e. (PI_{α^*,u^*}).

Nous pouvons formuler les mêmes remarques générales que pour la classe $histo + 10$, à savoir que la reformulation QCR devient très intéressante si l'instance déjà convexe est difficile à résoudre directement par $CPLEX9$. Par exemple, lorsque l'investisseur peut choisir entre 150 projets avec un rendement moyen de 7.5%, on passe d'un temps de résolution de 8h49' à 4h38'.

En revanche, lorsque la résolution de (PI) se fait en quelques secondes, QCR se révèle inefficace puisque, malgré une amélioration de la borne inférieure obtenue par relaxation continue, le temps CPU global (pré-traitement + résolution exacte) peut être nettement supérieur à cause de la difficulté de convergence des logiciels SDP.

7.3 Un problème d'ordonnancement

Dans cette section, nous nous intéressons à des problèmes d'ordonnancement où des tâches doivent être exécutées sur des machines parallèles. L'objectif est ici de minimiser la somme pondérée des dates de fin de tâches. Bien sûr, selon les problèmes d'ordonnancement que nous allons considérer, différentes hypothèses sont à faire, notamment sur les vitesses des machines.

Ce travail a été réalisé en collaboration avec Yasmin Rios, doctorante au laboratoire d'Informatique de l'Université Paris 6.

7.3.1 Tâches à exécuter sur des machines parallèles à vitesses différentes et indépendantes

Considérons le problème d'ordonnancement où les tâches doivent être exécutées sur des machines parallèles à vitesses différentes et indépendantes.

Données du problème :
- un ensemble J de n tâches
- m machines parallèles
- p_{ij}, temps d'exécution de la tâche i si elle est exécutée totalement sur la machine j
- $\beta_i \geq 0$, poids de la tâche j représentant une pénalité de retard (pondération selon l'importance de la tâche).

Variable du problème :
- C_i, date de fin de la tâche i.

Quelques contraintes doivent être prises en compte :
- une tâche ne peut être exécutée que sur une seule machine à la fois
- une machine ne peut exécuter qu'une seule tâche à la fois
- toutes les machines sont libres à la date 0

L'objectif du problème est ici de minimiser la somme pondérée des dates de fin de tâches et donc ainsi, de minimiser la quantité $\sum_{i \in J} \beta_i C_i$. Ce problème, dans la notation à trois champs de Graham et al. [68], est communément noté $R\|\sum_i \beta_i C_i$. Ici, R représente le cas où les machines sont parallèles, à vitesses différentes et indépendantes.

Un état de l'art

Le problème $R\|\sum_i \beta_i C_i$ est NP-difficile au sens fort ([31] [102]). Plusieurs méthodes exactes de type branch-and-bound ont été proposées, notamment par Chen et Powell [38] ainsi que par Azizoglu et Kirca [8] quand les machines sont différentes et à vitesses indépendantes. En particulier, les meilleurs résultats sont obtenus par Chen et al., qui sont capables de résoudre des problèmes à l'optimum pour des instances à 100 tâches et 20 machines. Belouadah et Potts [14] ont également développé un branch-and-bound, mais pour le cas où les machines sont parallèles et identiques. Enfin, Lee et Uzsoy [98] ont utilisé la programmation dynamique et Webster [135] a proposé des calculs de bornes inférieures.

Dans ce travail, nous allons nous concentrer que la formulation de $R\|\sum_i \beta_i C_i$ déterminée par Skutella [129] ainsi que sa reformulation quadratique convexe à partir de laquelle il développe une $(1-\epsilon)-$approximation. Il est important de noter que Skutella ne résout pas (PO) de manière exacte.

Il est intéressant de remarquer que le même problème, mais avec une seule machine (et donc $1\|\sum_i \beta_i C_i$), peut être résolu en temps polynomial en utilisant la règle des ratios de Smith [130]. L'idée est d'ordonnancer les tâches dans l'ordre non croissant des ratios $\frac{\beta_i}{p_{i1}}$.

La difficulté du problème avec des machines parallèles à vitesses différentes réside dans l'affectation des tâches aux machines. Ainsi, Skutella [129] définit un ordre total sur l'ensemble des tâches. Dans cette section, nous allons proposer la même modélisation, qui donne lieu à un programme en 0-1 avec un objectif linéaire mais sous contraintes quadratiques.

Soit $(J, \prec_i)$ l'ordre total de l'ensemble des tâches dans J pour une machine j. On définit $i \prec_j k$ si $\frac{\beta_i}{p_{ij}} > \frac{\beta_k}{p_{kj}}$. Si pour une machine j, il existe une paire de tâches i et k telles que $\frac{\beta_i}{p_{ij}} = \frac{\beta_k}{p_{kj}}$, alors on pose $i \prec_j k$ si $i < k$.

Le problème $R\|\sum_i \beta_i C_i$ peut se modéliser sous la forme suivante :

$$\begin{aligned}
\min \quad & \sum_{i\in J} \beta_i C_i \\
\text{s.c.} \quad & \sum_{j=1}^{m} x_{ij} = 1 && \forall i \in J \qquad (7.4)\\
& C_i = \sum_{j=1}^{m} x_{ij} \left(p_{ij} + \sum_{k \prec_j i} x_{kj} p_{kj} \right) && \forall i \in J \qquad (7.5)\\
& x \in \{0,1\}^{nm}
\end{aligned}$$

où les variables binaires $x_{ij}\ \forall i \in J,\ j = 1, \ldots, m$ sont définies comme suit :

$$x_{ij} = \begin{cases} 1 & \text{si la tâche i est exécutée sur la machine j} \\ 0 & \text{sinon} \end{cases}$$

Les contraintes linéaires (7.4) assurent que chaque tâche n'est affectée qu'à une seule et même machine. Si une tâche i est affectée à la machine j, alors les contraintes quadratiques (7.5) impliquent que sa date de fin est égale à la somme de son temps d'exécution p_{ij} additionnée à la somme des temps d'exécution des tâches affectées à cette même machine et vérifiant $k \prec_j i$.

L'idée maintenant est d'intégrer les contraintes quadratiques (7.5) dans la fonction objectif de manière à se ramener à un programme en 0-1 avec un objectif quadratique et des contraintes linéaires :

$$\begin{aligned}
(PO) \ : \ \min \quad & p(x) = \sum_{i\in J} \beta_i \sum_{j=1}^{m} x_{ij} \left(p_{ij} + \sum_{k \prec_j i} x_{kj} p_{kj} \right) \\
\text{s.c.} \quad & \sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \in J \\
& x \in \{0,1\}^{nm}
\end{aligned}$$

que l'on peut réécrire sous forme matricielle :

$$\begin{aligned}
(PO) \ : \ \min \quad & p(x) = \frac{1}{2} x^t Q x + c^t x \\
\text{s.c.} \quad & Ax = \mathbf{1} \quad \forall i \in J \\
& x \in \{0,1\}^{nm}
\end{aligned}$$

où $\mathbf{1}$ représente le vecteur unité à n composantes.

Les variables x_{ij} sont tout d'abord ordonnées lexicographiquement selon l'ordre des valeurs $1, \ldots, m$ des machines. Ensuite, pour chaque machine j, elles sont ordonnées selon l'ordre $\prec_j$. Si l'on pose $[r]^j$ l'index de la tâche effectuée à la position r ($r = 1, \ldots, n$) selon l'ordre $(J, \prec_j)$ de la machine j, alors x a la forme suivante :

$$x^t = \left(x_{[1]^1 1}, \ldots, x_{[n]^1 1}, x_{[1]^2 2}, \ldots, x_{[n]^2 2}, \ldots, x_{[n]^m m}\right)$$

ou plus simplement,

$$x^t = \left(x_{[1]1}, \ldots, x_{[n]1}, x_{[1]2}, \ldots, x_{[n]2}, \ldots, x_{[n]m}\right)$$

Chaque composante c_{ij} du vecteur $c \in \mathbb{R}^{nm}$ est égale à $c_{ij} = \beta_i p_{ij}$ et est ordonnée de la même manière que pour le vecteur x. La matrice $Q = q_{(ij)(hk)}$ de dimension $nm \times nm$ est de la forme :

$$q_{(ij)(hk)} = \begin{cases} 0 & \text{si } j \neq h \quad \text{ou } i = k \\ \beta_i p_{ik} & \text{si } j = h \quad \text{ou } k \prec_j i, \\ \beta_k p_{ij} & \text{si } j = h \quad \text{ou } i \prec_j k \end{cases}$$

Du fait de l'ordre lexicographique sur les indices, la matrice Q peut se décomposer en m blocs diagonaux Q_i ($i = 1, \ldots, n$), correspondant aux m machines (pour plus de clareté, on remplacera ici $\beta_{[k]^j} p_{[l]^j j}$ par $\beta_k p_l$) :

$$Q_j = \begin{pmatrix} 0 & \beta_2 p_1 & \beta_3 p_1 & \ldots & \beta_n p_1 \\ \beta_2 p_1 & 0 & \beta_3 p_2 & \ldots & \beta_n p_2 \\ \beta_3 p_1 & \beta_3 p_2 & 0 & \ldots & \beta_n p_3 \\ \vdots & \vdots & & \ddots & \vdots \\ \beta_n p_1 & \beta_n p_2 & \beta_n p_3 & \ldots & 0 \end{pmatrix}$$

Enfin, la matrice $A \in \mathbb{R}^{n \times nm}$ est composée des vecteurs transposés A_j, dont les coefficients sont ordonnés comme précédemment. Ainsi, les éléments (i, j) du vecteur A_i sont égaux à 1 quand la tâche à la position $[r]$ correspond à la tâche i sur la machine j et 0 sinon.

Clairement, par construction, la fonction économique de (PO) n'est pas convexe, son hessien n'étant pas semidéfini positif. Les différentes idées que nous avons développées consistent en une reformulation de l'objectif dans le but d'obtenir un nouveau problème quadratique en 0-1 avec un objectif

convexe. Nous allons donc présenter trois différentes approches :
- une convexification simple mais efficace de Skutella
- une convexification par QCR
- une reconvexification du problème transformé par Skutella, par la méthode QCR puis par EQCR.

7.3.2 La convexification de Skutella

La reformulation développée par Skutella [129] consiste à modifier les termes diagonaux de la matrice Q de manière à la rendre semidéfinie positive. Ainsi, l'objectif est réécrit comme suit :

$$p_\gamma(x) = (1-\gamma).c^t x + \frac{1}{2}x^t \left(Q + 2\gamma Diag(c)\right) x.$$

où $\gamma \in \mathbb{R}$. Clairement, cette nouvelle fonction perturbée est égale à la fonction initiale $p(x)$ pour tout x solution réalisable de (PO). En revanche, quand $x \in [0,1]^{nm}$, alors $p_\gamma(x) \leq p(x)$. Cette transformation est spécifique au problème $R||\sum_i \beta_i C_i$: il utilise les coefficients linéaires de l'objectif.

Skutella montre que pour toute valeur de γ supérieure ou égale à $\frac{1}{2}$, alors $p_\gamma(x)$ est convexe. Une idée de preuve consiste à poser tout d'abord $\gamma = \frac{1}{2}$ et à considèrer ainsi la matrice $Q + Diag(c)$. Le *ième* bloc de cette matrice $Q + Diag(c)$ s'écrit :

$$\begin{pmatrix} \beta_1 p_1 & \beta_2 p_1 & \beta_3 p_1 & \dots & \beta_n p_1 \\ \beta_2 p_1 & \beta_2 p_2 & \beta_3 p_2 & \dots & \beta_n p_2 \\ \beta_3 p_1 & \beta_3 p_2 & \beta_3 p_3 & \dots & \beta_n p_3 \\ \vdots & \vdots & & \ddots & \vdots \\ \beta_n p_1 & \beta_n p_2 & \beta_n p_3 & \dots & \beta_n p_n \end{pmatrix}$$

Cette matrice peut être montrée semidéfinie positive puisque $\frac{\beta_1}{p_1} \geq \dots \geq \frac{\beta_n}{p_n}$. Par exemple, si l'on considère le déterminant du mineur principal

$$\begin{pmatrix} \beta_1 p_1 & \beta_2 p_1 \\ \beta_2 p_1 & \beta_2 p_2 \end{pmatrix}$$

alors il est positif si et seulement si $\beta_1 p_1 \beta_2 p_2 \geq \beta_2^2 p_1^2$ et donc si et seulement si $\frac{\beta_1}{p_1} \geq \frac{\beta_n}{p_n}$. Par ailleurs, lorque $\gamma > \frac{1}{2}$, la matrice $Q + \gamma Diag(c)$ n'est autre que la somme de deux matrices semidéfinies positives : $Q + Diag(c)$ et $(\gamma_1)\, Diag(c)$.

En particulier, en posant $\gamma = \frac{1}{2}$, la borne inférieure obtenue par relaxation continue du problème transformé par Skutella sera maximale. Ainsi, le nouveau programme quadratique en 0-1 à résoudre est :

$$\begin{array}{rlll} (SkPO) \ : & \min & p_{sk}(x) \ = \dfrac{1}{2}x^t \left(Q + Diag(c)\right) x + \dfrac{1}{2}c^t x & \\ & \text{s.c.} & Ax \ = \ \mathbf{1} & \forall i \in J \\ & & x \in \{0,1\}^{nm} & \end{array}$$

Cette transformation est très intéressante car elle permet de trouver une borne inférieure de bonne qualité en très peu de temps puisqu'une simple modification des termes diagonaux est requise. Cette reformulation n'a jamais été testée dans la litérature pour la résolution exacte de (PO). On rappelle que Skutella a utilisé sa reformulation dans le but de trouver une solution approchée de la solution optimale. Nous allons présenter quelques résultats sur des instances générées aléatoirement.

Génération des instances :

Nous avons testé l'approche de Skutella pour de nombreuses instances, dont les données $(p_{ij}, \beta_i, \bar{t_i}, \dots)$ sont toutes générées aléatoirement selon la loi uniforme dans l'intervalle $[1, 100]$. Pour chaque paire (n, m), nous avons résolu 10 instances. Les tableaux suivants ne présentent que les moyennes pour 10 instances.

Légende des tableaux :

- n nombre de tâches
- m nombre de machines parallèles
- CPU, temps de résolution requis pas $CPLEX9$ pour résoudre $(SkPO)$
- $\#n\text{œ}uds$, nombre de nœuds dans l'arborescence de recherche
- gap, écart relatif entre la valeur optimale de $(SkPO)$ et la valeur optimale de sa relaxation continue

TAB. 7.22 – Convexification de Skutella pour $R||\sum_i \beta_i C_i$.

n	m	CPU	$\#n\text{œ}uds$	$gap(\%)$
10	2	0.01"	2.5	1.36
	4	0.01"	6.2	3.49
	6	0.02"	12.7	6.34
	8	0,02"	14.9	10.23
30	2	0.02"	4	0.21
	4	0.06"	21.2	1.08
	6	0.08"	49.7	1.74
	8	0.13"	76	3.16
50	2	0.04"	6.2	0.09
	4	0.11"	36.4	0.52
	6	0.21"	75.9	0.84
	8	0.40"	135.7	1.55
60	2	0.05"	6.7	0.06
	4	0.17"	52.9	0.37
	6	0.39"	120.5	0.71
	8	0.51"	113	1.29
100	2	0.11"	8.5	0.02
	4	0.47"	64.4	0.13
	6	1.32"	195.2	0.34
	8	1.89"	260.5	0.50
150	2	0.20"	3.6	0.01
	4	0.81"	38.1	0.05
	6	2.25"	146.4	0.14
	8	5.48"	429.4	0.29
200	2	0.31"	2.8	0.01
	6	1.62"	46.4	0.03
	6	3.78"	147.1	0.08
	8	9.99"	507.1	0.14

Pour le moment, nous n'avons résolu que les instances avec au plus 200 tâches et 8 machines. On remarque que les temps de résolution sont très rapides, même s'ils augmentent avec le nombre de tâches : le temps CPU le plus long requis pour la résolution du branch-and-bound est de 10". Maintenant, il serait intéressant de tester la convexification de Skutella pour les problèmes à 100 tâches et 20 machines (voire plus) pour pouvoir se comparer aux meilleurs résultats de la littérature, obtenus par Chen et Powell [38].

7.3.3 Convexification/Reconvexification par QCR

Nous allons maintenant nous intéresser à la méthode QCR et développer deux approches différentes, basées sur notre reformulation. La première consiste tout simplement à reformuler (PO) selon QCR. On montrera théoriquement que la borne inférieure obtenue par relaxation continue du problème ainsi modifié est meilleure que celle de $(SkPO)$. Ensuite, on comparera les résultats obtenus entre une approche quelque peu différente mais utilisant les mêmes techniques : la "reconvexification" de $(SkPO)$ par QCR.

La reformulation QCR

Si l'on applique la reformulation QCR au problème (PO), $p(x)$ devient :

$$p_{\alpha,u}(x) = p(x) + \left(\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{l=1}^{n}\alpha_{jkl}x_{kl}\right)\left(\sum_{i=1}^{n}x_{ij}-1\right) + \sum_{i=1}^{n}\sum_{j=1}^{m}u_{ij}\left(x_{ij}^2 - x_{ij}\right)$$

et le problème semidéfini à résoudre est :

$$\begin{array}{lll}
(SDPO): & \min & Q \bullet X + c^t x \\
& \text{s.c.} & X_{ijij} = x_{ij} \qquad i=1,\dots,n; j=1,\dots,m \qquad \leftarrow u_{ij}^* \\
& & \displaystyle\sum_{i=1}^{n} X_{ijkl} - x_{kl} = 0 \qquad l,j=1,\dots,m; k=1,\dots,n \qquad \leftarrow \alpha_{jkl}^* \\
& & \displaystyle\sum_{j=1}^{m} x_{ij} = 1 \quad i=1,\dots,n \\
& & \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 \\
& & x \in \mathbb{R}^{nm}, X \in S_{nm}
\end{array}$$

On peut montrer théoriquement que la borne inférieure obtenue par relaxation continue de (PO_{α^*,u^*}) est meilleure que celle de Skutella. Pour cela, on se base sur les mêmes schémas de preuve que pour la section 4.4.

Proposition 13 La solution optimale γ^* est obtenue en résolvant le problème :

$$(p_1) \quad : \max_{\substack{\gamma \geq \frac{1}{2} \\ Q_\gamma \succeq 0}} \left\{ L_1(\gamma) = \min_{\substack{Ax=\mathbf{1} \\ 0 \leq x \leq 1}} \left\{ F_1(x) = (1-\gamma).c^t x + \frac{1}{2}x^t\left(Q + 2\gamma Diag(c)\right)x \right\} \right\}$$

et la solution (α^*, u^*) est obtenue en résolvant le problème suivant :

$$(p_2) \ : \max_{\substack{\alpha\in\mathbb{R}^{nnm}, u\in\mathbb{R}^{nm} \\ Q_{\alpha,u}\succeq 0}} \left\{ L_2(\alpha,u) = \min_{\substack{Ax=\mathbf{1} \\ 0\leq x\leq 1}} \{F_2(x) = p_{\alpha,u}(x)\} \right\}$$

La valeur optimale de (p_1) est inférieure ou égale à celle de (p_2).

Preuve 16 *Soit γ une solution réalisable de (p_1). Si on pose $\tilde{\alpha} = O_{nm}$ (avec O_{nm} la matrice nulle de dimension $nm \times nm$) et $\tilde{u_{ij}} = \gamma \ \forall i,j$ alors $(\tilde{\alpha}, \tilde{u})$ est une solution réalisable de (p_2) puisque $F_1(x)$ et $F_2(x)$ ont précisément le même hessien, semidéfini positif d'après les hypothèse de (p_1). De plus, on a bien $F_1(x) = F_2(x)$ et donc $L_1(\gamma) = L_2(\alpha, u)$* □

Nous avons résolu les instances générées en section 7.3.2. Les résultats sont présentés dans le tableau suivant (ici, CPU_{CSDP} représente le temps requis par $CSDP$ pour résoudre $(SDPO)$ et le signe '-' représente les instances non résolues par le logiciel $CSDP$ du fait de leur trop grande taille) :

TAB. 7.23 – Convexification par QCR pour $R||\sum_i \beta_i C_i$.

n	m	CPU_{CSDP}	CPU	$\#n\text{œ}uds$	$gap(\%)$
10	2	0.41"	0.01"	2.6	0.47
	4	2.02"	0.05"	11.5	1.44
	6	6.28"	0.12"	48.6	3.24
	8	50.20"	2.67"	245	4.17
30	2	27.49"	0.09"	12.2	0.16
	4	8'8"	1.22"	214.6	0.57
	6	27'35"	10.50"	1840.4	1.23
	8	-	-	-	-
50	2	6'46"	0.35"	10.8	0.04
	4	1h38'41"	74.31"	3105.8	0.24
	6	-	-	-	-
	8	-	-	-	-
60	2	22'	0.70"	26.3	0.03
	4	-	-	-	-
	6	-	-	-	-
	8	-	-	-	-

Comme nous l'avons démontré théoriquement, les bornes inférieures obtenues par relaxation continue de $(SkPO)$ sont toutes inférieures à celles obtenues par relaxation continue de (PO_{α^*,u^*}). Par exemple, pour les instances à 10 tâches et 8 machines parallèles, on passe d'un saut d'intégrité de 10.23% pour la reformulation de Skutella à 4.17% avec QCR. Ainsi, l'approche QCR

se révèle très intéressante car, comme pour le problème d'investissements étudié dans la section précédente, on pourrait naturellement espérer que les temps de résolution de (PO_{α^*,u^*}) soient plus rapides que ceux obtenus par la convexication de Skutella. Malheureusement, le logiciel $CSDP$ ne nous permet pas cet optimisme. En effet, les temps de résolution du programme $(SDPO)$ sont beaucoup trop longs comparés à la résolution exacte (banch-and-bound) : entre 0.41" pour les plus petits instances et 1h38' pour les plus grande. Ce qui augmente considérablement le temps de résolution global des instances considérées. De plus, nous ne sommes pas capables de résoudre les instances à 30 tâches et 8 machines, 50 tâches et plus de 6 machines, tout comme les instances à 60 tâches et plus de 4 machines à cause de la mémoire requise par le logiciel $CSDP$ pour la résolution. Comme nous l'avons vu dans la section 7.1.2, le logiciel $CSDP$ est contraint par la taille des problèmes semidéfinis à résoudre. Ainsi, les instances de très grande taille ne peuvent pas se résoudre efficacement avec le logiciel $CSDP$.

Suite à ces résultats non satisfaisants en terme de temps de calcul, nous avons testé l'approche QCR sur le problème reformulé par Skutella. Les résultats ne sont pas présentés ici car les temps de résolution des programmes semidéfinis sont également très longs, la taille des instances à résoudre restant la même. Il faut noter que pour toutes les instances considérées, les bornes obtenues par relaxation continue de $(SkPO_{\alpha^*,u^*})$ sont égales à celle des problèmes (PO_{α^*,u^*}).

7.3.4 Reconvexification par une reformulation inspirée de EQCR, une approche plus intéressante

Aux vues des premiers résultats obtenus avec l'approche QCR, on peut se demander s'il ne serait pas plus intéressant d'appliquer une approche plus simple à mettre en œuvre dans le but d'améliorer les temps CPU de prétraitement, tout en acceptant une détérioration des bornes inférieures obtenues par relaxation continue.

Nous avons choisi d'appliquer EQCR au problème $(SkPO)$. On peut montrer facilement qu'appliquer EQCR à (PO) directement donne une moins bonne relaxation continue que celle de $(SkPO)$. De la même manière, quand on applique EQCR à (PO) au lieu de QCR, la relaxation obtenue est moins bonne (section 4.4).

Si l'on applique la reformulation EQCR au problème $(SkPO)$, $p_{sk}(x)$ devient :

$$p_{sk\beta}(x) = p_{sk}(x) + \beta \left(\sum_{j=1}^{m} \sum_{i=1}^{n} x_{ij} - 1 \right)^2 - \lambda_{\min}(Q_\beta) \sum_{i=1}^{n} (x_i^2 - x_i)$$

Le paramètre β optimal, noté β^*, peut se calculer grâce au programme semidéfini suivant :

$$\begin{array}{lll} (SDPO_2): & \min & Q \bullet X & \leftarrow \lambda_{\min}(Q_{\beta^*}) \\ & \text{s.c.} & & \\ & & tr(X) = 1 & \\ & & A^t A \bullet X = 0 & \leftarrow \beta^* \\ & & X \succeq 0 & \end{array}$$

où A représente la matrice des contraintes d'égalité de $(SkPO)$ (et donc aussi (PO)). Cependant, d'après la section 4.2, on sait pour la borne inférieure obtenue par relaxation continue de :

$$(PO_\beta) \ : \min \left\{ p_{sk\beta}(x) : Ax = b, x \in \{0,1\}^{nm} \right\}$$

est croissant en fonction de β. Ainsi, un algorithme très simple nous permet de ne pas résoudre le programme semidéfini $(SDPO_2)$. Il consiste à fixer au préalable une valeur de β (notée β_1) puis à calculer itérativement les valeurs propres de Q_β pour des valeurs croissantes de $\beta > \beta_1$. On arrête l'algorithme quand la différence de deux valeurs propres consécutives est nulle ou inférieure à 10^{-3}. Généralement, en fixant la première valeur de β très grande (par exemple $\beta_1 = 1000$), une seule itération est nécessaire.

Nous avons résolu les instances générées en section 7.3.2. Les résultats sont présentés dans le tableau suivant :

TAB. 7.24 – Reconvexification par EQCR pour $R||\sum_i \beta_i C_i$.

n	m	CPU	$\#n\text{œ}uds$	$gap(\%)$	$\%gain$
10	2	0.01"	2.4	1.31	0.0495
	4	0.02"	6	3.41	0.0819
	6	0.02"	12.6	6.23	0.1174
	8	0,028	15,3	10.14	0.0910
30	2	0.03"	3.8	0.21	0.0032
	4	0.05"	21	1.08	0.0059
	6	0.10"	49.7	1.73	0.0097
	8	0.17"	72.2	3.16	0.0062
50	2	0.04"	6.2	0.09	0.0006
	4	0.12"	32.8	0.52	0.0014
	6	0.26"	77.4	0.84	0.0012
	8	0.53"	140.1	1.55	0.0014
60	2	0.05"	6.6	0.06	0.0002
	4	0.19"	53.3	0.37	0.0007
	6	0.47"	120.7	0.70	0.0004
	8	0.66"	110	1.29	0.0005
100	2	0.11"	8.7	0.02	0.0001
	4	0.56"	66	0.13	9.6×10^{-05}
	6	1.54"	199.7	0.34	0.0001
	8	2.44"	267.1	0.50	0.0001
150	2	0.24"	3.9	0.01	1.6×10^{-05}
	4	0.96"	37.3	0.05	1.6×10^{-05}
	6	2.98"	157	0.13	1.8×10^{-05}
	8	7.05"	415.5	0.29	3.5×10^{-05}
200	2	0.36"	2.2	0.01	5.7×10^{-06}
	4	1.65"	29.5	0.03	6.5×10^{-06}
	6	5.36"	151.3	0.08	7.3×10^{-06}
	8	15.59"	545.5	0.14	1.2×10^{-05}

La reconvexification par EQCR permet d'améliorer les bornes obtenues par relaxation continue. Bien sûr, l'amélioration est moins importante qu'avec la reformulation QCR. On peut même considérer qu'elle est "minime" puisque le gain moyen des sauts d'intégrité est de $10^{-5}\%$. Cependant, ce résultat est tout de même intéressant car il permet de ne pas utiliser la programmation semidéfinie pour résoudre le problème. Et on peut espérer que pour les instances plus difficiles à résoudre, la reconvexification par EQCR est bien plus efficace. C'est donc un travail en cours que nous allons réaliser avec Yasmin Rios : génération d'instances plus difficiles à résoudre et comparaison entre la convexification de Skutella et la reconvexification EQCR.

Enfin, une remarque assez étonnante est à faire concernant les temps de

résolution et le nombre de nœud moyen associé. En effet, malgré des bornes inférieures plus grande pour la reconvexification EQCR, le CPU requis par la branch-and-bound peut se révéler plus long.

7.4 Conclusion

Ce chapitre d'expérimentations montre que notre méthode de convexification QCR peut s'étendre aux cas de problèmes d'optimisation avec un objectif convexe. Quels que soient les problèmes que nous avons traité, QCR permet d'améliorer significativement la borne inférieure obtenue par relaxation continue. L'idée est ici de comparer la résolution directe d'un problème déjà convexe par un solveur de programmation quadratique convexe avec la résolution du problème reconvexifié par QCR. Par exemple, pour le problème d'investissements, soumettre le problème reformulé est, dans certains cas, plus intéressant puisque les temps de calcul requis par la résolution en 0-1 peuvent être nettement améliorés.

Comme pour le cas des problèmes d'optimisation non convexes, QCR peut se révéler inefficace lorsque le pré-traitement est limité par les logiciels SDP. Dans ce chapitre, nous ne pouvons pas résoudre de manière efficace le problème du plus court chemin probabiliste avec la méthode QCR puisque les tailles des relaxations SDP sont beaucoup trop grandes et donc non résolvables par les logiciels que nous utilisons.

Pour un problème d'ordonnancement déjà convexifié par une modification simple du hessien, QCR se révèle trop coûteuse en temps de calcul. Dans ce cas, une variante de QCR est une bonne alternative : on améliore des temps de résolution tout en acceptant une détérioration de la borne inférieure.

Chapitre 8

Conclusion

Dans ce travail de thèse, nous nous sommes intéressés à la résolution exacte de problèmes d'optimisation en variables 0-1, comportant une fonction quadratique soumise à des contraintes linéaires :

$$(Q01) \ : \min \left\{ q(x) : Ax = b, A'x \leq b', x \in \{0,1\}^n \right\}$$

Depuis de nombreuses années, des méthodes efficaces ont été proposées et sans cesse améliorées pour résoudre des problèmes d'optimisation quadratiques convexes en variables entières, et comme nous l'avons vu, plusieurs solveurs standards, dans leurs versions récentes, mettent en œuvre ces méthodes. Malheureusement, la plupart des problèmes $(Q01)$ sont non convexes. Autrement dit, leur relaxation continue n'est pas un problème convexe.

Le but de notre travail a donc été de proposer des méthodes de résolution exacte de $(Q01)$ passant par différentes reformulations du problème. L'idée principale est ici de reformuler $(Q01)$ en un nouveau problème quadratique en variables 0-1 et dont la fonction objectif est convexe. Le problème ainsi modifié peut être résolu grâce aux solveurs standards. Une autre approche a été étudiée : elle consiste à transformer $(Q01)$ en un nouveau problème linéaire en variables mixtes sous contraintes linéaires, qui peut ensuite être soumis à des solveurs standards de programmes linéaires en variables mixtes.

La principale approche exposée dans ce manuscrit est basée sur une reformulation quadratique convexe de $(Q01)$: la fonction objectif $q(x)$ est reformulée en une nouvelle fonction quadratique $q'(x)$, convexe et égale à $q(x)$ pour toute solution admissible de $(Q01)$. Cette reformulation est déterminée

grâce à la résolution d'une relaxation semidéfinie du problème initial : on "capture" les valeurs optimales de cette relaxation pour les intégrer ensuite dans notre reformulation. L'idée est ici d'obtenir une borne inférieure obtenue par relaxation continue la plus fine possible. Ensuite, le programme peut être soumis à un solveur MIQP. Nous avons choisi d'utiliser, pour la résolution exacte, les solveurs implémentant des schémas de séparation et d'évaluation progressive et dont la procédure de bornes est fondée sur la valeur optimale de la relaxation continue du problème considéré. Bien sûr, d'autres approches utilisant notre reformulation pourraient être envisageables. Nous avons appelé notre nouvelle méthode *QCR*, pour Quadratic Convex Reformulation.

Dans le but d'utiliser très facilement notre méthode de résolution exacte, nous avons implémenté un logiciel, facile d'utilisation et qui permet d'appliquer QCR à tout problème d'optimisation quadratique en variables 0-1 sous contraintes linéaires.

Expérimentalement, la reformulation QCR, suivie d'une résolution par un logiciel standard, s'est révélée très efficace pour deux problèmes d'optimisation dans les graphes. Pour le problème du k-cluster, nous arrivons à résoudre des instances jusqu'ici jamais résolues dans la littérature et pour le problème de bipartition, les performances de notre méthode générale sont comparables à des méthodes très récentes, spécifiques à ce problème.

Des prémices de QCR ont également été étudiés (chapitre 4) : des méthodes déjà développées dans la littérature (celle de la plus petite valeur propre [72] et IQCR [18]) ainsi qu'une variante de QCR, notée EQCR. Ces méthodes consistent également en une reformulation de l'objectif. Alors que la méthode de la plus petite valeur propre consiste tout simplement à soustraire à chaque coefficient diagonal du hessien de (Q01) sa plus petite valeur propre, les reformulations IQCR et EQCR requièrent la résolution de programmes semidéfinis.

Comparé aux autres variantes, QCR a été montrée, théoriquement, la meilleure convexification possible en terme de calculs de bornes. Expérimentalement, les tests effectués sur le problème du k-cluster (section 6.2), problème non convexe, montrent clairement que le problème reformulé par QCR est une meilleure relaxation que ceux modifiés par les approches de la plus petite valeur propre, EQCR et IQCR. Cependant, ces variantes peuvent avoir un intérêt pratique pour les problèmes de très grande taille. Alors que QCR

est efficace pour les problèmes non convexes de bipartition ou du k-cluster (amélioration des résultats obtenus jusqu'à ce jour dans la littérature), la résolution du programme semidéfini pour le problème de la minimisation des échanges d'outils par exemple (section 6.4) est trop coûteuse en temps de calcul. Dans ce cas, les autres reformulations quadratiques convexes peuvent se révéler de bonnes alternatives : on améliore les temps de résolution tout en acceptant une détérioration de la borne inférieure.

De plus, QCR peut aussi s'étendre au cas des problèmes avec un objectif déjà convexe. Elle permet d'améliorer significativement la borne inférieure obtenue par relaxation continue, ce qui est tout à fait novateur. Par exemple, pour le problème d'investissements étudié en section 7.2, soumettre le problème reformulé par QCR est, dans certains cas, plus intéressant que de soumettre le problème initial directement à un solveur MIQP : les temps de calcul requis par la résolution en 0-1 peuvent être améliorés.

Parallèlement à la programmation quadratique convexe, nous avons également proposé une autre approche, basée cette fois-ci sur la programmation linéaire (chapitre 5). Cette dernière a été appelée *reformulation linéaire compacte positive*. Ici, on "capture" les valeurs optimales de la relaxation produit développée par Sherali et al.[126] afin de trouver une bonne reformulation linéaire compacte, c'est-à-dire ne comportant pas trop de variables ni de contraintes, et de soumettre le problème ainsi modifié à un solveur MIP.

De nombreuses questions peuvent maintenant se poser quant aux possibles extensions de notre méthode QCR. Tout d'abord, nous allons nous pencher sur la nature des contraintes de $(Q01)$. Dans notre reformulation, seules les contraintes d'égalité (en plus des contraintes d'intégrité) sont utilisées : pourrait-on améliorer encore QCR en lui intégrant les contraintes d'inégalité dans le processus de reformulation ? Pour une contrainte linéaire d'inégalité dont tous les coefficients sont entiers, il est possible de répondre positivement à cette question par la transformation suivante : on ajoute une variable d'écart positive v, dont on peut facilement calculer une borne supérieure $\bar{v}$, qui est un entier. On peut ensuite remplacer v par une décomposition en puissances de 2 de $\bar{v}$, qui fait intervenir de l'ordre de $\lceil log_2 \bar{v} \rceil$ nouvelles variables binaires. Une des perspectives de ce travail est d'une part de tester cette transformation et d'autre part, d'envisager une méthode qui

travaille directement sur la contrainte d'inégalité, sans passer par la transformation ci-dessus.

Un des intérêts de pouvoir travailler sur les contraintes d'inégalité est que l'on pourrait ainsi intégrer des coupes (étude polyédrale) et améliorer les bornes inférieures obtenues par relaxation continue. Par exemple, pour le problème de la bipartition de graphe (section 6.3), Karisch et al. [90] intègrent dans leur relaxation semidéfinie des inéquations spécifiques à leur formulation. On pourrait utiliser le même procédé de manière à déterminer une borne inférieure obtenue par relaxation continue plus fine. Les coupes qu'ils utilisent pourraient également être intégrées dans le problème reformulé après la résolution de la relaxation semidéfinie.

Toujours en ce qui concerne les contraintes de ($Q01$), le cas des programmes en 0-1 avec une fonction quadratique sous contraintes quadratiques n'a pas été traité. L'extension de QCR pourrait être envisageable si les contraintes quadratiques sont convexes et si elles sont des contraintes d'égalité. Si les contraintes ne sont pas convexes, un autre pré-traitement pourrait consister à les convexifier.

Nous avons montré, dans ce travail, que QCR permet d'ajouter toutes les fonctions nulles quadratiques sur le domaine admissible pour l'optimisation d'une fonction quadratique en variables 0-1 et soumis à une contrainte de cardinalité. Peut-être pourrions-nous trouver d'autres familles de fonctions nulles pour certaines classes de problèmes. L'idée est donc ici d'améliorer la borne obtenue par relaxation continue.

Enfin, une autre perpective porterait sur la nature des variables de ($Q01$). En effet, QCR ne traite que les problèmes en variables bivalentes mais notre approche pourrait facilement s'étendre aux problèmes en nombres entiers dans le cas où les variables entières positives y_i sont bornées. Ces variables peuvent être remplacées par une décomposition en puissance de 2, qui fait intervenir des variables 0-1. Une extension serait également envisageable dans le cas des variables mixtes.

Bibliographie

[1] W.P. Adams, A. Billionnet, and A. Sutter. Unconstrained 0-1 otpimization and lagragean relaxation. *Discrete Applied Mathematics*, 29 :131–142, 1990.

[2] W.P. Adams, R. Forrester, and F. Glover. Comparisons and enhancement strategies for linearizing mixed 0-1 quadratic programs. *Discrete Optimization*, 1(2) :99–120, 2004.

[3] W.P. Adams and R.J. Forrester. A simple recipe for concise mixed 0-1 linearizations. *Operations Research Letters*, 33 :55–61, 2005.

[4] W.P. Adams and H.D. Sherali. A tight linearization and an algorithm for 0-1 quadratic programming problems. *Management Science*, 32(10) :1274–1290, 1986.

[5] F. Alizader. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimisation*, 5 :13–51, 1995.

[6] K.M. Anstreicher. Eigenvalue bounds versus semidefinite relaxations for the quadratic assignment problem. *SIAM Journal of Optimization*, 11(1) :254–265, 2000.

[7] K.M. Anstreicher and N.W. Brixius. A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming*, 89 :341–357, 2001.

[8] M. Azizoglu and O. Kirca. On the minimization of total weighted flow time with identical and uniform parallel machines. *European Journal of Operational Research*, 113(1) :91–100, 1999.

[9] E. Balas and J.B. Mazzola. Non-linear 0-1 programming : linearization techniques. *Mathematical Programming*, 30 :1–21, 1984.

[10] F. Barahona, M. Grötschel, M. Jünger, and Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36 :493–513, 1988.

[11] E.R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal on Algebraic and Discrete Mathematics*, 3 :541–550, 1982.

[12] E.R. Barnes, A. Vanelli, and J.Q. Walker. A new heuristic for partitioning the nodes of a graph. *SIAM Journal on Discrete Mathematics*, 1 :299–305, 1988.

[13] J.E. Beasley. Heuristic algorithms for the unconstrained binary quadratic programming problem. *Technical report, Department of Mathematics, Imperial College of Science and Technology, London, England*, 1998.

[14] H. Belouadah and C.N. Potts. Scheduling identical parallel machines to minimize total weighted completion time. *Discrete Applied Mathematics*, 48(3) :201–218, 1994.

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

[16] A. Billionnet. Optimisation quadratique en variables 0-1. *Optimisation combinatoire, concepts fondamentaux*, pages 191–234, Chapitre de livre. Levoisier. 2005.

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

[18] A. Billionnet and S. Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. *Mathematical Programming, published online*, 2006.

[19] A. Billionnet, S. Elloumi, and M.C. Plateau. QCR : Quadratic convex reformulation for exact solution of 0-1 quadratic programs. *Technical Report CEDRIC*, http ://cedric.cnam.fr/PUBLIS/RC856.pdf, 2005.

[20] A. Billionnet, S. Elloumi, and M.C. Plateau. Quadratic 0-1 programming : tightening linear or quadratic convex reformulation by use of relaxations. *RAIRO*, accepté, 2005.

[21] A. Billionnet, S. Elloumi, and M.C. Plateau. Quadratic convex reformulation : a computational study of the graph bisection problem. *Technical Report CEDRIC*, http ://cedric.cnam.fr/PUBLIS/RC1003.pdf, 2005.

[22] A. Billionnet and A. Faye. A lower bound for a constrained quadratic 0-1 minimization problem. *Discrete Applied Mathematics*, 74(0) :135–146, North–Holland, 1997.

[23] A. Billionnet and E. Soutif. An exact method based on lagrangian decomposition for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 157(3) :565–575, 2004.

[24] A. Billionnet and E. Soutif. Using a mixed integer programming tool for solving the 0-1 quadratic knapsack problem. *INFORMS Journal of Computing*, 16 :188–197, 2004.

[25] A. Billionnet and A. Sutter. Minimization of a quadratic pseudo-boolean function. *European Journal of Operational Research*, 78 :106–115, 1994.

[26] R.B. Boppana. Eigenvalues and graph bisection : an average case analysis. *Proceeding of the 28th annual symposium on computer sciences, IEEE London*, pages 280–285, 1987.

[27] S. Borchers. CSDP, a c library for semidefinite programming. *Optimization methods and Software*, 11(1) :613–623.

[28] S. Borchers. CSDP software. *https ://projects.coin-or.ord/Csdp/*.

[29] E. Boros and P.L. Hammer. Pseudo-boolean optimization. *Technical Report TR : 2001-33, DIMACS*, 2001.

[30] L. Brunetta, M. Conforti, and G. Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78 :243–263, 1997.

[31] J.L. Bruno, E.G. Goffman, and R. Sethi. Scheduling independant tasks to reduce mean finishing time. *Comm. ACM*, 17 :382–387, 1974.

[32] T.N. Bui and B.R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 45 :841–855, 1995.

[33] A. Caprara, P. Pisinger, and P. Toth. Exact solutions method on the quadratic knapsack problem. *Informs Journal on Computing*, 11(2) :125–137, 1999.

[34] P. Carraresi and F. Malucelli. A new lower bound for the quadratic assignment problem. *Operations Research*, 40(Suppl. No1) :S22–S27, 1992.

[35] M.W. Carter. The indefinite zero-one quadratic problem. *Discrete Applied Mathematics*, 7 :23–44, 1984.

[36] C.T. Chang and C.C. Chang. A linearization method for mixed 0-1 polynomial programs. *Computers and Operations Research*, 27 :1005–1016, 2000.

[37] P. Chardaire and A. Sutter. A decomposition method for quadratic zero-one programming. *Management Science*, 41(4) :704–712, 1995.

[38] Z.L. Chen and W.B. Powell. Solving parallel scheduling problems by column generation. *INFORMS Journal of Computing*, 11(1) :78–94, 1999.

[39] N. Christofides and P. Brooker. The optimal partitioning of graphs. *SIAM Journal on Applied Mathematics*, 30 :55–69, 1976.

[40] J. Clausen and J. Larsson Träff. Implementation of parallel branch-and-bound algorithms - experiences with the graph partition problem. *Ann. Operations Research*, 33 :331–349, 1991.

[41] Cplex. Ilog cplex 9.0 reference manual. *ILOG CPLEX Division, Gentilly, France*, http ://www.ilog.com/products/cplex, 2004.

[42] Y. Crama, P. Hansen, and B. Jaumard. The basic algorithm for pseudo-boolean programming revisited. *Discrete Applied Mathematics*, 29(2-3) :171–185, 1990.

[43] Y. Crama, A.W.J. Kolen, A.G. Oerlemans, and F.C. Spieksma. Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems*, pages 33–54, 1994.

[44] C.C. de Souza, R. Keunings, L.A. Wolsey, and O. Zone. A new approach to minimising the frontwidth in finite element calculations. *Computer Methods in Applied Mechanics and Engineering*, 111 :323–334, 1994.

[45] C. DeSimone. The cut polytope and boolean quadratic polytope. *Discrete Mathematics*, 79 :71–75, 1989.

[46] M. Deza and M. Laurent. Facets for the cut cone i. *Mathematical Programming*, 56(2) :121–160, 1992.

[47] M. Deza and M. Laurent. Facets for the cut cone ii. *Mathematical Programming*, 56(2) :161–188, 1992.

[48] R. Diekmann, B. Monien, and R. Preis. Using helpful sets to improve graph bisections. *In D.F. Hsu, A.L. Rosenberg, D. Sotteau, ,editors, Interconnection Networks and Mapping and Scheduling Parallel Computations. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 21 :AMS, 57–73, 1995.

[49] R. Djabali. Optimisation non linéaire en variables bivalentes et applications. *Thèse de doctorat, CNAM*, 1998.

[50] W.E. Donath and A.J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Developments*, 66 :420–425, 1994.

[51] S. Elloumi, A. Faye, and E. Soutif. Decomposition and linearization for 0-1 quadratic programming. *Annals of Operations Research*, 99 :79–93, 2000.

[52] J. Falkner, F. Rendl, and H. Wolkowicz. A computational study of graph partitioning. *Mathematical Programming*, 66 :211–240, 1994.

[53] A. Faye. Optimisation mathématique. *Polycopié IIE-CNAM.*

[54] A. Faye and F. Roupin. Partial lagrangian and semidefinite relaxations of quadratics programs. *Rapport technique CEDRIC N°673*, http ://cedric.cnam.fr/PUBLIS/RC673.pdf, 2004.

[55] C.E. Ferreira, A. Martin, C.C de Souza, R. Weismantel, and L.A. Wolsey. The node capacitated graph partitioning problem : a computational study. *Mathematical Programming*, 81 :229–256, 1998.

[56] I. Fischer, G. Gruber, F. Rendl, and R. Sotirov. Computational experiments with a bundle approach for semidefinite cutting plane relaxations of max-cut and equipartition. *Mathematical Programming, Ser. B*, 105 :451–469, 2006.

[57] R. Fortet. Applications de l'algèbre de boole en recherche opérationnelle. *Revue Française d'Automatique d'Informatique et de Recherche Opérationnelle*, 4 :5–36, 1959.

[58] R. Fortet. L'algèbre de boole et ses applications en recherche opérationnelle. *Cahiers du Centre d'Etudes de Recherche Opérationnelle*, 4 :17–26, 1960.

[59] A.M. Frieze and J. Yadegar. On the quadratic assignment problem. *Discrete applied mathematics*, 5 :89–98, 1983.

[60] M. Garey and D. Johnson. Computers and intractibility : a guide to the theroy of np-completeness. *W.H. freeman & Company*, 1979.

[61] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theorical Computer Science*, 1 :237–267, 1976.

[62] P.E. Gill and W. Murray. Numerically stable methods for quadratic programming. *National Physical Laboratories NAC Report No 78, England*, 1977.

[63] F. Glover. Improved linear integer programming formulation of non linear integer problems. *Management Science*, 22 :445–460, 1975.

[64] F. Glover, G.A. Kochenberger, and B. Alidaee. Adaptative memory tabu search for binary quadratic programs. *Management Science*, 44 :336–345, 1998.

[65] F. Glover and E. Woolsey. Further reduction of 0-1 polynomial programming problems to 0-1 linear programming problems. *Operations Research*, 21 :156–161, 1973.

[66] F. Glover and E. Woolsey. Converting a 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 79 :180–182, 1974.

[67] M.X. Goemans. Semidefinite programming in combinatorial optimization. *Mathematical Programming*, pages 143–161, 1997.

[68] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, 5 :287–326, 1979.

[69] S. Gueye. Linéarisation et relaxation lagrangienne pour les problèmes quadratiques en variables binaires. *Thèse de doctorat, Université d'Avignon*, 2002.

[70] S. Gueye and P. Michelon. Miniaturized linearizations for quadratic 0/1 problems. *Annals of Operations Research*, 140 :235–261, 2005.

[71] P. Hammer, P.L. Hansen and B. Simeone. Roof duality, complementation and persistency in quadratic 0-1 optimization. *Mathematical Programming*, 28 :121–155, 1984.

[72] P.L. Hammer and A.A. Rubin. Some remarks on quadratic programming with 0-1 variables. *RAIRO*, 3 :67–79, 1970.

[73] P.L. Hammer and S. Rudeanu. Boolean methods in operations research. *Springer, Berlin*, 1968.

[74] P. Hansen, B. Jaumard, and V. Mathon. Constrained nonlinear 0-1 programming. *ORSA Journal on Computing*, 5 :97–119, 1993.

[75] P. Hansen, B. Jaumard, and C. Meyer. A simple enumerative algorithm fur unconstrained 0-1 quadratic programming. *Technical Report G-2000-59, Les Cahiers du Gerad*, 2000.

[76] C. Helmberg. An interior-point method for semidefinite programming and max-cut bounds. *Ph.D.-thesis, Department of Mathematics, Graz University of Technlogy*, 1994.

[77] C. Helmberg. A c++ implementation of the spectral bundle method. *Manual version 1.1.1*, http ://www.zib.de/helmberg/SBmethod, 2000.

[78] C. Helmberg. Semidefinite programming. *European Journal of Operational Research*, 137(3) :461–482, 2002.

[79] C. Helmberg and F. Oustry. Bundle methods to minimize the maximum eigenvalue function. *Hanbook on Semidefinite Programming. Theory, Algorithms and Applications. Lieven.*, 11, 2000.

[80] C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Math. Programming*, 8(3) :291–315, 1998.

[81] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, pages 673–696, 2000.

[82] C. Helmberg, F. Rendl, R.J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM, Journal of Otimization*, 6(2) :342–361, 1996.

[83] C. Helmberg, F. Rendl, and R. Weismantel. A semidefinite approach to the quadratic knapsack problem. *Journal of Global Optimization*, 4 :197–215, 2000.

[84] R.A. Horn and L.R. Johnson. Matrix analysis. *Cambridge university Press*, 1985.

[85] R.A. Horn and L.R. Johnson. Topics in matrix analysis. *Cambridge university Press*, 1991.

[86] L.D. Iasemidis, P.M. Pardalos, J.C. Sackellares, and D.-S. Shiau. Quadratic binary programming and dynamical system approach to determine the predictability of epileptic seizures. *Journal of Combinatorial Optimization*, 5(1) :9–26, 2001.

[87] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing : an experimental evaluation ; part1, graph partitioning. *Operations Research*, 37(6) :865–892, 1989.

[88] E.L. Johnson, A. Mehrotra, and G.L. Nemhausen. Min-cut clustering. *Mathematical Programming*, 62 :133–151, 1993.

[89] B. Kalantari and A. Bagchi. An algorithm for quadratic zero-one programs. *Naval Research Logistics*, 37 :527–538, 1990.

[90] S.E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *INFORMS Journal on Computing*, 12(3) :177–191, 2000.

[91] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *the Bell System Technical Journal*, 49 :291–307, 1970.

[92] G. Kochenberger and F. Glover. A royal road to combinatorial optimization ? - the 0-1 quadratic programming problem. 2002.

[93] J Krarup, D. Pisinger, and F. Plastria. Discrete location problems with push-pull objectives. *Discrete Applied Mathematics*, 123 :365–378, 2002.

[94] K. Krishnan. Linear programming approaches to semidefinite programming problems. *Thèse de doctorat,*, 2001.

[95] F. Körner. A tight bound for the boolean quadratic optimization problem and its use in a branch and bound algorithm. *Optimization*, 19(5) :711–721, 1988.

[96] G. Laporte, J.J. Salazar Gonzalez, and F. Semet. Exact algorithms for the job sequencing and tool switching problem. *Les cahiers du GERAD*, 2002.

[97] J.B. Lasserre. An explicit exact sdp relaxation for nonlinear 0-1 programs. *Lecture Notes in Computer Science*, 2081 :293, 2001.

[98] C.Y. Lee and R. Uzsoy. A new dynamic programming algorithm for the parallel machines total weighted completion time problem. *Operations Research Letters*, 11(2) :73–75, 1992.

[99] C. Lemaréchal and F. Oustry. Semidefinite relaxations and lagrangian duality with application to combinatorial optimization. *RR-3710, INRIA Rhones-Alpes*, 1999.

[100] C. Lemaréchal and F. Oustry. Sdp relaxations in combinatorial optimization from a lagrangian point of view. *In N. Hadjisavvas and P.M.Pardalos, editors, Advances in Convex Analysis and Global Optimization, Kluwer*, pages 119–134, 2001.

[101] C.E. Lemke. On complementary pivot theory. *Mathematics of Decision Sciences*, pages 95–114, 1968.

[102] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1 :343–362, 1977.

[103] L. Liberti. Linearity embedded in nonconvex programs. *Journal of Global Optimization*, 33(2) :157–196, 2005.

[104] L. Liberti. Compact linearization for bilinear quadratic problems. *4OR*, à paraître.

[105] A. Lisser and F. Rendl. Graph partitioning using linear and semidefinite programming. *Mathematical Programming, Ser. B*, 95 :91–101, 2003.

[106] A. Lodi, K. Allemand, and T.M. Liebling. An evolutionary heuristic for quadratic 0-1 programming. *European Journal of Operational Research*, 119 :662–670, 1999.

[107] H. Markowitz. Portfolio selection. *Journal of Finance*, 7 :77–91, 1952.

[108] R.D. McBride and J.S. Yormark. An implicit enumeration algorithm for quadratic integer programming. *Management Science*, 26(3), 1980.

[109] P. Merz and B. Freisleben. Greedy and local search heuristics for unconstrained quadratic programming. *Journal of Heuristics*, 8(2) :197–213, 2002.

[110] P. Michelon and N. Maculan. Lagrangian decomposition for integer non linear programming with linear constraints. *Mathematical Programming*, 52(2) :303–314, 1991.

[111] P. Michelon, S. Ripeau, and N. Maculan. Un algorithme pour la bipartition d'un graphe en sous-graphes de cardinalité fixée. *RAIRO Operations Research*, 35 :401–415, 2001.

[112] Y. Nesterov and A. Nemirivskii. Interior point polynomial algorithms in convex programming. *SIAM studies in Applied Mathematics, Philadelphia, PA*, 1993.

[113] M. Padberg. The boolean quadratic polytope : some characteristics, facets and relatives. *Mathematical Programming*, 45 :139–172, 1989.

[114] P. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem : a survey of recent developments. *In P.Pardalos and H. Wolkowicz, editors, Quadraric assignment and related problems, DIMACS*

Series in Discrete Mathematics and Theoretical Computer Science, pages 1–42. AMS, 1994.

[115] H. Pirkul and E. Rolland. A lagrangian based heuristic for uniform graph partitioning. *Working paper, A. Gary Anderson Graduate School of Management, University of California, Riverside, USA*, 1996.

[116] D. Pisinger. Exact solution of p-dispersion problems. *Technical Report 99/14, DIKU, University of Copenhagen, Denmark*, 1999.

[117] D. Pisinger. Upper bounds and exact algorithm for p-dispersion problems. *Computers and Operations Research*, 33(5) :1380–1398, 2006.

[118] M.C. Plateau, A. Billionnet, and S. Elloumi. Eigenvalue methods for linearly constrained quadratic 0-1 problems with application to the densest k-subgraph problem. *In 6ème congrès ROADEF, Tours, 14-16 février, Presses Universitaires Francois Rabelais*, pages 55–66, 2005.

[119] 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.

[120] J. Renegar. A mathematical view of interior-point methods in convex optimization. *MPS-SIAM Series on Optimization, SIAM, Philadelphia*, 2001.

[121] E. Rolland, H. Pirkul, and F. Glover. Tabu search for graph partitioning. *Ann. Operations Research*, 63 :209–232, 1997.

[122] C. Roucairol and P. Hansen. Problème de la bipartition minimale d'un graphe. *RAIRO*, 21 :325–348, 1987.

[123] F. Roupin. Logiciel sdp_s. *http ://semidef.free.fr/SDP_S/SDP_S.html.*

[124] F. Roupin. L'approche par programmation semidéfinie en optimisation combinatoire. *bulletin n°13 de la ROADEF*, 2004.

[125] Scilab. http ://scilabsoft.inria.fr.

[126] H.D. Sherali and W.P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems.* Kluwer Academic Publishers, Norwell, MA, 1999.

[127] H.D. Sherali and H. Tuncbilek. A reformulation-convexification approach for solving nonconvex quadratic programming problems. *Journal of Global Optimization*, 7(1) :1–31, 1995.

[128] N.Z. Shor. Quadratic optimization problems. *Soviet Journal of computer Systems Sciences*, 25 :1–11, 1987.

[129] M. Skutella. Semidefinite relaxations for parallel machine scheduming. *IEEE symposium on foundations of computer science*, pages 472–481, 1998.

[130] W.E. Smith. Various optimizers for single-stage production. *Naval research and Logistics Quarterly*, 3 :59–66, 1956.

[131] E. Soutif. Résolution du problème du sac-à-dos quadratique en variables bivalentes. *Thèse de doctorat, CNAM*, 2000.

[132] C.S. Tang and V. Denardo. Models arising from a flexible manufacturing machine. part i : minimizing the number of tool switches. *Operations Research*, 36 :767–777, 1988.

[133] M.J. Todd, K.C. Toh, and R.H. Tutuncu. On the nesterov-todd direction in semidefinite programming. *SIAM Journal on Optimisation*, 8 :769–796, 1998.

[134] L.G. Walters. Reduction of integer polynomial problems to 0-1 linear problems. *Operations Research*, 15 :1171–1174, 1967.

[135] S. Webster. Weighted flow time bounds for scheduling identical processors. *European Journal of Operational Research*, 80(1) :103–111, 1995.

[136] T.J. Ypma. Historical development of the newton-raphson method. *SIAM review*, 37(4) :1171–1174, 1995.

[137] Y. Zhang. On extending some primal-dual interior point algorithms from linear programming to semidefinite programming. *SIAM Journal on Optimization*, 8 :365–386, 1998.

[138] W.X. Zhu. Penalty parameter for linearly constrained 0-1 quadratic programming. *Journal of Optimization Theory and Applications*, 116(1) :229–239, 2003.

Annexe 1 - Rappel sur la programmation semidéfinie et son utilisation en programmation quadratique en 0-1

Un programme semidéfini peut être considéré comme un programme linéaire où les variables sont des matrices semidéfinies positives. Les bonnes propriétés de ces matrices et leur formulation quadratique convexe associée ont fasciné les mathématiciens depuis les années 70. Dans les années 80, des méthodes de points intérieurs ou des méthodes dites de faisceaux ont été développées pour la programmation semidéfinie. Depuis, de nombreux algorithmes ont été proposés et quelques implémentations de très bonne qualité sont maintenant accessibles via internet. La programmation semidéfinie peut ainsi être utilisée comme un outil standard d'optimisation et de ce fait, de nombreux problèmes sont modélisés comme des problèmes semidéfinis.

Cette annexe présente quelques uns des concepts fondamentaux de la programmation semidéfinie (matrices semidéfinies positives, dualité, ...) et donne un aperçu des méthodes de résolution des problèmes semidéfinis les plus efficaces et les plus utilisés.

Les matrices semidéfinies positives

Dans cette section, nous rappelons quelques définitions et propriétés des matrices réelles carrées symétriques, définies dans S_n.

On notera S_+^n l'ensemble des matrices symétriques semidéfinies positives et S_{++}^n l'ensemble des matrices symétriques définies positives.

Définition 4 *Soit A et B des matrices de dimension $n \times m$. On pose $A \bullet B = Tr(A^t B) = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij} b_{ij}$. L'opérateur $\bullet$ définit un produit scalaire dans $\mathbb{R}^{n \times m}$.*

Propriété 3 $M = (m_{ij}), \forall i, j \in \{1, \ldots, n\}^2$ est une matrice semidéfinie positive (notée SDP) si et seulement si une des conditions suivantes est vérifiée [84] [85] :

1. $\forall u \in \mathbb{R}^n$, $u^t M u \geq 0$
2. Les valeurs propres de M sont toutes positives ou nulles.
3. Les déterminants de tous les mineurs symétriques de M sont positifs ou nuls.

On rappelle qu'un mineur symétrique d'une matrice symétrique M est une sous-matrice de M obtenue en supprimant un sous-ensemble de lignes et les colonnes correspondantes. Par exemple :

$\begin{pmatrix} 1 & 1 \\ 1 & 3 \end{pmatrix}$ et $\begin{pmatrix} 1 & 2 \\ 2 & 5 \end{pmatrix}$ sont des mineurs symétriques de $\begin{pmatrix} 1 & 1 & 2 \\ 1 & 3 & 4 \\ 2 & 4 & 5 \end{pmatrix}$.

Il existe un cas particulier : chaque élément de la diagonale d'une matrice est un mineur symétrique.

On peut parfois facilement repérer qu'une matrice M est ou n'est pas semidéfinie positive :

- dès qu'un élément de la diagonale est négatif, la matrice M n'est pas semidéfine positive (ce qui découle du 2. de la propriété 3)
- dès qu'un élément de la diagonale est nul, il faut que les éléments de la ligne et de la colonne correspondantes soient nuls, ou alors la matrice n'est pas semidéfinie positive.

– une matrice symétrique est semidéfinie positive si elle vérifie la *dominance* quadratique, i.e. $m_{ii} \geq \sum_{j=1, j\neq i}^{n} \mid m_{ij} \mid$.

Les programmes semidéfinis et la notion de dualité [124]

Un programme semidéfini peut être défini comme la maximisation d'une fonction linéaire de $X \in S_n$ sous contraintes linéaires :

$$\begin{array}{lll} (SDP): & \min & C \bullet X \\ & \text{s.c.} & \\ & & A_i \bullet X = b_i \quad i = 1, \ldots, m \\ & & X \succeq 0 \end{array}$$

où $b \in \mathbb{R}^m$ et les constantes A_i $(i = 0, \ldots, m)$ sont des matrices carrées réelles de dimension $m \times n$.

Schématiquement, comparé aux programmes linéaires, le vecteur $x \in \mathbb{R}^n_+$ de variables est remplacé par une variable matricielle $X \in S^+_n$. Les programmes semidéfinis paraissent souvent naturels pour les problèmes dont les données sont des matrices.

Le programme dual associé à (SDP) est :

$$\begin{array}{lll} (DSDP_1): & \max & f(x) = b^t y \\ & \text{s.c.} & \\ & & A^t y + Z = C \succeq 0 \\ & & y \in \mathbb{R}^m, Z \succeq 0 \end{array}$$

il peut aussi s'écrire :

$$\begin{array}{lll} (DSDP_2): & \max & f(x) = b^t y \\ & \text{s.c.} & \\ & & F(y) = A^t y - C \succeq 0 \\ & & X \succeq 0 \end{array}$$

La fonction F associe donc à tout vecteur x de $\mathbb{R}^m$ une matrice qui est positive lorsque y est admissible.

On peut ici aisément constater que le programme semidéfini $(DSDP_2)$ est un problème d'optimisation convexe. En effet, F est une fonction linéaire

et si $F(x) \succeq 0$ et $F(y) \succeq 0$ on a $F(\lambda x + (1-\lambda)y) = \lambda F(x) + (1-\lambda)F(y) \succeq 0$ pour tout $0 \leq \lambda \leq 1$.

Ainsi, de nombreux problèmes d'optimisation peuvent se formuler comme des programmes semidéfinis. Plus particulièrement, les programmes quadratiques peuvent facilement s'écrire comme des programmes semidéfinis. Cela découle du thèorème suivant :

Theorème 4 *(Complément de Shur)*
Soient $A \in S_r^{++}$, $B \in S_q$ et $C \in \mathbb{R}^{r \times q}$ alors :

$$\begin{pmatrix} A & C \\ C^t & B \end{pmatrix} \succeq O \Longleftrightarrow B \succeq C^t A^{-1} C$$

Par exemple, suppposons que l'on ait la contrainte quadratique $(B_i x + b_i)^t (B_i x + b_i) - c_i x \leq 0$ où $B \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$ et $c \in \mathbb{R}$ sont donnés. Chaque contrainte a donc la forme suivante : $\|Bx + b\| \leq c^t x$. Ainsi, on a :

$$\begin{pmatrix} I & Bx+b \\ (Bx+b)^t & c^t x \end{pmatrix} \succeq O$$

Ainsi, la contrainte non linéaire est transformée en une contrainte linéaire sur le domaine des matrices semidéfinies positives.

La théorie de la dualité pour les programmes SDP peut fournir des résultats aussi intéressants qu'en programmation linéaire moyennant des hypothèses un peu plus fortes.

Propriété 4 *On a égalité entre les valeurs optimales des deux programmes semidéfinis (SDP) et $(DSDP_2)$ (notées respectivement p^* et d^*) si au moins l'une des conditions suivantes est vérifiée :*

- *Le problème primal est strictement réalisable*
  ($\exists Z$ t.q. $Z = Z^t \succ 0$, $A_i \bullet Z = b_i$ $i = 1, \ldots, m$)
- *Le problème dual est strictement réalisable*
  (il existe des points intérieurs : $\exists x$ t.q. $F(x) \succ 0$).

Si les deux conditions sont satisfaites alors les ensembles optimaux $\{x : F(x) \succeq 0, c^t x = p^*\}$ et $\{Z : Z \succeq 0, A_i \bullet Z = c_i, i = 1, \ldots, m, A_0 \bullet Z = d^*\}$ sont non vides.

Approches lagrangiennes et semidéfinie

Lemarechal et Oustry, dans [99], présentent le lien entre programmes semidéfinis et relaxation lagrangienne partielle d'un programme quadratique en 0-1 (les contraintes linéaires sont maintenues). Ils montrent que dualiser les contraintes d'intégrité reproduit la relaxation semidéfinie classique pour le problème de la coupe maximale et du stable maximal. Le saut de dualité résultant est alors meilleur que celui de la relaxation lagrangienne classique. Ce lien peut être généralisé au cas plus général d'un problème quadratique où x n'est pas nécessairement une variable bivalente [54] : le dual d'une relaxation lagrangienne partielle où les contraintes linéaires sont maintenues peut être également formulé comme un programme semidéfini avec plus de contraintes que dans le cas booléen.

Approches semidéfinies

L'élaboration des méthodes de résolution des programmes semidéfinis est un domaine récent et toujours très actif. De nombreux logiciels sont maintenant mis en ligne, dont les plus connus sont *CSDP* (Borchers [28]) et *SB* (Helmberg [77]). Notons que le modeleur automatique SDP_S, développé par Roupin [123], utilise le logiciel SB.

Ces deux solveurs s'appuient sur des méthodes de points intérieurs pour *CSDP* [27] [82] et sur la méthode des faisceaux pour *SB* [81], [79].

Les méthodes de points intérieurs ($CSDP$)

Les premières méthodes de points intérieurs pour les programmes semidéfinis ont été proposées par [5] et [112]. Pour plus de précisions, les références [76], [133], [137], [94] et [120] sont très complètes.

Les méthodes de points intérieurs sont applicables à tous les programmes semidéfinis, elles exploitent les informations du second ordre et offrent une complexité polynomiale en terme de temps de calcul.

On suppose tout d'abord que la dualité forte est vérifiée pour les problèmes (SDP) et ($DSDP_1$), autrement dit ces deux problèmes sont strictement réalisables. L'algorithme des points intérieurs introduit une fonction

barrière et se base sur la résolution du problème auxiliaire associé :

$$(BSDP): \quad \begin{array}{ll} \min & C \bullet X - \mu \; log \; det(X) \\ \text{s.c.} & \\ & A_i \bullet X = b_i \quad i = 1, \ldots, m \\ & X \succ 0 \end{array}$$

où $\mu > 0$ est le paramètre barrière et $-log\ det(X)$ la fonction barrière. On évolue ainsi à l'intérieur (strictement) du cone des matrices semidéfinies positives. De plus, $-log\ det(X)$ tend vers l'infini quand on s'approche de la frontière de l'ensemble admissible.

La méthode des faisceaux (SB)

Considérons maintenant la méthode des faisceaux, méthode du premier ordre développée par Helmberg [78].

Si l'on suppose que la condition suivante est vérifiée :

$$\exists \text{ un } \tilde{y} \in \mathbb{R}^m \text{ t.q. } A^t \tilde{y} = I \tag{8.1}$$

La méthode des faisceaux se base sur le problème de valeur propre suivant, équivalent à $(DSDP_1)$:

$$\min_y a\lambda_{\max}\left(C - A^t y\right) + b^t y \tag{8.2}$$

L'hypothèse (8.1) revient à supposer que les solutions primales admissibles de (SDP) ont une trace constante, notée a i.e. $tr(X) = a \; \forall X \in \{X \succeq 0, AX = b\}$. Il est important de noter qu'un grand nombre de programmes semidéfinis peuvent s'écrire ainsi.

Par ailleurs, la fonction $\lambda_{\max}\left(C - A^t y\right)$ est une fonction convexe. Les méthodes de faisceaux permettent de donner une solution approchée de la valeur propre maximale.

Annexe 2 - Programmation quadratique convexe continue - l'algorithme du pivot de Lemke

Tout au long de ce travail de thèse, nous avons choisi d'utiliser des solveurs MIQP implémentant des algorithmes de branch-and-bound, avec une procédure de borne fondée sur la valeur optimale de la relaxation continue du problème considéré, c'est-à-dire sur la résolution d'un programme quadratique convexe continu. Nous allons présenter, dans cette annexe, une méthode de résolution exacte pour les programmes quadratiques convexes continus sous contraintes linéaires [53]. Cette approche est basée sur l'algorithme du pivot de Lemke [101], qui repose sur un concept de variables de base et hors base.

Soit le problème quadratique sous contraintes linéaires suivant :

$$(QP) \ : \quad \begin{array}{ll} \min & \frac{1}{2}x^tQx + c^tx \\ \text{s.c.} & Ax \leq b \\ & x \geq 0 \end{array}$$

où Q est une matrice carrée symétrique d'ordre n, c un vecteur à n composantes, A une matrice de dimension $m \times n$ et b un vecteur à m composantes.

Les conditions de Karusch Kuhn et Tucker (KKT) de ce problème ont une structure particulière. Elles se présentent sous la forme d'un système d'équations linéaires avec des contraintes de complémentarité appelé LCP

(Linear Complementary Problem).

On introduit le vecteur y des m variables d'écart du système $Ax \leq b$ de sorte que $Ax + y = b$ avec $y \geq 0$. On note u le vecteur des m multiplicateurs de Lagrange des contraintes $Ax \leq b$ et v le vecteur des n multiplicateurs des contraintes $x \geq 0$.

Les conditions de Kuhn et Tucker du problème quadratique s'écrivent alors :

$$Ax + y = b \tag{8.3}$$
$$-Qx - A^t u + v = c \tag{8.4}$$
$$x^t v = 0, \quad u^t y = 0 \tag{8.5}$$
$$x, y, u, v \geq 0 \tag{8.6}$$

L' équation (8.3) spécifie que x vérifie les contraintes du problème. L'équation (8.4) représente les conditions KKT où u et v sont les multiplicateurs de Lagrange associés. Enfin, la troisième équation (8.5) correspond aux contraintes de complémentarité : le multiplicateur d'une contrainte non saturée est nul.

Si l'on pose maintenant $M = \begin{pmatrix} 0 & -A \\ A^t & Q \end{pmatrix}$, $q = \begin{pmatrix} b \\ c \end{pmatrix}$, $w = \begin{pmatrix} y \\ v \end{pmatrix}$, $z = \begin{pmatrix} u \\ x \end{pmatrix}$, les conditions KKT se mettent sous la forme :

$$w - Mz = q,\ z^t w = 0, \quad z \geq 0, \quad w \geq 0$$

qui est la forme usuelle d'un LCP.

On va maintenant appliquer l'algorithme de Lemke pour résoudre LCP. Pour cela, on introduit une variable supplémentaire $z_0 \geq 0$, dite variable artificielle, que l'on ajoute à chacune des égalités du système : $w - Mz = q + z_0 e$. On note que les solutions admissibles vont évoluer sur les points extrêmes du polyèdre défini par les conditions KKT.

Algorithme de Lemke

L'algorithme du pivot de Lemke se fait à partir du passage d'une base à une autre par une méthode de pivot classique permettant de rentrer une variable en base alors qu'une autre en sort. Les étapes peuvent être représentées clairement par des tableaux où les lignes représentent les variables en base. Ainsi, pivoter est l'opération qui permet de passer d'une base à l'autre. Supposons que l'on soit à une itération quelconque de l'algorithme. Soit d_s la colonne du tableau sous la variable entrante y_s et r l'indice de la ligne de la variable sortante. Pour passer d'une base à l'autre, il suffit de prémultiplier le tableau courant de la matrice par la matrice

$$P = \begin{pmatrix} 1 & 0 & -\frac{d_{1s}}{d_{rs}} & 0 \\ 0 & 1 & & \dots \\ & & \frac{1}{d_{rs}} & \\ & & & 1 \\ & & & \dots \\ & & -\frac{d_{ps}}{d_{rs}} & \end{pmatrix},$$

matrice carrée d'ordre p égale à l'identité à ceci près que la colonne r vaut $-\frac{d_s}{d_{rs}}$ sauf sur la ligne r où le coefficient vaut $\frac{1}{d_{rs}}$. L'élement d_{rs}, qui joue un rôle central, est appelé le pivot.

Au début de l'algorithme, le tableau est, à une permutation des colonnes près, la matrice $(I, -M, -e, q)$ représentant le système $w - Mz - z_0 e = q$ où I est la matrice identité d'ordre p.

Phases d'initialisation

Si $q \geq 0$ alors LCP est résolu en posant $w = q$ et $z = 0$.
Initialiser le tableau avec la solution de base (non réalisable) $w = q$ (en base), $z = 0$, $z_0 = 0$ (hors base).
Déterminer l'indice s réalisant $-q_s = \underset{1 \leq j \leq p}{max}(-q_j)$, pivoter à la ligne s et la colonne z_0 (z_0 entre en base et w_s en sort).
Poser $y_s = z_s$.

Itérations

1. Soit d_s la colonne du tableau sous la variable y_s. Si $d_s \leq 0$, aller en 2. Déterminer l'indice r réalisant $\underset{1 \leq j \leq p}{min} \left\{ \frac{\bar{q}_j}{d_{js}} : d_{js} > 0 \right\}$ où $\bar{q}$ est le membre

droit du tableau.

Pivoter à la ligne r et la colonne y_s (y_s entre en base et la variable à la ligne r en sort).

Si la variable qui sort de base est z_0, aller en 3.

Si la variable qui sort de base est w_l, poser $y_s = z_l$ et aller en 1.

Si la variable qui sort de base est z_l, poser $y_s = w_l$ et aller en 1.

2. STOP LCP est non résolu (sauf si $z_0 = 0$). Dans le cas où Q est semidéfinie positive, on obtient un minimum non borné.

3. STOP LCP est résolu

Annexe 3 - Le logiciel QCR

Dans le cadre de notre travail, nous avons élaboré un logiciel implémentant la reformulation QCR. Cet outil permet d'appliquer de manière simple et efficace notre approche pour tous les problèmes quadratiques sous contraintes linéaires et en variables 0-1 que l'on souhaite tester. On considère donc le problème suivant :

$$
\begin{array}{lll}
(Q01): & \min & q(x) = x^t Q x + c^t x \\
& \text{s.c.} & \\
& & Ax = b \\
& & A'x \leq b' \\
& & x \in \{0,1\}^n
\end{array}
$$

Que l'objectif soit convexe ou non, on va pouvoir appliquer à $(Q01)$ la reformulation QCR. Pour ce faire, l'utilisateur du logiciel doit rentrer toutes les données du problème dans les fichiers correspondants, i.e. :

- la matrice Q de dimension $n \times n$ dans le fichier $q.txt$
- le vecteur c à n composantes dans le fichier $c.txt$
- la matrice A de dimension $m \times n$ dans le fichier $A.txt$
- la matrice A' de dimension $p \times n$ dans le fichier $Abis.txt$
- le vecteur b à m composantes dans le fichier $b.txt$
- le vecteur b' à p composantes dans le fichier $bbis.txt$

Une fois que les fichiers de données sont créés correctement (voir section suivante), deux choix s'offrent à l'utilisateur : ou il décide de résoudre les programmes semidéfinis via le logiciel SB ou alors, il décide de les résoudre via $CSDP$. En effet, pour pouvoir utiliser le logiciel QCR, il faut avoir, au préalable, téléchargé SB ou $CSDP$ au choix pour la résolution des programmes semidéfinis et posséder $AMPL$ et $CPLEX$ ou encore $XPRESS$

pour la résolution exacte via un branch-and-bound. On peut noter que SB, $CSDP$ sont des logiciels gratuits ainsi que $XPRESS$ dans sa version étudiante pouvant se télécharger sur internet aux adresses suivantes :

- http ://www.zib.de/helmberg/SBmethod pour SB
- https ://projects.coin-or.ord/Csdp/ pour $CSDP$
- http ://dashoptimization.com/home/products/evaluation/student_request.html pour $XPRESS$ (version étudiante)

Selon les cas, l'utilisateur doit taper, par exemple, la ligne de commande :

$$./QCR\ s\ c \text{ ou encore } ./QCR\ c\ x$$

La première option correspond aux logiciels SB (s) et $CSDP$ (c). De la même manière, la seconde option représente les deux différents solveurs quadratiques convexes : c pour $CPLEX$ et x pour $XPRESS$.

Le programme général qui lit les données et crée les fichiers que traitent les logiciels semidéfinis et les solveurs quadratiques est implémenté en langage C.

Formats des fichiers

La matrice Q ($q.txt$) :
Représentation de la donnée matricielle Q sous la forme suivante :

n	h	
1	2	q_{12}
1	3	q_{13}
...		
i	j	q_{ij}
...		

où h représente le nombre de termes non nuls de la matrice triangulaire supérieure de Q. On n'inscrit que les couples (i, j) tels que $i < j$. Ainsi, il faut que Q soit mise sous sa forme symétrique : si l'objectif possède le terme $2x_1x_2$ alors dans $q.txt$ on écrira la ligne 1 2 1.

Le vecteur c ($c.txt$) :
Représentation de la donnée vectorielle c sous la forme suivante :

o
c_1
c_2
...
c_n

où $o = 1$ si $q(x)$ possède des termes linéaires et 0 sinon. Dans le cas où c est un vecteur nul, il n'est pas nécessaire de mettre à zéro tous les coefficients c_i. Un simple 0 dans le fichier *c.txt* est suffisant.

Les matrices A et A' (*A.txt* et *Abis.txt*) :

Représentation des données matricielles A et A' sous les formes suivantes :

m	h			p	$hbis$	
1	1	a_{11}	et	1	1	a'_{11}
1	3	a_{13}		1	3	a'_{13}
...				...		

où h et $hbis$ représentent le nombre de termes non nuls des matrices A et A' respectivement.

Les vecteurs b et b' (*b.txt* et *bbis.txt*) :

Représentation des données vectorielles b et b' sous les formes suivantes :

b_1		b'_1
b_2	et	b'_2
...		...
b_m		b'_p

Exemple d'utilisation

Imaginons que l'on veuille trouver la solution optimale du problème (Π), déjà défini au chapitre 4 (*Exemple 1*) :

$$
\begin{aligned}
(\Pi): \quad & \min \quad x^t Q x + c^t x \\
& \text{s.c.} \\
& \qquad Ax = b \\
& \qquad A'x \leq b' \\
& \qquad x \in \{0,1\}^5
\end{aligned}
$$

où c est un vecteur à n composantes nulles, $l = 0$ et

$$P = \begin{pmatrix} 0 & -0.5 & -0.5 & -0.5 & -0.5 \\ -0.5 & 0 & 0 & -0.5 & 0 \\ -0.5 & 0 & 0 & 0 & 0 \\ -0.5 & -0.5 & 0 & 0 & -0.5 \\ -0.5 & 0 & 0 & -0.5 & 0 \end{pmatrix} \not\succeq 0$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 9 & 0 & 9 \end{pmatrix}, b = \begin{pmatrix} 3 \\ 11 \end{pmatrix}.$$

$$A' = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \end{pmatrix}, b' = 1.$$

Clairement, l'objectif du problème (Π) n'est pas convexe, on ne peut donc pas utiliser directement les solveurs quadratiques convexes pour déterminer une solution optimale du problème. Ici, la reformulation QCR est nécessaire.

L'utilisateur doit donc créér les fichiers de données :

q.txt →:

5	6	
1	2	−0.5
1	3	−0.5
1	4	−0.5
1	5	−0.5
2	4	−0.5
4	5	−0.5

, *c.txt* →

0

A.txt →

2	9	
1	1	1
1	2	1
1	3	1
1	4	1
1	5	1
2	1	1
2	2	1
2	3	9
2	5	9

, *b.txt* →

3
11

,

Abis.txt →

1	2	
1	1	1
1	4	1

et *bbis.txt* →

1

.

Après lancement de la commande $./QCR\ s\ c$, on obtient l'affichage sui-

vant :

```
nb non nuls dans C = 7
nb contraintes = 19
nb EQ = 18, nb INEQ = 1
File .sdpa written.

*********************************
RESOLUTION DU PROGRAMME SEMIDEFINI
*********************************

nb non nuls dans C = 7
nb contraintes = 19
nb EQ = 18, nb INEQ = 1
File .sdpa written.

Iter:  0 Ap: 0.00e+00 Pobj:  0.0000000e+00 Ad: 0.00e+00 Dobj:  0.0000000e+00
Iter:  1 Ap: 8.92e-01 Pobj: -2.7328334e+00 Ad: 9.70e-01 Dobj:  4.4475580e+02
Iter:  2 Ap: 9.67e-01 Pobj:  1.5528532e+00 Ad: 9.70e-01 Dobj:  3.8570964e+02
Iter:  3 Ap: 9.68e-01 Pobj:  1.6181907e+00 Ad: 9.68e-01 Dobj:  9.0304872e+01
Iter:  4 Ap: 9.68e-01 Pobj:  1.6042989e+00 Ad: 9.68e-01 Dobj:  1.0109394e+01
Iter:  5 Ap: 9.68e-01 Pobj:  1.6279367e+00 Ad: 9.69e-01 Dobj:  2.3515729e+00
Iter:  6 Ap: 9.68e-01 Pobj:  1.9638425e+00 Ad: 1.00e+00 Dobj:  2.1292053e+00
Iter:  7 Ap: 9.68e-01 Pobj:  1.9946941e+00 Ad: 9.85e-01 Dobj:  2.0051190e+00
Iter:  8 Ap: 9.91e-01 Pobj:  1.9997939e+00 Ad: 9.85e-01 Dobj:  2.0001828e+00
Iter:  9 Ap: 9.97e-01 Pobj:  1.9999936e+00 Ad: 9.87e-01 Dobj:  2.0000037e+00
Iter: 10 Ap: 1.00e+00 Pobj:  1.9999997e+00 Ad: 1.00e+00 Dobj:  1.9999998e+00
Iter: 11 Ap: 9.70e-01 Pobj:  2.0000000e+00 Ad: 9.70e-01 Dobj:  2.0000000e+00
Success: SDP solved
Primal objective value: 2.0050000e+00 <-VALEUR OPTIMALE DU PROGRAMME SEMIDEFINI
Dual objective value: 2.0050000e+00
Relative primal infeasibility: 3.41e-13
Relative dual infeasibility: 4.57e-09
Real Relative Gap: -3.34e-10
XZ Relative Gap: 6.74e-09
DIMACS error measures: 3.55e-13 0.00e+00 1.07e-08 0.00e+00 -3.34e-10 6.74e-09
temps processeur : 0.016001     <- TEMPS DE RESOLUTION DU PROGRAMME SEMIDEFINI

********************
RESOLUTION PAR CPLEX
********************
```

```
ILOG AMPL 9.000, licensed to "cnam-paris".
AMPL Version 20021031 (Linux 2.4.18-14)


5 variables, all nonlinear
3 constraints, all linear; 11 nonzeros
1 nonlinear objective; 5 nonzeros.

ILOG CPLEX 9.00, licensed to "cnam-paris", options: e m b q
CPLEX 9.0.0: mipdisplay=2
mipinterval=10000
timing=1
absmipgap=0.99
timelimit=3600

Times (seconds):
Input =  0
Solve =  0
Output = 0.004
CPLEX 9.0.0: optimal solution; objective -2.00545172<- VALEUR OPTIMALE DE LA
                                                        RELAXATION CONTINUE
11 QP dual simplex iterations (8 in phase I)
a :=
1 1    761.596
1 2    760.292
1 3   -627.039
1 4    935.393
1 5   -626.539
2 1   -140.171
2 2   -140.114
2 3    248.508
2 4   -188.787
2 5    248.008
;

u [*] :=
1  -0.054438
2   1.52134
3  -2
4  -0.385729
```

```
5   2
;

x [*] :=                        <-SOLUTION OPTIMALE DE LA RELAXATION CONTINUE
1  0.957488
2  0.994686
3  0.502657
4  0.0425123
5  0.502657
;


5 variables, all nonlinear
3 constraints, all linear; 11 nonzeros
1 nonlinear objective; 5 nonzeros.

ILOG CPLEX 9.000, licensed to "cnam-paris", options: e m b q
CPLEX 9.0.0: mipdisplay=2
mipinterval=10000
timing=1
absmipgap=0.99
timelimit=3600
Clique table members: 3
MIP emphasis: balance optimality and feasibility
Root relaxation solution time =    0.00 sec.

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap

      0     0       -2.0050     2                    -2.0050        1
*     0+    0                   0       -2.0000       -2.0050        1    0.25%

Times (seconds):
Input =  0
Solve =  0
Output = 0.004
CPLEX 9.0.0: optimal integer solution within mipgap or absmipgap;
objective -2                     <-VALEUR OPTIMALE

1 MIP simplex iterations
0 branch-and-bound nodes
```

```
x [*] :=                              <- SOLUTION OPTIMALE
1  1
2  1
3  1
4  0
5  0
;
```

Pour plus de précision sur l'affichage des solutions des logiciels SB ou $CSDP$, il suffit de se référer aux manuels créés par Helmberg et Borchers.

Résumé :

Dans ce travail de thèse, nous nous sommes intéressés à la résolution exacte de problèmes d'optimisation en variables 0-1 (notés ($Q01$)), comportant une fonction objectif quadratique soumise à des contraintes linéaires. Les problèmes de bipartition de graphe, d'affectation quadratique, du sac-à-dos quadratique en sont des exemples très connus et amplement étudiés dans la littérature de l'optimisation combinatoire. On les rencontre également, de façon plus concrète, dans les domaines scientifiques, économiques et industriels tels que le choix d'investissements, la gestion de portefeuilles, la conception de réseaux, le transport, etc. Malgré des travaux pionniers datant de plus de vingt ans, les problèmes non linéaires en variables bivalentes demeurent en général mal résolus. Parvenir à résoudre les programmes ($Q01$) de grande taille représente donc un enjeu très important.

Le but de cette thèse est de proposer des méthodes originales et efficaces de résolution exacte des programmes ($Q01$). Plus précisément, notre approche principale consiste en une reformulation quadratique convexe du problème initial dans le but d'utiliser les méthodes générales de résolution des programmes quadratiques convexes en variables entières. Cette approche peut être comparée, dans son esprit, à la reformulation de problèmes non linéaires en variables 0-1 par des problèmes linéaires en variables mixtes, technique étudiée depuis longtemps et connue sous le nom de linéarisation. Une partie de notre travail est consacrée à la présentation d'une nouvelle reformulation linéaire des programmes d'optimisation quadratique en 0-1. Dans les deux cas, les problèmes reformulés peuvent être résolus par des solveurs standards de programmes en variables mixtes.

Nous présentons également de nombreuses expérimentations de notre reformulation quadratique convexe concernant trois problèmes non convexes classiques de l'optimisation combinatoire (le problème de bipartition, du sous-graphe le plus dense et un problème de minimisation d'échange d'outils). Les expérimentations montrent l'intérêt de l'approche : dans la plupart des cas, sur les instances étudiées, la méthode proposée est nettement plus efficaces que les méthodes connues à ce jour. Des résultats expérimentaux sont également présentés, concernant trois problèmes convexes (le problème de plus court chemin probabiliste, un problème d'investissements, un problème d'ordonnancement). Notre approche peut également se révèler, dans certains cas, très intéressante.

Mots-clés :

Programmes quadratiques en variables bivalentes, Programmation quadratique convexe, Reformulations, Programmation semidéfinie, Expérimentations.