HAL
open science

Optimisation globale de polynômes en variables mixte-entières

Arnaud Lazare

To cite this version:

Arnaud Lazare. Optimisation globale de polynômes en variables mixte-entières. Recherche opérationnelle [math.OC]. École doctorale de mathématiques Hadamard (EDMH, ED 574), 2019. Français. NNT : . tel-02462537v1

HAL Id: tel-02462537

https://hal.science/tel-02462537v1

Submitted on 31 Jan 2020 (v1), last revised 10 Mar 2020 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

NNT : 2019SACLY011

THÈSE DE DOCTORAT

de

L'UNIVERSITÉ PARIS-SACLAY

École doctorale de mathématiques Hadamard (EDMH, ED 574)

Établissement d'accueil : École Nationale Supérieure des Techniques Avancées

Laboratoire d'accueil : Unité de Mathématiques Appliquées, ENSTA-CNRS-INRIA

Spécialité de doctorat : Mathématiques appliquées

Arnaud LAZARE

Global optimization of polynomial programs with mixed-integer variables

Date de soutenance : 3 décembre 2019

Après avis des rapporteurs : CHRISTOPH BUCHHEIM (Technische Universität Dortmund)
FRANÇOIS CLAUTIAUX (Institut de Mathématiques de Bordeaux)

Jury de soutenance :

ALAIN BILLIONNET	(ENSIIE) Examinateur
CHRISTOPH BUCHHEIM	(Technische Universität Dortmund) Rapporteur
FRANÇOIS CLAUTIAUX	(Institut de Mathématiques de Bordeaux) Rapporteur
YVES CRAMA	(HEC Liège) Examinateur
SOUROUR ELLOUMI	(ENSTA ParisTech) Directrice de thèse
AMÉLIE LAMBERT	(CNAM) Co-encadrante de thèse
DOMINIQUE QUADRI	(Université Paris Sud) Examinatrice

Acknowledgements

Je tiens tout d'abord à remercier les deux personnes à l'origine de cette thèse. Merci Sourour et Amélie pour votre investissement dans cette thèse. Vous m'avez accompagné durant trois ans en partageant votre passion pour la recherche. Je vous remercie pour vos nombreuses idées, vos encouragements ainsi que pour vos qualités humaines. J'éprouve énormément de reconnaissance pour le travail que vous avez effectué. En particulier, merci Sourour d'avoir été à l'origine de cette aventure et de m'avoir suivi depuis l'ENSIIE avec bienveillance, en me donnant de nombreux conseils. Merci Amélie pour ton énergie, ta disponibilité, pour les nombreuses heures passés sur mon code. Merci aussi pour ta bonne humeur au quotidien.

J'aimerais aussi remercier les rapporteurs de cette thèse pour le temps qu'ils ont consacré ainsi que l'intérêt qu'ils ont porté. Thank you Professor Christoph Buchheim for your review and for the discussions we had on quadratizations. Merci au Professeur François Clautiaux pour les commentaires très encourageants qui m'ont particulièrement motivé.

Je remercie aussi les Professeurs Alain Billionnet, Yves Crama et Dominique Quadri de m'avoir fait l'honneur de faire partie de mon jury de thèse.

Je tiens à remercier toutes les personnes du CEDRIC pour ces années passées dans la bonne humeur, ainsi que les membres de l'UMA. Merci tout d'abord à Marie-Christine et Frédéric pour leur aide précieuse dans l'obtention de la bourse de thèse. Merci à Agnès, Alain, Cédric, Christophe M, Christophe P, Corinne, Daniel, Éric, Maurice, Pierre, Safia, Sami, Stéphane, Tifanie et Zacharie.

Je voudrais faire une mention spéciale pour tous ceux qui ont partagé leur bureau avec moi. Merci Dimitri et Pierre-Louis pour toutes les choses qu'ils m'ont apprises. Merci à Thomas pour son accueil. Merci aussi à Alicia, Antoine, Hadrien, Rémi et Sofya.

Je n'oublie pas tous les collègues que j'ai côtoyés durant mes séjours à Liège et à Aix-la-Chapelle. Je remercie particulièrement Élisabeth pour sa générosité, sa disponibilité ainsi que pour son implication dans les différentes collaborations. Ce fut un plaisir de travailler ensemble.

Un grand merci à ma famille, sans qui je n'aurais pas surmonté toutes ces épreuves. Merci d'avoir cru en moi et de m'avoir soutenu et encouragé dans les moments difficiles. Merci Papa pour tout ce que tu m'as appris. Bien plus qu'un professeur, tu as été le meilleur modèle pour moi. J'aimerais te dédier cette thèse, car je sais que si tu avais eu les mêmes chances que moi, tu aurais fait un bien meilleur docteur. Peu importe les diplômes, je continuerai à te regarder avec admiration. Merci Maman pour m'avoir appris à ne pas baisser les bras et pour m'avoir donné les bons conseils. J'ai toujours admiré ta façon de surmonter les difficultés et j'essaie de m'en inspirer au quotidien pour te ressembler. Je ne peux pas rêver de meilleur diplôme que celui d'avoir été votre fils. Merci Jo et Arsène pour votre affection, je vous dois aussi beaucoup de choses dans la réussite de cette thèse, et vous souhaite, à mon tour, beaucoup de réussite dans tout ce que vous entreprendrez.

J'aimerais aussi dédier un paragraphe de mes remerciements à Ragavi. Merci pour ton soutien inconditionnel, ta patience et ta compréhension. Il y énormément de raisons de te dire merci, mais j'aimerais simplement te remercier d'être là au quotidien. Pendant ces trois ans, tu as aussi traversé des moments importants, c'est l'occasion pour moi de te dire que je suis fier de toi. Je remercie aussi ta famille pour sa bienveillance et son soutien, notamment durant la rédaction.

Merci à mes cousins, cousines, oncles, tantes pour m'avoir toujours accueilli à bras ouverts et pour m'avoir aidé à sourire dans les moments plus compliqués. Merci aussi à mes grands-parents et au reste de ma famille pour leur soutien.

Merci à tous mes professeurs, à mes amis de Lyon, de la Martinière, de l'ENSIIE, du MPRO ainsi qu'à tous ceux qui ont contribué à la réalisation de ce travail.

Contents

List of Figures **8**

List of Tables **9**

0.1 Présentation du problème 12

0.2 Plan détaillé du manuscrit 13

1 Introduction **19**

1.1 Presentation of the problem 19

1.2 Applications considered in this thesis 20

1.2.1 The image restoration problem (Vision) 20

1.2.2 The *Low Auto-correlation Binary Sequence* problem (LABS) 21

1.3 Outline of the thesis 23

2 State-of-the-art **28**

2.1 Polynomial optimization 28

2.2 Unconstrained polynomial optimization with binary variables 32

2.3 Quadratic reformulation of binary polynomial programs 38

2.3.1 Quadratizations with no additional constraints 40

2.3.2 Quadratizations with additional constraints 44

3 One-step reformulation methods for pseudo-Boolean optimization **48**

3.1 Linear reformulations 49

3.1.1 Linearization of a single monomial 49

3.1.2 Linearization of a polynomial 53

3.2 Direct convexification 55

3.2.1 Approximation of the Hessian matrix 55

3.2.2 Convexification by uniform diagonal perturbation 56

3.2.3 Convexification by non-uniform diagonal perturbation 58

3.3 Conclusion 59

4 PQCR: Polynomial optimization in binary variables through Quadratic Convex Reformulation 61
4.1 A quadratic convex reformulation framework . . . 62
4.1.1 Phase 1: Quadratization of polynomial program (P) into (QP) . . . 63
4.1.2 Phase 2: Optimal quadratic convex reformulation of (QP) . . . 65
4.2 Discussion on the impact of the chosen 2×2 quadratization . . . 76
4.2.1 2×2 Full quadratization . . . 82
4.2.2 2×2 Partial quadratization . . . 84
4.2.3 Replacing PQCR's quadratization scheme with other pairwise covers . . . 86
4.3 Link with the Lasserre's hierarchy . . . 88
4.3.1 A hierarchy of exact quadratic convex reformulations . . . 88
4.4 Conclusion . . . 91

5 Convexification with other quadratization schemes 93
5.1 "Rosenberg-like" quadratizations . . . 94
5.1.1 Applying diagonal convexification after Rosenberg's procedure . . . 94
5.1.2 Using Rosenberg's procedure in a new compact convexification framework . 97
5.2 Applying QCR after quadratizations using pairwise covers . . . 101
5.3 Applying QCR after termwise quadratization . . . 103
5.4 Conclusion . . . 105

6 Semi-definite relaxations of box-constrained polynomial programs 106
6.1 A quadratic reformulation of $(\bar{P})$ . . . 107
6.2 A compact semidefinite programming relaxation . . . 107
6.3 An improved semidefinite programming relaxation . . . 108
6.4 Conclusion . . . 110

7 Computational experiments and implementations 112
7.1 Hardware and used software . . . 113
7.2 Details on the instances . . . 113
7.3 Performance of PQCR (presented in Chapter 4) . . . 114
7.3.1 The compared methods (PQCR, Q+QCR, Q+MIQCR, Q+cplex and Baron) . . . 114
7.3.2 Results on Vision instances . . . 115
7.3.3 Results on LABS instances . . . 116
7.4 Convex reformulation with other quadratization schemes (presented in Chapter 5) 123
7.4.1 Results related to Rosenberg's reformulation . . . 123
7.4.2 Experimental comparison between termwise and pairwise convexification . 131
7.5 Tightness of semi-definite relaxations for box-constrained polynomial programs (presented in Chapter 6) . . . 136

7.5.1 Instance generation 136
7.5.2 Experimental results 136

8 Conclusion **139**
8.1 Main contributions 139
8.2 Perspectives and research directions 142

A The `PQCR` solver **145**
A.1 Implemented algorithms 145
A.2 Quadratizations 146
A.3 Instances 146

Bibliography **148**

List of Figures

1.1 An example of a 5×5 Vision instance. The problem consists in retrieving the base image (right) from a perturbed one (left) 21

7.1 *Legend of Tables 7.1-7.3.* 118
7.2 Comparison between the final gap of `PQCR` and the final gap computed with the best known solution and the best bound from `minlplib` for the unsolved autocorrelation instances 121
7.3 *Legend of Tables 7.4-7.15* 125
7.4 *Legend of Table 7.16* 137

8.1 Theoretical comparison of the different methods. A vertical link means that the method placed at the top is proven to have a better bound than the one at the bottom. 142
8.2 Experimental comparison of the different methods. A vertical link means that the method placed at the top obtains shorter CPU times than the one at the bottom on the instances considered in this thesis. 142

List of Tables

4.1 Comparison of the full quadratization, the partial quadratization and the quadratization of Algorithm 4.2.1 . . . 85

4.2 Comparison of bounds (LB), number of variables (N) and total time (T_t) of `PQCR` after different quadratizations . . . 87

7.1 Comparison of `PQCR` to `Q+QCR`, `Q+Cplex` and `Baron` for the *vision* instances - time limit one hour. Legend on Figure 7.1. . . . 119

7.2 Results of `PQCR` and `Baron` for the 45 instances of the LABS problem. Time limit 5 hours (3h for SDP, 2h for `Cplex`). Legend on Figure 7.1. . . . 120

7.3 Comparison of the best known solution and best lower bound values of `PQCR` and of the `minlplib` for the unsolved LABS instances. $**$: solved for the first time, #: best known solution improved, and $*$: best known lower bound improved. Legend on Figure 7.1. . . . 122

7.4 Performance of `ROS+QCR` on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. . . . 125

7.5 Performance of `PQCR ROS` on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. . . . 126

7.6 Performance of `Q+QCR` on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. . . . 126

7.7 Performance of `PQCR 2` on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. . . . 126

7.8 Performance of `ROS+QCR` on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. . . . 127

7.9 Performance of `PQCR ROS` on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. . . . 128

7.10 Performance of `Q+QCR` on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. . . . 129

7.11 Performance of `PQCR 2` on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. . . . 130

7.12 Performance of PC+QCR using Pairwise Cover 0 on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. 133
7.13 Performance of T+QCR on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. 134
7.14 Performance of PC+QCR using Pairwise Cover 0 on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. 135
7.15 Performance of T+QCR on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3. 135
7.16 Comparison of the lower bounds of $SDP^0_{\mathcal{E}}$ and $SDP^1_{\mathcal{E}}$ with the lower bounds obtained by the branch-and-cut of SCIP in one hour of CPU time. Legend on Figure 7.4 138

Notations

- $X \succeq 0$, matrix X is positive semidefinite
- $diag(\alpha)$, diagonal matrix composed of the values of vector α
- $\lambda_{min}(M)$, smallest eigenvalue of M
- $\lambda_{max}(M)$, largest eigenvalue of M
- $(\overline{P})$, continuous relaxation of problem (P)
- $v(P)$, optimal value of problem (P)
- $\sqcup$, disjoint union operator.
- Vision, Image restoration problem
- LABS, Low autocorrelation binary sequences problem

Résumé long

0.1 Présentation du problème

L'optimisation polynomiale est l'une des classes de problèmes d'optimisation les plus difficiles à résoudre à l'heure actuelle. Dans sa forme la plus simple, le problème consiste à trouver la valeur optimale d'une fonction polynomiale dans un ensemble défini par des contraintes polynomiales. Nous appelons (P_G) le problème d'optimisation polynomiale suivant.

$$(P_G) \begin{cases} \min h(x) \\ \text{s.t.} \\ \quad h_i(x) \geq 0 \qquad i = 1, ..., m \\ \quad x \in \mathbb{R}^n \end{cases}$$

où les fonctions h et h_i sont des fonctions polynomiales de x. Les progrès réalisés dans les cas linéaires et quadratiques au cours des cinquante dernières années ont amené les chercheurs à se concentrer sur la généralisation naturelle que sont les problèmes d'optimisation polynomiale. Cependant, cette classe de problèmes est beaucoup plus difficile à résoudre. En effet, Murty et al ont prouvé dans [65] que le fait de vérifier qu'un polynôme de degré 4 est positif est NP-difficile en général. L'optimisation polynomiale est donc extrêmement difficile mais son étude est motivée par un large éventail d'applications, puisque (P_G) est un modèle général qui permet de formuler plusieurs problèmes importants en optimisation et de nombreux problèmes non linéaires peuvent être modélisés comme des problèmes d'optimisation polynomiale.

Considérons par exemple le problème de restauration d'images. Il consiste à reconstruire une image nette à partir d'une image floue. Ce problème peut être modélisé comme un problème d'optimisation polynomiale. Les algorithmes permettant de résoudre ce problème sont utilisés dans de nombreux logiciels de retouche photo par exemple. De nombreux ingénieurs travaillent sur ce problème mais, à notre connaissance, personne ne calcule la solution exacte du problème d'optimisation sous-jacent. En effet, ce problème est très difficile et seuls des algorithmes heuris-

tiques sont implémentés. Le but est de calculer une solution rapide et approximative plutôt que de calculer un optimum global avec un coût en temps prohibitif. En raison de sa difficulté, l'optimisation polynomiale a été principalement étudiée dans certains cas spécifiques, comme l'optimisation polynomiale en variables binaires ou dans le cas des polynômes univariés, par exemple.

L'optimisation polynomiale a repris de l'intérêt au début du 21ème siècle avec les travaux de Lasserre introduisant la puissante hiérarchie Moment/Sum-Of-Squares [50]. Bien que la méthode originale contienne un vaste apport théorique, elle est aussi bien connue pour son intérêt pratique. Le solveur `GloptiPoly` développé par Lasserre et al. [41] était sans doute le plus efficace pour résoudre les programmes polynomiaux lorsqu'il a été publié. La tractabilité de la hiérarchie de Lasserre s'explique aussi par plusieurs améliorations apportées à la programmation semi-définie, sur laquelle la méthode Moment/SOS s'appuie fortement. Ce premier travail a ouvert la voie à de nombreuses variantes et extensions de la méthode de Lasserre. Depuis, de nombreuses méthodes alternatives ont vu le jour la littérature. Dans le chapitre 2, nous passons en revue plusieurs avancées significatives dans le domaine de l'optimisation polynomiale, en commençant par des problèmes généraux et en terminant par des problèmes plus spécifiques.

0.2 Plan détaillé du manuscrit

Tout au long de la première partie de cette thèse, nous étudions des problèmes d'optimisation en variables binaires et non contraints. La forme générale d'un tel programme est la suivante

$$(P) \begin{cases} \min f(x) \\ \text{s.t.} \\ \quad x \in \{0,1\}^n \end{cases}$$

Ce problème d'optimisation est aussi appelé *optimisation pseudo-booléenne* dans la littérature. En effet, il est prouvé dans [38] que chaque fonction pseudo-Booléenne f (c'est-à-dire une application $f : \{0,1\}^n \to \mathbb{R}$) peut s'écrire comme un unique polynôme multilinéaire. Ce problème est difficile à résoudre même lorsque f est quadratique. Plusieurs solveurs conçus pour des problèmes plus généraux n'ont pas réussi à résoudre les instances difficiles de la littérature. Un des buts principaux de cette thèse est de développer une méthode générale pour résoudre le problème (P) et d'améliorer la performance sur les instances difficiles. Nous présentons plusieurs méthodes de reformulation pour résoudre le problème de façon optimale. En partant du problème (P), nous voulons construire un programme équivalent qui possède une sous-structure intéressante (linéaire, quadratique et convexe, etc.) et qui est généralement plus facile à résoudre, et où la résolution de ce nouveau problème revient à résoudre (P). Tout au long de notre étude, nous sommes amenés à comparer théoriquement nos approches à certaines des méthodes bien connues dans la littérature.

Nous fournissons également des résultats expérimentaux pour les comparaisons pratiques.

En plus de l'introduction dans le chapitre 1 et de la conclusion dans le chapitre 8, cette thèse contient 6 autres chapitres. Nous commençons par une revue de la littérature dans le chapitre 2. Ensuite, le chapitre 3 traite de deux méthodes de convexification, à savoir la linéarisation et la convexification directe. Le chapitre 4 présente un nouvel algorithme utilisant la reformulation quadratique convexe. Une discussion plus approfondie sur les quadratisations peut être trouvée au chapitre 5. Le chapitre 6 discute des extensions de notre travail au cas des variables continues. Enfin, tous les résultats expérimentaux sont présentés et commentés dans le chapitre 7. Nous présentons un résumé détaillé de chaque chapitre dans la suite.

Chapitre 1 : Introduction Nous présentons le contexte actuel de l'optimisation polynomiale et nous montrons deux applications " réelles " qui nous intéressent dans cette thèse.

Chapitre 2 : Etat de l'art Dans ce chapitre, nous étudions la littérature concernant l'optimisation polynomiale. Nous commençons par les méthodes classiques qui résolvent les problèmes généraux d'optimisation polynomiale. Ensuite, nous nous concentrons sur des restrictions spécifiques de (P), telles que l'optimisation quadratique et l'optimisation polynomiale en variables binaires sans contrainte. Nous terminons cette revue en présentant différentes reformulations quadratiques pour des programmes polynomiaux en variables binaires. Nous comparons ensuite nos méthodes avec ces familles de reformulations.

Chapitre 3 : Méthodes de reformulation en une étape pour l'optimisation pseudo-booléenne Dans le chapitre 3, nous discutons des méthodes de reformulation en une seule étape par opposition aux autres méthodes qui nécessitent plusieurs reformulations consécutives.

Le chapitre 3.1 présente la famille des reformulations linéaires aussi connues sous le nom de *linéarisations*. Elle consiste à reformuler f en une fonction linéaire après avoir ajouté des variables auxiliaires et des contraintes à (P). Plus précisément, pour chaque monôme, la linéarisation standard introduit une variable supplémentaire et un ensemble de contraintes qui imposent l'égalité entre le monôme et la nouvelle variable. Cependant, nous montrons que ce n'est pas la seule reformulation linéaire valide et que l'on peut appliquer de nombreuses linéarisations pour une instance donnée. Chaque linéarisation conduit à un nombre différent de variables, à des ensembles de contraintes différents et à des valeurs de bornes par relaxation continue différentes. Cette section vise à mettre en évidence les cas particuliers où les bornes par relaxation continue peuvent être comparées. Pour cela, nous définissons formellement un cadre général de linéarisation que nous appelons $q-$*linéarisation* qui représente une classe de reformulations linéaires. Nous prouvons que, pour le cas d'un seul monôme, toute $q-$linéarisation donne la même borne. Nous montrons ensuite que ce résultat peut être étendu à un polynôme entier lorsque chaque variable auxiliaire n'est utilisée qu'une seule fois.

La section 3.2, au contraire, introduit une reformulation qui ne modifie ni le degré ni la dimension du problème. Ici, nous voulons une reformulation de (P) qui a une relaxation continue convexe. Pour cela, nous ajoutons des fonctions nulles sur $\{0,1\}^n$ à f. Plus précisément, nous utilisons le fait que la fonction $x_i \mapsto x_i^2 - x_i$ est nulle lorsque x_i est une variable binaire. Puis, nous considérons la fonction paramétrée par le vecteur λ, $f_\lambda : \ x \mapsto f(x) + \sum_i \lambda_i(x_i^2 - x_i)$ qui est égal à f sur l'ensemble des variables binaires. La matrice hessienne H_{f_λ} de f_λ est telle que $H_{f_\lambda} = H + 2diag(\lambda)$. Ainsi, en choisissant soigneusement les valeurs de λ, nous pouvons rendre f_λ convexe. Evidemment, nous pouvons prendre des valeurs positives arbitrairement grandes pour λ mais la valeur de la relaxation continue serait très faible. Le problème est de trouver λ qui rend f_λ convexe et qui donne la meilleure borne par relaxation continue. Cependant, lorsque le degré du polynôme considéré est supérieur à 2, la matrice hessienne n'est pas constante et est donc une fonction de x. Dans ce contexte, il est plus difficile de calculer un tel λ. Notre première étape est d'approximer cette matrice non constante par une matrice d'*intervalle*. Puis nous appliquons les théorèmes de Gerschgorin [32]. Le premier donne une borne inférieure sur la plus petite valeur propre d'une telle matrice, et conduit donc à une perturbation uniforme de la diagonale de la matrice hessienne. La seconde fournit une perturbation non uniforme.

Chapitre 4: `PQCR`: Optimisation polynomiale dans les variables binaires par reformulation quadratique convexe Dans ce chapitre, nous nous intéressons à la reformulation quadratique convexe de (P) c'est-à-dire une reformulation de (P) en (QP), un problème quadratique une certaine propriété de convexité.

Dans la section 4.1, nous proposons une approche de résolution exacte pour le problème (P). Nous appelons `PQCR` (`Polynomial Quadratic Convex Reformulation`) cette méthode en trois phases. La première phase consiste à reformuler (P) en un programme quadratique (QP). Pour cela, nous réduisons itérativement le degré de (P) jusqu'à deux, en utilisant la linéarisation standard du produit de deux variables par une nouvelle variable, que nous appelons une *quadratisation* 2×2, ou de façon équivalente une *pairwise cover* telle qu'elle est introduite dans [5]. On obtient alors un programme quadratique en variables binaires et avec des contraintes linéaires. Dans la deuxième phase, nous reformulons la fonction objectif de (QP) en une fonction quadratique équivalente et paramétrée en utilisant l'identité $x_i^2 = x_i$ et d'autres égalités quadratiques valides que nous introduisons à partir de la reformulation de la phase 1. Ensuite, nous nous concentrons sur la recherche des meilleurs paramètres pour obtenir un programme quadratique convexe dont la valeur optimale de la relaxation continue est maximisée. Pour cela, nous construisons une nouvelle relaxation semi-définie (SDP) de (QP). Ensuite, nous prouvons que les inégalités de linéarisation standard, utilisées pour l'étape de quadratisation, sont redondantes en présence des nouvelles égalités quadratiques. Ensuite, nous déduisons nos paramètres optimaux à partir de la solution optimale du dual de (SDP). La troisième phase consiste à résoudre (QP^*), le problème reformulé de façon optimale, avec un solveur standard. En particulier, à chaque nœud du branch-and-bound, le solveur calcule la valeur optimale d'un programme quadratique convexe

continu.

Dans la section 4.2, nous nous concentrons sur la phase de quadratisation. En effet, la valeur de la relaxation continue (QP^*) dépend du schéma de quadratisation et varie donc pour différentes reformulations quadratiques 2×2. Le but de cette section est d'essayer de classer les quadratisations en fonction de leurs bornes après convexification. Nous définissons d'abord la propriété de *stabilité* pour une famille de quadratisations, qui caractérise un ensemble de quadratisations ayant la même valeur par relaxation continue après convexification. Nous prouvons ensuite que, si deux quadratisations différentes introduisent le même nombre de variables et si chaque variable ajoutée représente le même produit, alors les deux quadratisations conduisent à la même borne après convexification. Ensuite, nous montrons que l'ajout de variables à une quadratisation peut aider à améliorer la borne. Nous présentons ainsi deux cas particuliers de quadratisation, à savoir les quadratisations *complète* et *partielle*. La quadratisation complète présente un intérêt théorique car elle permet de lier `PQCR` et la hiérarchie de Lasserre. En revanche, la quadratisation partielle a un intérêt purement pratique car elle présente un bon compromis entre la qualité des bornes et la dimension des problèmes. Enfin, nous comparons expérimentalement les bornes obtenues par différents schémas de quadratisation 2×2.

Nous terminons ce chapitre en présentant les liens entre `PQCR` et la hiérarchie de Lasserre pour les programmes polynomiaux en variables binaires non contraints dans la section 4.3. Nous montrons que, en appliquant `PQCR` avec la quadratisation complète, on obtient la même relaxation semi-définie que le premier ordre de la hiérarchie de Lasserre. Ensuite, nous affirmons que pour chaque ordre k de la hiérarchie de Lasserre, il existe une quadratisation de la méthode `PQCR`, dont la relaxation semi-définie correspond à cette relaxation semi-définie d'ordre k. Ceci conduit naturellement à une hiérarchie de reformulations quadratiques convexes basées sur la hiérarchie de Lasserre. Cette hiérarchie hérite de toutes les propriétés intéressantes de la hiérarchie Moment/SOS définie par Lasserre, y compris la propriété de convergence finie. Enfin, nous montrons que `PQCR`, avec différentes quadratisations 2×2, se révèle être assez performante pour l'optimisation polynomiale en variables binaires et sans contrainte car il ne nécessite pas de reformuler tous les produits de deux variables d'un degré donné d. En effet, la reformulation quadratique effectuée dans `PQCR` peut reformuler les produits apparaissant dans le problème initial seulement. Cela conduit à un meilleur compromis entre la qualité de la borne et le temps de résolution.

Chapitre 5 : Convexification avec d'autres schémas de quadratisation Dans ce chapitre, nous étudions la compatibilité entre `PQCR` et d'autres quadratisations décrites dans la littérature. En effet, comme le Chapitre 4 est limité aux quadratisations utilisant les contraintes de Fortet [28], nous construisons ici de nouvelles reformulations convexes plus adaptées à chaque quadratisation.

Nous commençons par la procédure de Rosenberg dans la section 5.1. Etant donné un polynôme f en variables binaires, cette quadratisation introduit de nouvelles variables et ajoute un terme de pénalité à f. Plus précisément, on remarque que pour une nouvelle variable y_{ij}

remplaçant le produit $x_i x_j$, l'égalité $R(x,y) = x_i x_j - 2x_i y_{ij} - 2x_j y_{ij} + 3y_{ij} = 0$ est toujours satisfaite. L'idée est de forcer cette égalité par une pénalité P dans une nouvelle fonction $f_p(x,y) = f(x,y) + P \times R(x,y)$, où $f(x,y)$ est une fonction quadratique dans laquelle chaque variable y_{ij} remplace le produit correspondant $x_i x_j$ et P est un grand nombre positif. Pour toute solution optimale, on a $f_p(x,y) = f(x,y)$. De plus, une des caractéristiques de la procédure de Rosenberg est de reformuler quadratiquement f sans ajouter de contraintes. Cependant, après la phase de convexification, cette propriété n'est plus vérifiée car les contraintes de linéarisation classiques sont nécessaires pour imposer l'équivalence à (P). Nous introduisons donc une première reformulation convexe pour de telles quadratisations en utilisant l'égalité $x_i^2 = x_i$, ce qui revient à appliquer la méthode `QCR` [13]. Nous présentons ensuite une nouvelle convexification qui est adaptée de la procédure de Rosenberg. Elle consiste à utiliser $R(x,y) = 0$ comme égalité valide afin de calculer une reformulation convexe. Nous comparons ensuite l'impact de cette famille d'égalités valides avec celle utilisée dans `PQCR`.

Dans la section 5.2, nous dérivons une nouvelle reformulation convexe pour la quadratisation introduite dans [5]. De la même manière que la procédure de Rosenberg, les auteurs introduisent une fonction quadratique égale à f sur tous les points optimaux en introduisant des variables et des monômes supplémentaires. La seule égalité valide sur le domaine est $x_i^2 = x_i$ et on construit une convexification ne perturbant que la diagonale de la matrice hessienne.

Dans la section 5.3, nous appliquons la reformulation convexe aux quadratisations *termwise*. Cette famille de quadratisations introduit une reformulation quadratique pour chaque monôme séparément. Nous appliquons la convexification diagonale standard en utilisant $x_i^2 = x_i$.

Chapter 6 : Relaxations semi-définies de programmes polynomiaux avec contraintes de boîtes Dans ce chapitre, nous calculons des relaxations convexes serrées pour l'optimisation polynomiale continue et en particulier pour les problèmes avec des contraintes de boîtes (P_c) où les variables sont dans des intervalles réels. On observe qu'il n'y a pas de moyen simple de dériver une reformulation quadratique convexe comme cela a été fait dans le cas discret. En effet, l'égalité $y_{ij} = x_i x_j$ ne peut plus être linéarisée de façon équivalente. De plus, certaines des égalités utilisées dans `PQCR` ne sont plus valides, mais elles peuvent être remplacées par des inégalités. Dans ce contexte, nous proposons une reformulation quadratique de (P_c). Nous présentons ensuite deux relaxations semi-définies pour (P_c). Nous introduisons une première relaxation compacte que nous améliorons encore pour atteindre des meilleures bornes.

Chapitre 7 : Résultats expérimentaux et implémentations Ce chapitre fournit des résultats expérimentaux pour les méthodes qui sont introduites dans cette thèse. Nous présentons des résultats où nous comparons notre méthode principale `PQCR` introduite dans le chapitre 4 avec d'autres méthodes de convexification, et avec le solveur `Baron` [73]. Nous évaluons notre méthode sur des instances du problème de restauration d'image [24] et du problème de suite binaire à faible auto-corrélation [9] de `minlplib` [63]. Pour ce dernier problème, 33 instances parmi les 45

n'ont pas été résolus dans `minlplib`. Nous en résolvons 6 à l'optimum, et pour les 27 autres, nous améliorons les bornes primales et/ou duales. Nous comparons et analysons les différentes reformulations convexes introduites dans le chapitre 5 en termes de bornes et de nombre de variables et nous justifions le choix d'une quadratisation 2×2. En particulier, nous comparons ces nouvelles convexifications avec une version restreinte de `PQCR` où la seule égalité valide utilisée est $x_i^2 = x_i$. Enfin, nous illustrons la qualité de nos relaxations semi-définies introduites dans le chapitre 6 sur des instances polynomiales générées aléatoirement.

Chapitre 8 : Conclusion Nous terminons cette étude par un bref aperçu des principaux résultats présentés dans cette thèse. Nous présentons également des perspectives de travaux futurs dans la continuité de ceux déjà réalisés. En particulier, nous considérons les programmes polynomiaux en variables binaires avec contraintes et nous essayons de relier cette famille de problèmes à notre méthode général utilisée dans `PQCR`. Ensuite, nous proposons une approche pour prendre en compte les variables entières car la partie discrète de notre étude ne porte que sur les variables binaires. Enfin, nous abordons le problème des programmes polynomiaux généraux en variables mixtes-entières.

Nous terminons la partie théorique en présentant une extension de la reformulation quadratique convexe à certains programmes polynomiaux en nombres entiers mixtes. Comme pour la linéarisation d'un produit binaire, il est possible de reformuler les produits d'une variable binaire par une variable continue à l'aide de contraintes linéaires. Nous exploitons cette reformulation pour un cas spécifique de programmes polynomiaux en variables mixtes-entières. En effet, lorsque l'on considère des monômes avec au plus deux variables continues, on peut appliquer une linéarisation récursive des produits de deux variables jusqu'à obtenir une fonction quadratique (QP_{MI}). Ensuite, nous ajoutons des fonctions nulles à cette fonction quadratique pour obtenir une fonction quadratique paramétrée (QP_λ^{MI}). Nous nous concentrons ensuite sur la recherche des meilleurs paramètres, c'est-à-dire des paramètres qui rendent la nouvelle fonction objectif convexe et tels que la valeur optimale de la relaxation continue de (QP_λ^{MI}) soit maximisée.

Chapter 1

Introduction

Contents

1.1 Presentation of the problem **19**

1.2 Applications considered in this thesis **20**

1.2.1 The image restoration problem (Vision) 20

1.2.2 The *Low Auto-correlation Binary Sequence* problem (LABS) 21

1.3 Outline of the thesis **23**

1.1 Presentation of the problem

Polynomial optimization is one of the current challenging classes of optimization problems. In its simplest form, it consists in finding the optimal value of a polynomial function within a set defined by polynomial constraints. We call (P_G) the following polynomial optimization problem.

$$(P_G) \begin{cases} \min h(x) \\ \text{s.t.} \\ \quad h_i(x) \geq 0 \qquad i = 1, ..., m \\ \quad x \in \mathbb{R}^n \end{cases}$$

where functions h and h_i are polynomial functions of x. The advances made in both linear and quadratic cases in the last fifty years have led researchers to focus on the natural generalization of polynomial optimization problems. However, this class of problem is much harder to solve. Indeed, Murty et al proved in [65] that even testing whether a degree-4 polynomial is nonnegative is NP-hard in general. Polynomial optimization is thus extremely hard but its study is motivated

by a wide range of applications since (P_G) is a general model that allows to formulate several important problems in optimization and many non-linear problems can be modeled as a polynomial optimization problem.

Let us consider for example the image restoration problem that will be addressed later in this thesis. It consists in retrieving a sharp image from a blurred one. This problem can be modeled as a polynomial optimization problem. Algorithms solving this problem are used in many photo editing softwares for example. Many engineers are working on this problem but none of them venture to compute the exact global optimum. Indeed, this problem is very hard and only heuristic algorithms are implemented. The goal is to compute a fast approximate solution rather than computing the global optimum with a prohibitive time cost. Due to its difficulty, polynomial optimization has been mostly studied in some specific cases, such as binary polynomial optimization or univariate polynomials, for example.

Polynomial optimization has regained interest in the beginning of the 21st century with the work of Lasserre introducing the powerful Moment/Sum-Of-Squares hierarchy [50]. Although the original method contains a vast theoretical contribution, it is also well-known for its practical interest. The solver `GloptiPoly` developped by Lasserre et al. [41] was arguably the most efficient for general polynomial programs when it was published. The tractability of the Lasserre's hierarchy can also be explained by several improvements made on semi-definite programming, on which the Moment/SOS method relies heavily. This first work has opened the way for many variants and extensions of Lasserre's method. Since then, many alternative methods are available in the literature. In Chapter 2, we review several significant advances in polynomial optimization, starting with general problems and ending with more specific ones.

1.2 Applications considered in this thesis

In addition to random instances coming from the literature, in this thesis we evaluate our algorithms on real applications. In the following, we describe two applications that can be modeled as unconstrained polynomial optimization problems with binary variables.

1.2.1 The image restoration problem (Vision)

The *vision* instances are inspired from the image restoration problem, which arises in computer vision. The goal is to reconstruct an original sharp base image from a blurred image. In this thesis, we study the restriction where there is only two colors, each pixel can be black or white. An image is a rectangle containing $n = l \times h$ pixels. A complete description of these instances can be found in [24]. Figure 1.1 illustrates this problem on a small example.

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Figure 1.1: *An example of a* 5×5 *Vision instance. The problem consists in retrieving the base image (right) from a perturbed one (left)*

This problem can be modeled as the minimization of a degree 4 polynomial of binary variables where each variable represents a pixel. Indeed, the rectangle is modeled as a $l \times h$ matrix of binary variables. For the objective function, we consider 2×2 squares of adjacent pixels in the original image and we focus on the values taken by each of the 4 pixels in the base image. The monomials thus represent a specific configuration of a given square. For a given 2×2 square c, each monomial is of the form $a_c f_c(x)$, where f_c is a multilinear product of the variables in c and their complementary. For example, The monomial $x_1 x_2 x_3 \overline{x}_4$ refers to the following 2×2 square:

x_1	x_2
x_3	x_4

and represents the configuration where pixels 1, 2 and 3 are white and pixel 4 is black. The coefficient a_c of a monomial is indicative of how likely configuration c is to appear on the sharp base image. More precisely, an improbable configuration will be penalized with a large positive coefficient. Finally, the polynomial optimization problem corresponding to the image restoration problem is

$$\min_{\{0,1\}^n} \sum_{c \in C} a_c f_c(x_c)$$

where C is the set of 2×2 pixel squares of the initial image.

1.2.2 The *Low Auto-correlation Binary Sequence* problem (LABS)

We consider the problem of binary sequences with low off-peak auto-correlations, that is minimizing the auto-correlations outside their peak values. This problem has numerous practical applications in communication engineering, or theoretical physics [9]. Among them, we can cite the problem of synchronization in communication systems, where it is easier to detect a sequence having a single high-peak and much lower off-peak levels. LABS is also used as modulation pulses in radar and sonar as one can get accurate target detections with a sharply peaked auto-correlation function.

An auto-correlation is the correlation between successive values of a variable over time. More formally, let S be a sequence $S = (s_1, \ldots, s_n)$ with $s \in \{-1, 1\}^n$. This sequence can be thought as a representation of the value of variable S over a discrete time window. For a given $k = 0, \ldots, n-1$, we are interested in the correlation between the values of S at times i and $i + k$. We thus define the auto-correlations $C_k(S)$ of S:

$$C_k(S) = \sum_{i=1}^{n-k} s_i s_{i+k}$$

The problem is to find a sequence S of length n that minimizes the energy level $E(S)$. This problem amounts to the minimization of a degree 4 polynomial whose variables can take values -1 and 1:

$$E(S) = \sum_{k=1}^{n-1} C_k^2(S)$$

In this thesis, we consider truncated instances, i.e. sequences of length n where we compute low off-peak auto-correlation up to a certain distance (or time window) $2 \leq n_0 \leq n$, i.e. we consider the following function to minimize:

$$E_{n_0}(S) = \sum_{k=1}^{n_0-1} C_k^2(S)$$

Example 1. *We want to compute the optimal energy level for $n = 4$ et $n_0 = 4$. Let $S = (s_1, s_2, s_3, s_4)$ be a binary sequence. The auto-correlations are given by*

$$C_1(S) = s_1 s_2 + s_2 s_3 + s_3 s_4, \;\; C_2(S) = s_1 s_3 + s_2 s_4, \;\; C_3(S) = s_1 s_4$$

The energy level $E_4(S)$ is defined by

$$E_4(S) = C_1(S)^2 + C_2(S)^2 + C_3(S)^2 =$$

$$4s_1 s_2 s_3 s_4 + 2s_1 s_2^2 s_3 + 2s_2 s_3^2 s_4 + s_1^2 s_2^2 + s_2^2 s_3^2 + s_3^2 s_4^2 + s_1^2 s_3^2 + s_2^2 s_4^2 + s_1^2 s_4^2$$

The corresponding LABS problem is

$$\min_{S \in \{-1,1\}^4} E_4(S)$$

The sequence $S = (1, 1, -1, 1)$ is an optimal solution with an energy level of 2.

LABS instances up to $n = n_0 = 60$ can be found on the `MINLPLIB` website [63], and most of them remain unsolved. Note that on the proposed instances, variables are converted from $s \in \{-1, 1\}^n$ to $x \in \{0, 1\}^n$ using the standard transformation $x = \frac{s+1}{2}$.

This problem admits several symmetries. One in particular is interesting from a computational point of view: the correlations C_k are identical for a sequence S and its complement.

1.3 Outline of the thesis

Throughout the first part of this thesis we study unconstrained binary polynomial optimization problems. The general form of such a program is the following

$$(P)\begin{cases} \min f(x) \\ \text{s.t.} \\ \quad x \in \{0,1\}^n \end{cases}$$

where f is a polynomial of n variables. This optimization problem is also called *pseudo-Boolean optimization* in the literature. Indeed, it is proven in [38] that every pseudo-Boolean function f (i.e. a mapping $f : \{0,1\}^n \to \mathbb{R}$) can be represented uniquely as a multilinear polynomial. This problem is NP-hard even when f is quadratic. Several solvers designed for more general problems failed to solve challenging instances from the literature. One of the main purpose of this thesis is to develop a general method to solve problem (P) and to improve tractability on difficult instances. We present several reformulation methods to solve the problem to optimality. Starting from problem (P) we want to build an equivalent program which has some nice sub-structures (linear, quadratic and convex, etc.) and is usually easier to solve, and where solving this new problem amounts to solving (P). Throughout our study, we are led to theoretically compare our approaches to some of the well-known methods in the literature. We also provide experimental results for practical comparisons.

In addition to the introduction in Chapter 1 and the conclusion in Chapter 8, this thesis contains 6 other chapters. We begin with a literature review in Chapter 2. Then, Chapter 3 is concerned with two convexification methods, namely the linearization and the direct convexification. Chapter 4 introduces a new algorithm using quadratic convex reformulation. Further discussion on quadratizations takes place in Chapter 5. Chapter 6 discuss of extensions of our work to the case of continuous variables. Finally, all the computational results are presented and commented in Chapter 7. We present a detailed summary of each chapter in the following.

Chapter 1: Introduction We present the current context of polynomial optimization and we display two "real-life" applications by which we are concerned with in this thesis.

Chapter 2: State-of-the-art In this chapter, we study the literature concerning polynomial optimization. We start with mainstream methods that solve general polynomial optimization problems. Then we focus on specific restrictions of (P), such as quadratic optimization and unconstrained binary polynomial optimization. We end this review by presenting different quadratic reformulations for binary polynomial programs. We further compare our methods with these families of reformulations.

Chapter 3: One-step reformulation methods for pseudo-Boolean optimization In Chapter 3, we discuss single-step reformulation methods in opposition to other methods requiring several consecutive reformulations.

Section 3.1 presents the family of linear reformulations also known as *linearizations*. It consists in reformulating f into a linear function after adding auxiliary variables and constraints to (P). More precisely, for each monomial, the standard linearization introduces an additional variable and a set of constraints that enforces the equality between the monomial and the new variable. However, we show that this is not the only valid linear reformulation and one can apply plenty of linearizations on a given instance. Each linearization leads to a different number of variables, different sets of constraints and different continuous relaxation bound values. This section aims at pointing out the special cases where the continuous relaxation bounds can be compared. For this, we formally define a general linearization framework that we call $q-$*linearization* which represents a large class of linear reformulations. We prove that, for the case of a single monomial, any $q-$linearization gives the same bound. We then show that this result can be extended to a whole polynomial when each auxiliary variable is only used once.

Section 3.2, on the opposite, introduces a reformulation that do not modify neither the degree nor the dimension of the problem. Here, we want a reformulation of (P) that has a convex continuous relaxation. For this we add null functions on $\{0,1\}^n$ to f. More precisely, we use the fact that the function $x_i \mapsto x_i^2 - x_i$ is null whenever x_i is a binary variable. Now, let us consider the function parameterized by the vector λ, $f_\lambda :\ x \mapsto f(x) + \sum_i \lambda_i(x_i^2 - x_i)$ which is equal to f on every binary point. The Hessian matrix of f_λ is such that $H_{f_\lambda} = H + 2diag(\lambda)$. Thus, by carefully choosing the values of λ, we can make f_λ convex. Obviously one can take large positive values for λ but the continuous relaxation bound value would be very weak. The issue lies in finding λ that makes f_λ convex and gives the best continuous relaxation bound value. However, when the degree of the considered polynomial is greater than 2, the Hessian matrix is not constant and thus is a function of x. In this context it is more difficult to compute a tight λ. Our first step is to approximate this non-constant matrix with an *interval matrix*. Then we apply the Gerschgorin Theorems [32]. The first one gives a lower bound on the smallest eigenvalue of such a matrix, and thus leads to a uniform perturbation of the diagonal of the Hessian matrix. The second one provides a non-uniform perturbation.

Chapter 4: `PQCR`: Polynomial optimization in binary variables through Quadratic Convex Reformulation In this Chapter, we are interested in the quadratic convex reformulation of (P) that is, a reformulation of (P) into (QP), a quadratic problem with some convexity properties.

In section 4.1, we propose an exact solution approach for problem (P). We call `PQCR` (`Polynomial Quadratic Convex Reformulation`) this three-phase method. The first phase consists in reformulating (P) into a quadratic program (QP). To that end, we iteratively reduce the degree of (P) to two, by use of the standard substitution of the product of two variables by a new one,

which we call a 2×2 *quadratization*, or equivalently a *pairwise cover* as it is introduced in [5]. We then obtain a linearly constrained binary quadratic program. In the second phase, we rewrite the objective function of (QP) into an equivalent and parameterized quadratic function using the identity $x_i^2 = x_i$ and other valid quadratic equalities that we introduce from the reformulation of Phase 1. Then, we focus on finding the best parameters to get a quadratic convex program which continuous relaxation's optimal value is maximized. For this, we build a new semi-definite relaxation (SDP) of (QP). Then, we prove that the standard linearization inequalities, used for the quadratization step, are redundant in presence of the new quadratic equalities. Next, we deduce our optimal parameters from the dual optimal solution of (SDP). The third phase consists in solving (QP^*), the optimally reformulated problem, with a standard solver. In particular, at each node of the branch-and-bound, the solver computes the optimal value of a continuous quadratic convex program.

In Section 4.2, we focus on the quadratization phase. Indeed, the continuous relaxation bound value of (QP^*) is dependent on the scheme of quadratization and thus varies for different 2×2 quadratic reformulations. The purpose of this section is to try to classify the quadratizations depending on their convexification bound values. We first define the property of *stability* for a family of quadratizations, that characterizes a set of quadratizations having the same continuous relaxation bound value after convexification. We then prove that, if two different quadratizations introduce the same number of variables and if each added variable represents the same product, then the two quadratizations lead to the same bound after convexification. Next, we show that adding more variables to a quadratization can help improving the bound. Thereby, we present two special cases of quadratization, namely the *full* and the *partial* quadratizations. The full quadratization displays a theoretical interest as it allows to link `PQCR` and the Lasserre's hierarchy. On the other hand, the partial quadratization has a purely practical interest since it has a good trade-off between the quality of the bounds and the dimension of the problems. Finally, we experimentally compare the bounds obtained by different 2×2 quadratization schemes.

We end this chapter by presenting the connection between `PQCR` and the Lasserre's hierarchy for unconstrained binary polynomial programs in Section 4.3. We show that, when applying `PQCR` with the full quadratization, one gets the same semi-definite relaxation as the first order of the Lasserre's hierarchy. Then, we claim that for each order k of the Lasserre's hierarchy, there exists a quadratization phase of method `PQCR`, which semi-definite relaxation corresponds to this order k semi-definite relaxation. This naturally leads to a hierarchy of quadratic convex reformulations based on the Lasserre's hierarchy. This hierarchy inherits all the nice properties of the Moment/SOS hierarchy defined by Lasserre including the finite convergence property. Finally, we show that `PQCR` , with different 2×2 quadratizations, can be quite tractable for unconstrained binary polynomial optimization as it does not require to reformulate all the products of two variables of a given degree d. Indeed, the quadratic reformulation performed in `PQCR` may reformulate the products appearing in the initial problem only. This leads to a better trade-off between the

quality of the bound and the solution time.

Chapter 5: Convexification with other quadratization schemes In this chapter, we study the compatibility between PQCR and other quadratizations described in the literature. Indeed, as Chapter 4 is restricted to quadratizations using Fortet's constraints [28], here we build new convex reformulations that are more suited to each quadratization.

We begin with Rosenberg's procedure in Section 5.1. Given a polynomial f in binary variables, this quadratization introduces new variables and adds a penalty term to f. More precisely, we remark that for a new variable y_{ij} replacing the product $x_i x_j$, the equality $R(x, y) = x_i x_j - 2x_i y_{ij} - 2x_j y_{ij} + 3y_{ij} = 0$ is always satisfied. The idea is to enforce this equality through a penalty term P in a new function $f_p(x, y) = f(x, y) + P \times R(x, y)$, where $f(x, y)$ is a quadratic function in which each y_{ij} variable replaces the corresponding product $x_i x_j$ and P is a large positive number. For all the optimal solutions, $f_p(x, y) = f(x, y)$. Furthermore, one of the characteristics of Rosenberg's procedure is to quadratically reformulate f without adding any constraints. However, after the convexification phase, this property does not hold as the classical linearization constraints are required to enforce the equivalence to (P). We thus introduce a first diagonal convex reformulation for such quadratizations using the equality $x_i^2 = x_i$, which amounts to applying the QCR method [13]. We then present a new convexification that is adapted from the Rosenberg's procedure. It consists in using $R(x, y) = 0$ as a valid equality in order to compute a convex reformulation. We then compare the impact of this family of valid equalities with the one used in PQCR.

In Section 5.2, we derive a new convex reformulation for the quadratization introduced in [5]. In the same way as Rosenberg's procedure, the authors introduce a quadratic function that is equal to f on all the optimal points by introducing additional variables and monomials. The only valid equality on the domain is $x_i^2 = x_i$ again and we build a convexification perturbing only the diagonal of the Hessian matrix.

In Section 5.3, we apply convex reformulation to termwise quadratizations. This family of quadratizations introduces a quadratic reformulation for each monomial separately. We apply the standard diagonal convexification using $x_i^2 = x_i$.

Chapter 6: Semi-definite relaxations of box-constrained polynomial programs In this chapter, we compute tight convex relaxations to continuous polynomial optimization and in particular to box-constrained problems (P_c) where the variables are contained in real intervals. Observe that there is no simple way of deriving a convex quadratic reformulation as it was done in the discrete case. Indeed, the equality $y_{ij} = x_i x_j$ cannot anymore be linearized equivalently. Moreover, some of the equalities used in PQCR are not valid anymore, but they can be replaced by inequalities. In this context, we propose quadratic reformulation of (P_c). We then present two semi-definite relaxations for (P_c). We introduce a first compact relaxation that we further improve to reach tight bounds.

Chapter 7: Computational experiments and implementations This chapter provides experimental results for the methods that are introduced in this thesis. We present computational results where we compare our main method `PQCR` introduced in Chapter 4 with other convexification methods, and with the solver `Baron` [73]. We evaluate our method on instances of the image restoration problem [24] and the low auto-correlation binary sequence problem [9] from `minlplib` [63]. For this last problem, 33 instances among the 45 were unsolved in `minlplib`. We solve to optimality 6 of them, and for the 27 others, we improve primal and/or dual bounds. We compare and analyze the different convex reformulations introduced in Chapter 5 in terms of bounds and number of variables and we justify the choice of a 2×2 quadratization. In particular, we compare these new convexifications with a restricted version of `PQCR` where the only valid equality used is $x_i^2 = x_i$. Finally, we illustrate the tightness of our semi-definite relaxations introduced in Chapter 6 on random box-constrained polynomial instances.

Chapter 8: Conclusion We end this study with a brief overview of the main results presented in this thesis. We also present perspectives for future work in the continuity of those already carried out. In particular, we consider constrained binary polynomial programs and we try to connect this family of problems with our general `PQCR` framework. Next, we propose an approach to take into account integer variables as the discrete part of our study only focuses on binary variables. Finally, we address the problem of general mixed-integer polynomial programs.

Chapter 2

State-of-the-art

Contents

2.1 Polynomial optimization **28**

2.2 Unconstrained polynomial optimization with binary variables . . . **32**

2.3 Quadratic reformulation of binary polynomial programs **38**

2.3.1 Quadratizations with no additional constraints 40

2.3.2 Quadratizations with additional constraints 44

In this chapter we are interested in the state of the art of polynomial optimization. We begin this review with methods dedicated to the general problem (P_G) defined in (1.1). Last, we consider the restriction of (P_G) to unconstrained binary polynomial optimization problems. The goal in this chapter is to describe some of the methods available in the literature.

2.1 Polynomial optimization

In this section, we describe theoretical and practical methods devoted to solve (P_G). We also present commercial and non-commercial solvers that can handle this problem at the end of this section.

Convex envelops and linear relaxations One of the families of methods that are able to solve (P_G) to optimality are spatial Branch and Bound algorithms based on convex relaxations. The most classical relaxation consists in computing linear relaxations of (P_G), but non-linear convex relaxations can also be used. To this end, several works revolve around characterizing the convex envelope of a given expression. In fact, we can derive the exact convex envelope in some special cases. For example, in [60], McCormick describes the convex envelope of a bilinear term $x_i x_j$ of bounded variables. This result is applied to solve factorable problems which is a

very large class of problems including polynomial optimization problems with box constraints. In this context, the paper introduces an algorithm to compute convex relaxations of both the objective function and the constraints. For this, the author considers several intervals of the feasible box. In each interval, one can compute a convex envelope if the objective function or the constraints are not already convex in this interval. Let x be a continuous vector contained in the box $[l, u] = \prod_i [l_i, u_i]$. Let y_{ij} be the additional variable representing the product $x_i x_j$. The McCormick inequalities corresponding to the linearization of the equality $y_{ij} = x_i x_j$ are written as follows:

$$\begin{cases} y_{ij} \geq l_i x_j + x_i l_j - l_i l_j \\ y_{ij} \geq u_i x_j + x_i u_j - u_i u_j \\ y_{ij} \leq u_i x_j + x_i l_j - u_i l_j \\ y_{ij} \leq x_i u_j + l_i x_j - l_i u_j \end{cases}$$

For trilinear terms (product of 3 different variables), recursive application of the previous envelope for bilinear terms is sufficient to obtain a convex relaxation. However, Meyer et al. [61] introduced linear constraints that fully describe the convex envelope of trilinear terms. They also proved that the recursive use of the bilinear envelope rarely yields the convex envelope in the case of a trilinear term.

Quadrilinear terms have been studied in [22]. For this special case, there is no explicit characterization of the convex envelope so far. However, the authors exploit the convex envelopes of bilinear and trilinear terms to compute tight relaxations for the quadrilinear case. More precisely, given a quadrilinear term $x_1 x_2 x_3 x_4$, they consider four types of term groupings and they apply the previously mentioned convex envelope:

- $((x_1 x_2) x_3) x_4$ where the bilinear envelope is applied three times
- $(x_1 x_2)(x_3 x_4)$ where the bilinear envelope is applied three times
- $(x_1 x_2 x_3) x_4$ where the trilinear envelope is applied on $x_1 x_2 x_3$ and then the bilinear envelope is applied to the resulting product
- $(x_1 x_2) x_3 x_4$ where the bilinear envelope is applied on $x_1 x_2$ and then the triilinear envelope is applied to the resulting product

They prove that the last two groupings lead to tighter relaxations than the first two.

For general multilinear terms, we can cite [78], where the authors explicit the convex envelope of fractional terms of the form x/y. They also provide an important property of the convex envelope of multilinear functions which include g. Let g_c be the convex envelope of a multilinear function g. They prove that $g_c(x) = g(x)$ only for the points x belonging to the faces over which g

is linear. Outside of these points, we have $g_c(x) \neq g(x)$. Further convex relaxations for signomial terms (polynomials with real exponents) are presented in [57].

$\alpha-$branch-and-bound and non-linear convex underestimators For general non-convex functions, the $\alpha-$branch-and-bound algorithm [2] computes convex underestimators by perturbing the diagonal of the Hessian matrix of the objective function. More precisely, given a polynomial f and n box-constrained continuous variables $x_i \in [l_i, u_i]$, the authors build the parameterized function $\mathcal{L}_\alpha(x) = f(x) + \sum_{i=1}^n \alpha_i(x_i - l_i)(x_i - u_i)$. The choice of α is motivated by making $\mathcal{L}_\alpha$ convex, which is equivalent to its Hessian matrix being positive semi-definite. Trivially, large values of α suffice to attain the convexity. However, one should choose α carefully as we want to obtain tight bounds when using $\mathcal{L}_\alpha$ as a relaxation of f in a branch-and-bound algorithm. Note that the parameter α is then recomputed at each node. Moreover, the main difficulty here is that the Hessian matrix of f is non-constant whenever the degree of f is greater than 2. In this context, it is proven in [2] that one of the most efficient computation of α is provided by the Gerschgorin Theorem and will be detailed in section 3.2.

Several improvements have been made to the general framework of $\alpha-$branch-and-bound. In [58], the authors obtain a convex relaxation of (P_G) by reducing it beforehand to a DC programming problem where both the constraints and the objective function are decomposed into the difference of two convex functions. In [62], the authors propose the *spline* $\alpha-$branch-and-bound where they use piecewise quadratic perturbations of f to compute the overall convex underestimator. More precisely, the initial box $B = [l_i, u_i]$ is partitioned into smaller boxes $B_k, k = 1, ..., s$, and in each box B_k a convex underestimator f_k is computed using the previous method. Moreover, all the convex underestimators should be continuous on the boundaries of their respective box. In [37], the authors compute each underestimator f_k independently, then a global underestimator is built from the f_k's. Finally, in [53], the authors use the Lasserre hierarchy to deduce a sequence of convex underestimators to approximate a given polynomial.

Separable underestimator In [20], Buchheim et al. compute a separable underestimator for f. More precisely a separable function g is such that $g(x) = \sum_{i=1}^n g_i(x_i)$, where each g_i is a function of variable x_i only. As a result, finding the minimum of g amounts to finding the minimum of each of the g_i's. Formally, one wants to find a separable function g such that for all $x \in [l, u],\ g(x) \leq f(x)$. They first prove that every polynomial f of degree d admits a separable underestimator of degree $\bar{d}$. For this it suffices to underestimate each monomial of f separately. For a monomial $f = c\prod_i x_i^{\alpha_i}$ of degree d, we consider two cases:

- If d is even, then the we have
$$f(x) \geq -\frac{|c|}{d}\sum_{i=1}^n \alpha_i x_i^d.$$

- If d is odd, then the we have

$$f(x) \geq -\frac{|c|}{2d} \sum_{i=1}^{n} \alpha_i (x_i^{d+1} + x_i^{d-1}).$$

Once they have proved the existence of a global separable underestimator, the authors focus on tight underestimators. They explicit optimal separable underestimators of every monomial of degree $d \leq 4$. They have implemented a solver named `Polyopt` which performs very well on continuous random instances. For integer programs it is a bit slower than other solvers but it still manages to solve every instance up to 40 variables and 400 monomials.

Hierarchies Other families of methods consists in a hierarchy of programs. Some of them are based on linear programming. This is the case for the RLT hierarchy [74]. This hierarchy is based on lift and project methods. More precisely, starting from a linear relaxation of (P_G), it consists in multiplying the constraints by each variable and then adding auxiliary variables to linearize the constraints. The relaxation bound value is then tightened. This process is repeated iteratively at each step of the hierarchy, again by increasing the dimension of the relaxation. The authors also proved that the optimal value of (P_G) is attained in a finite number of iterations. This hierarchy has been recently improved in [26].

More recently, Lasserre proposed in [52] an algorithm based on a hierarchy of semi-definite relaxations of (P_G) called Moment/SOS hierarchy. The idea is, at each rank of the hierarchy, to successively tighten semi-definite relaxations of (P_G) by increasing the dimension in order to reach its optimal solution value. It is also proven in [52] that this hierarchy converges in a finite number of iterations to the optimal solution of the considered problem. Further, this work has been extended to hierarchies of second order conic programs [3, 33, 48], and of sparse doubly non-negative relaxations [44]. For additional materials on semi-definite hierarchies, we refer the reader to seminal surveys such as [4]. We will further develop the Lasserre's hierarchy in detail for the special case of binary polynomial optimization in Section 2.2.

For the special case of binary polynomial programs, an interesting comparison between all these hierarchies can be found in [55]. The authors shows that the Lasserre's hierarchy provides the tightest relaxations of (P_G).

Commercial and non-commercial solvers There exist many commercial and non-commercial solvers that can handle (P_G) and they include implementations of some of the algorithms previously mentioned. Among them we can cite `Baron` [73], which is a commercial solver that can handle purely continuous, purely integer or mixed integer problems. It solves non-convex problems to optimality thanks to a branch-and-reduce algorithm. This algorithm is based on bound reducing techniques. For a variable $x \in [l, u]$, it consists in finding a new reduced interval

$[l', u'] \subseteq [l, u]$ containing x and then propagating this reduction to other variables recursively. For comparison purpose, we used `Baron` for our experiments.

`Antigone` [64] is a mixed integer nonlinear program (MINLP) solver based on a branch-and-cut algorithm. Given a MINLP, it first detects special structures of the problem such as convexity, separability, etc. Then it computes convex relaxations of non convex problems at each node of the branch-and-bound.

`SCIP` [1] is also a solver dedicated to MINLP. It relies on linear relaxations of MINLP alongside with cutting planes. It focuses on having strong dual bounds in order to reduce the number of nodes.

Next, `Couenne` [8] is a solver for factorable MINLP. It implements linearization, bound reduction and branching methods within a spatial branch-and-bound framework. It also implements convex under and over-estimators for general non convex program.

One can find a comparison between all these solvers on low autocorrelation instances (1.2.2) in the MINLPLIB website [63].

Finally, we can cite the `GloptiPoly` solver [41] which implements the Moment/SOS method that we described. It is tailored for small and medium-sized instances.

2.2 Unconstrained polynomial optimization with binary variables

A well studied restriction of polynomial optimization is the unconstrained binary polynomial optimization problem. The following program is the general form of such an optimization problem.

$$(P) \begin{cases} \min f(x) \\ \text{s.t.} \\ \quad x \in \{0,1\}^n \end{cases}$$

Unconstrained binary polynomial optimization (also known as *pseudo-Boolean optimization*), although being a special case of polynomial optimization, is still a general framework that can model various optimization problems.

Methods and applications for the quadratic case The special case where the polynomial objective function of (P) is a quadratic function (i.e. $d = 2$) has been widely studied. Padberg defined the class of boolean quadric polytopes in [66] which was followed by a comparison between the cut polytope and the quadric polytope in [76]. These papers introduced interesting resolution tools for $d = 2$ based on cutting plane algorithms. In this case, (P) has many applications, including those from financial analysis [54], cluster analysis [69], computer aided design [47] or machine scheduling [70]. Quadratic programming with binary variables has also been used recently in

quantum programming [36]. Moreover, many graph combinatorial optimization problems such as determining maximum cliques, maximum cuts, maximum vertex packing or maximum independent sets can be formulated as quadratic optimization problems [6, 19, 68]. Finally, we explicit below an efficient method that has proven its worth on binary quadratic programming.

Quadratic Convex Reformulation method (`QCR`) In [13], Billionet et al. consider problem (P) with f being a quadratic function. They compute an equivalent program to (P) where the continuous relaxation is convex. Then this continuous relaxation is solved at each node of a branch-and-bound algorithm. For this, they first introduce a parameter $\alpha \in \mathbb{R}^n$ and they compute the following equivalent program to (P) parameterized by α:

$$(P_\alpha) \begin{cases} \min f_\alpha(x) = f(x) + \sum_{i=1}^{n} \alpha_i (x_i^2 - x_i) \\ \text{s.t.} \\ \quad x \in \{0,1\}^n \end{cases}$$

Let us observe that it is possible to choose values of α that makes f_α convex. Indeed, f_α being convex is equivalent to the Hessian matrix H_α of f_α being positive semi-definite. Thereby, setting each component of α to large positive values would be sufficient to attain the convexity. However, the continuous relaxation bound value would be very weak and thus the resolution by branch-and-bound would be slow. The goal is to compute values of α that maximizes the continuous relaxation bound value and that makes f_α convex. Billionet et al. prove that such a vector α satisfying these two criteria can be computed thanks to a semi-definite relaxation of (P). The authors also consider linearly constrained quadratic programs and they derive a quadratic convex reformulation scheme for this case. Later, this approach has been extended to mixed-integer and quadratically constrained problems in [11] and we refer to this new method as `MIQCR`.

Methods and applications for $d > 2$ In the cubic case (i.e. $d = 3$), the important class of satisfiability problems known as 3-SAT, can be formulated as (P) [45]. In the case where $d \geq 3$, there also exists many applications including, for example: the construction of binary sequences with low aperiodic correlation [9] that is one of the most challenging problems in signal design theory, or the image restoration problem in computer vision [24]. We detail these two applications in Section 1.2.

Problem (P) is also $\mathcal{NP}$-hard [31]. Practical difficulties come from the non-convexity of f and the integrality of its variables. During the last decades, several algorithms that can handle (P) were introduced. In particular, methods were designed to solve the more general class of mixed-integer nonlinear programs. These methods are branch-and-bound algorithms based on a convex relaxation of (P). More precisely, in a first step a convex relaxation is designed and then a branch-

and-bound is performed based on this relaxation. Thus, algorithms presented for more general cases of polynomial optimization in the Section 2.1 can be used to solve (P). The most classical relaxation consists in the standard linearization of (P) [28, 81]. This standard linearization is used to linearly reformulate products of binary variables by adding auxiliary variables. More precisely, for each monomial $m = \alpha \prod_{i=1}^{d} x_i$ we add one additional y that represents the monomial. This is detailed in the following set of constraints:

$$\begin{cases} y = \prod_{i=1}^{d} x_i, \\ x \in \{0,1\}^d \\ y \in \{0,1\} \end{cases}$$

Observe that m is always multilinear as we consider binary variables. The corresponding standard linearization is the following set

$$\begin{cases} y \leq x_i, \quad i = 1, .., d \\ y \geq \sum_{i=1}^{d} x_i - d + 1 \\ y \geq 0 \\ x \in \{0,1\}^d \end{cases}$$

The two sets are equivalent and the standard linearization gives a linear reformulation of any binary polynomial programs by introducing additional variables and constraints.

This linearization was improved by Glover et al. in [34, 35] where the authors present equivalent formulations with a reduced number of variables and constraints. Further improvements can be found in [82, 75]. In [24] Crama et al. introduce valid inequalities for the standard linearization. More precisely, let us consider two monomials indexed by subsets S and T and two variables y_S, y_T such that $y_S = \prod_{i \in S} x_i$ and $y_T = \prod_{i \in T} x_i$. Then the following $2-$link inequality associated with (S, T) is valid for the standard linearization polyhedron

$$y_S \leq y_T - \sum_{i \in T \setminus S} x_i + |T \setminus S|.$$

Moreover they also prove that these $2-$link inequalities are facet defining in the case of nested monomials and in the case of a polynomial with 2 nonlinear monomials. We link this valid inequalities with our approach in Section 3.1. These inequalities are generalized in [27]. Quadratic reformulation methods have also been studied widely. We will mention some well-known quadratic reformulations in the next section.

The Lasserre's hierarchy for binary polynomial optimization This method has been introduced to solve general polynomial problems using a hierarchy of relaxations. More precisely, from the dual point of view, the general idea of the Moment/SOS method is to relax the initial problem and then solve semi-definite relaxations whose sizes increase at each iteration. At the end of the process, the last relaxation considered actually solves the initial problem. Indeed, several results from algebraic geometry are used to prove that such a sequence of relaxations converges towards the overall optimum [52]. Although this hierarchy is designed for continuous optimization, one can solve binary polynomial programs by considering the constraints $x_i^2 - x_i = 0$. In the following we present the dual version of the hierarchy for unconstrained binary polynomial programs.

We start by noticing that the problem (P) is equivalent to a linear infinite dimensional problem

$$(P_{eq}) \begin{cases} \min\limits_{\mu \in \mathcal{M}^+(\{0,1\}^n)} \int_{\{0,1\}^n} f d\mu \\ \text{s.t.} \\ \quad \mu(\{0,1\}^n) = 1 \end{cases}$$

where $\mathcal{M}^+(\{0,1\}^n)$ is the set of positive measures over $\{0,1\}^n$. The difficulty of this equivalent program lies in the characterization of the $\mathcal{M}^+(\{0,1\}^n)$ set. Indeed, it is difficult to optimize on all positive measures on K since it is an infinite dimensional set and also we do not know an effective characterization of this set. The author proposes an alternative equivalent reformulation which is more tractable. For this Lasserre uses the moment matrix.

Let $y_\alpha,\ \alpha \in \mathbb{N}^n$ be a finite sequence of real numbers. We can construct the moment matrix associated with y as follows.

Definition 2.2.1 (Moment matrix)**.** *Let $y = (y_\alpha)$ be a finite sequence of real numbers, $\alpha \in \mathbb{N}^n$. We define the moment matrix of dimension r, the matrix*

$$M_r(y)(\alpha, \beta) = y_{\alpha+\beta} \quad \forall \alpha, \beta \in \mathbb{N}_r^n.$$

The moment matrix can be thought as the matrix $\begin{pmatrix} 1 & y^T \\ y & Y \end{pmatrix}$ where Y_{ij} represents a linearization of the product $y_i y_j$.

One way to remove the difficulty from the objective function of (P_{eq}) is to reformulate it in order to minimize over a finite dimensional set. By reformulating the objective function and by moving the difficulty (set of positive measures over $\{0,1\}^n$) to the constraints we obtain an equivalent program to (P_{eq})

$$(P_{eq})\begin{cases} \min\limits_{y\in\mathbb{R}^{2^n}} \sum\limits_{\alpha\in\mathbb{N}^n} f_\alpha y_\alpha \\ \text{s.t.} \\ \quad y_\alpha = \int_{\{0,1\}^n} x^\alpha d\mu \quad \forall \alpha \in \mathbb{N}^n, \text{ for some } \mu \in \mathcal{M}(\{0,1\}^n) \\ \quad \mu(\{0,1\}^n) = 1 \end{cases} \tag{2.1}$$

Now the minimization is over the real vector y_α and the constraints consist in finding a measure μ satisfying a linear constraint which makes the problem more tractable than previously. The existence of a measure μ supported on $\{0,1\}^n$ and satisfying (2.1) is a well-known problem called the *truncated moment problem.* One sufficient condition for the existence of such a measure is the positive semi-definiteness of the moment matrix $M_{\lceil \frac{d}{2} \rceil}$. Problem (P_{eq}) can thus be rewritten

$$(P_{eq})\begin{cases} \min\limits_{y\in\mathbb{R}^{s(d)}} \sum\limits_{\alpha\in\mathbb{N}^n} f_\alpha y_\alpha \\ \text{s.t.} \\ \quad y_0 = 1 \\ \quad M_n(y) \succeq 0 \end{cases}$$

where $s(d) = \binom{n+r}{n}$ is the dimension of vector y_α. In example 2.2 below, we explicit y_α. Unfortunately, this problem is very hard to solve because of the size of the moment matrix. The idea of the Lasserre's hierarchy is to relax the semi-definite constraints. We start with a moment matrix of size $\lceil \frac{d}{2} \rceil$ and we solve the obtained problem. We then increase this size at each iteration. We obtain a hierarchy where the relaxation of order r, $\lceil \frac{d}{2} \rceil \leq r \leq n$ is of the form

$$(M_r)\begin{cases} \min\limits_{y\in\mathbb{R}^{2^n}} \sum\limits_{\alpha\in\mathbb{N}^n} f_\alpha y_\alpha \\ \text{s.t.} \\ \quad y_0 = 1 \\ \quad M_r(y) \succeq 0 \end{cases}$$

It is proven that the optimal value of these relaxations converges finitely to the optimal value of (P) in n iterations [51]. This method aims at solving small to medium sized binary polynomial instances. The Moment/SOS method is implemented in the solver GloptiPoly [41]. Unfortunately, the solver is not able to solve the Visions instance (1.2.1) or the low autocorrelation instances (1.2.1) within one hour. We illustrate the behavior of the Lasserre's hierarchy for binary polynomial programs on a small instance.

Example 2. *Let us consider the following problem which will be the common thread of the*

different methods introduced in this thesis:

$$(Ex)\left\{ \min_{x\in\{0,1\}^4} 2x_1 + 3x_2x_3 - 2x_2x_3x_4 - 3x_1x_2x_3x_4 \right.$$

The optimal value of (Ex) *is* 0.

We apply the Lasserre's hierarchy to (Ex). *With the Lasserre's notation, the initial variables* x_1, x_2, x_3 *and* x_4 *are represented by* $y_{1000}, y_{0100}, y_{0010}$ *and* y_{0001} *respectively, where* y_p *represents the monomial* $\prod_{i=1}^n x_i^{p_i}$. *The first iteration corresponds to the order* $\lceil \frac{d}{2} \rceil = 2$ *and gives the following program*

$$(M_2)\begin{cases} \min\limits_{y\in\mathbb{R}^{10}} 2y_{1000} + 3y_{0110} - 2y_{0111} - 3y_{1111} \\ s.t. \\ \quad y_{0000} = 1 \\ \quad M_2(y) \succeq 0 \end{cases}$$

Here $M_2(y)$ *is a* 11×11 *matrix of the form* $\begin{pmatrix} 1 & y^T \\ y & Y \end{pmatrix}$ *where* y *is a vector of dimension* 10 *containing the* 4 *initial variables and* 6 *additional variables representing the products of degree* 2 $(y_{1100}, y_{1010}, y_{1001}, y_{0110}, y_{0101}, y_{0011})$. *For this example,* $M_2(y)$ *writes*

$$M_2(y) = \begin{pmatrix}
y_{0000} & y_{1000} & y_{0100} & y_{0010} & y_{0001} & y_{1100} & y_{1010} & y_{1001} & y_{0110} & y_{0101} & y_{0011} \\
y_{1000} & y_{1000} & y_{1100} & y_{1010} & y_{1001} & y_{1100} & y_{1010} & y_{1001} & y_{1110} & y_{1101} & y_{1011} \\
y_{0100} & y_{1100} & y_{0100} & y_{0110} & y_{0101} & y_{1100} & y_{1110} & y_{1101} & y_{0110} & y_{0101} & y_{0111} \\
y_{0010} & y_{1010} & y_{0110} & y_{0010} & y_{0011} & y_{1110} & y_{1010} & y_{1011} & y_{0110} & y_{0111} & y_{0011} \\
y_{0001} & y_{1001} & y_{0101} & y_{0011} & y_{0001} & y_{1101} & y_{1011} & y_{1001} & y_{0111} & y_{0101} & y_{0011} \\
y_{1100} & y_{1100} & y_{1100} & y_{1110} & y_{1101} & y_{1100} & y_{1110} & y_{1101} & y_{1110} & y_{1101} & y_{1111} \\
y_{1010} & y_{1010} & y_{1110} & y_{1010} & y_{1011} & y_{1110} & y_{1010} & y_{1011} & y_{1110} & y_{1111} & y_{1011} \\
y_{1001} & y_{1001} & y_{1101} & y_{1011} & y_{1001} & y_{1101} & y_{1011} & y_{1001} & y_{1111} & y_{1101} & y_{1011} \\
y_{0110} & y_{1110} & y_{0110} & y_{0110} & y_{0111} & y_{1110} & y_{1110} & y_{1111} & y_{0110} & y_{0111} & y_{0111} \\
y_{0101} & y_{1101} & y_{0101} & y_{0111} & y_{0101} & y_{1101} & y_{1111} & y_{1101} & y_{0111} & y_{0101} & y_{0111} \\
y_{0011} & y_{1011} & y_{0111} & y_{0011} & y_{0011} & y_{1111} & y_{1011} & y_{1011} & y_{0111} & y_{0111} & y_{0011}
\end{pmatrix}$$

The optimum value of this relaxation is $-0,015$. *As the optimum of* (Ex) *is not attained, we*

increment the order and we compute the relaxation of order $r = 3$.

$$(M_3)\begin{cases} \min\limits_{y\in\mathbb{R}^{14}} 2y_{1000} + 3y_{0110} - 2y_{0111} - 3y_{1111} \\ s.t. \\ \quad y_{0000} = 1 \\ \quad M_3(y) \succeq 0 \end{cases}$$

Here $M_3(y)$ is a 15×15 matrix where y is a vector of dimension 14 containing all the previous variables and 4 additional variables representing the products of initial variables of degree 3. The optimum value of this relaxation is 0 which is the optimum value of (P). The hierarchy stops at order 3 as the optimal solution is attained.

In the following section we present in detail some state of the art quadratic reformulations that will be used in this thesis.

2.3 Quadratic reformulation of binary polynomial programs

In this section we review the literature concerning the solution of binary polynomial programs through quadratic reformulation. It consists in building a quadratic equivalent formulation to (P). The obtained reformulation can then be solved by several commercial and non-commercial solvers, since we consider quadratic problems with binary variables only. In particular, on top of the solvers designed for general quadratic programs, we can also cite `Cplex` [42] which can handle binary quadratic programs. `Cplex` uses a branch-and-cut algorithm after a custom convex reformulation of the binary quadratic program. Note that the idea of quadratization is also used for continuous variables for example in [25]. Once they have the equivalent quadratic reformulation to (P), the authors then deduce RLT based linear programming relaxations.

Pairwise covers Most of the quadratizations that will be presented rely on a prior phase that consits in building a 2×2 quadratization scheme or a *pairwise cover* as it was introduced by the authors of [5]. The idea is to reformulate selected products of variables by a new one in order to find a quadratic reformulation of f. Let $\mathcal{M}$ be the set of monomials of f. The authors associate $\mathcal{M}$ with its standard hypergraph, where the vertices are the n variables of the inital problem. Each edge of $\mathcal{M}$ represents a monomial, and contains the variables of this monomial. To this hypergraph, they add auxiliary edges forming the pairwise cover $\mathcal{H}$. $\mathcal{H}$ is a hypergraph such that for each monomial m of degree greater than 2, there are two edges $A(m), B(m) \in \mathcal{H}$ "covering" the monomial m, i.e. $|A(m)| < |m|$, $|B(m)| < |m|$ and $A(m) \cup B(m) = m$. In other words, every variable of a monomial m is covered by at least the edge $A(m)$ or the edge $B(m)$. By introducing additional variables y_A and y_B for the product of variables contained in $A(m)$

and $B(m)$ respectively, we obtain the product $y_A y_B$ which is a quadratic reformulation of the monomial m. We say that $A(m)$ and $B(m)$ form a pairwise cover of the monomial m. The example below illustrates several pairwise covers for a small instance.

Example 2 (continued). *We present different pairwise covers leading to different quadratization of* (Ex)

- *In quadratization* l_1*, the third monomial is covered by* x_6x_3 *whereas the fourth monomial is covered by* x_7x_8*:*

  $$l_1(x) = 2x_1 + 3x_2x_3 - 2\underbrace{x_2x_4}_{x_6}x_3 - 3\underbrace{x_1x_2}_{x_7}\underbrace{x_3x_4}_{x_8}$$

  and we have

  $$\mathcal{H}_1 = \{\{2,4\},\{3\},\{1,2\},\{3,4\}\}$$

- *In quadratization* l_2*, the third monomial is covered by* x_6x_4 *whereas the fourth monomial is covered by* x_8x_4*:*

  $$l_2(x) = 2x_1 + 3x_2x_3 - 2\underbrace{x_2x_3}_{x_6}x_4 - 3\underbrace{\underbrace{x_1x_2}_{x_7}x_3}_{x_8}x_4$$

  and we have

  $$\mathcal{H}_2 = \{\{2,3\},\{3\},\{1,2\},\{1,2,3\},\{4\}\}$$

This covering property of the sets $A(m)$ and $B(m)$ and the necessary condition $\mathcal{H} \subset \mathcal{M}$ make it very similar to the idea of reducibility [21] that will be mentionned hereunder. However, the main purpose of the authors here is to compute a quadratization of a polynomial program by introducing auxiliary variables associated to elements in the pairwise cover (here y_A and y_B for the monomial m). The authors were also interested in finding a pairwise cover with small cardinality so as to introduce a small number of auxiliary variables. Unfortunately finding the smallest pairwise cover is NP-hard [18]. Nevertheless we can find in [71] several quadratizations heuristically built by introducing a small number of additional variables. In Section 5.2, we will be interested in the impact of the choice of a pairwise cover for our particular framework.

Once a pairwise cover is defined, one can apply several quadratizations, i.e. different ways of enforcing the equality $m = y_A y_B$, either for any solutions or for optimal solutions only. Note that we specifically focus on quadratic reformulation since we later introduce in Chapter 4 an algorithm that relies on a quadratic reformulation. We make the distinction between two families of quadratizations: those introducing constraints and those who do not introduce constraints in the reformulated problem.

2.3.1 Quadratizations with no additional constraints

In [5], Anthony et al. define the concept of *quadratization*. More formally, given a binary polynomial f with a monomial set $\mathcal{M}$ containing M monomials and $x \in \{0,1\}^n$, $g(x,y)$ is a quadratization of f with N additional variables if $g(x,y)$ is a quadratic function of x and y where $y \in \{0,1\}^N$ are auxiliary binary variables such that

$$\forall x \in \{0,1\}^n, \quad f(x) = \min_{y \in \{0,1\}^N} g(x,y).$$

Throughout this section, we call quadratization the particular framework introduced in [5] that builds a quadratic reformulation without adding constraints.

Termwise quadratizations Termwise quadratizations consist in reformulating each monomial independently with a quadratic function using both initial and additional variables. In this context, the goal is to compute a quadratization $g_m(x, y_m)$ for each monomial m independently. The quadratization $g(x,y)$ of f is then obtained by $g(x,y) = \sum_{m \in \mathcal{M}} g_m(x, y_m)$. Obviously at least $\bar{M}$ additional variables are required for a termwise quadratization, where $\bar{M}$ is the number of monomials of degree greater than 2. When we consider termwise quadratization, we generally distinguish between negative monomials and positive monomials, that are monomials with negative or positive coefficients.

For negative monomials, the authors of [30] and [46] introduced the quadratization of a negative monomial m of degree d_m and with a negative coefficient α,

$$g_m(x,y) = -\alpha \left[(d_m - 1)y - \sum_{i=1}^{d_m} x_i y \right],$$

where $y \in [0,1]$ is a single auxiliary variable. For example, the monomial $-3x_1x_2x_3x_4x_5$ will be quadratized with $\min_y 3(4y - y(x_1 + x_2 + x_3 + x_4 + x_5))$. This quadratization has been used later in [5] and called *standard quadratization* for negative monomials.

The case of positive monomials is more complicated since one needs more additional variables to compute a quadratization. Several works revolve around reducing the number of auxiliary variables. Among them, we can cite [43] which introduced a quadratization for positive monomials using $p = \lfloor \frac{d_m - 1}{2} \rfloor$ auxiliary variables. More precisely, let $m(x) = \prod_{i=1}^{d_m} x_i$ be a positive monomial. The Ishikawa's quadratization is defined by

$$m(x,y) = \sum_{i=1}^{p} y_i (c_{i,d_m}(-|x| + 2i) - 1) + \frac{|x|(|x| - 1)}{2}$$

where $|x| = \sum_{j=1}^{d_m} x_j$ and

$$c_{i,d_m} = \begin{cases} 1, & \text{if } d_m \text{ is odd and } i = p \\ 2, & \text{otherwise} \end{cases}$$

Thus p is an upper bound on the number of auxiliary variables. Another independent proof of this upper bound can be found in [5].

This bound on the number of auxiliary variables was reduced recently by Boros, Crama and Rodríguez-Heck in [15] with a quadratization using only $\lceil \log_2(d_m) \rceil - 1$ auxiliary variables. More precisely, the authors define the following quadratization of $m(x)$

$$m(x,y) = \frac{1}{2}(|x| + 2^l - d_m - \sum_{i=1}^{l-1} 2^i y_i)(|x| + 2^l - d_m - \sum_{i=1}^{l-1} 2^i y_i - 1)$$

where $l = \lceil \log(d_m) \rceil$. They also proved that this is a lower bound on the number of additional variables for the unconstrained quadratization framework. Note that for a monomial of degree 6 or less, the Ishikawa's quadratization and the Boros-Crama-Rodríguez quadratization (which will be called "log quadratization" in the following) lead to the exact same reformulation with the same number of variables (see Remark 10 of [71]). In terms of number of auxiliary variables, it can be explained easily as $\lceil \log(d_m) \rceil - 1 = \lfloor \frac{d_m - 1}{2} \rfloor$ for all $1 \leq d_m \leq 6$. Thus for $m(x) = x_1 x_2 x_3 x_4 x_5$, both reformulations lead to the quadratization

$$m(x,y) = \frac{1}{2}(x_1 + x_2 + x_3 + x_4 + x_5 + 3 - 2y_1 - 4y_2)(x_1 + x_2 + x_3 + x_4 + x_5 + 2 - 2y_1 - 4y_2).$$

In [71], the author ran these quadratizations on several families of instances. More precisely they solve the equivalent quadratic reformulation with `Cplex 12.7`. On low degree sparse random instances, the Ishikawa's quadratization and the log quadratization give the same results and solve every instance within one hour. On higher degree random instances, the log quadratization is faster than the Ishikawa's one, but both methods solve all the instances to optimality within hour. On Vision instances (1.2.1), the two quadratizations solve every instance within 400 seconds. Finally, on the low autocorrelation instances (1.2.2), both methods only solve the 5 smallest instances. It is interesting to notice that on all the considered instances, the standard linearization performs better than both quadratizations.

In the above paragraph we were interested in termwise quadratizations, that is quadratization where each monomial is reformulated independently by introducing one or several additional variables. In the below paragraph we present quadratizations where we focus on reformulating products of several variables up to a certain degree rather than reformulating a whole monomial. An additional variable can thus be used in several monomials.

Rosenberg's procedure We start by introducing the first quadratic reformulation published chronologically. Rosenberg's procedure [72] is based on a perturbation of the polynomial objective function by adding additional variables but without adding constraints. This procedure was the first to use penalties to build equivalent quadratic reformulation to pseudo-Boolean optimization problems. We recall Rosenberg's quadratization. The idea is, given a pseudo-Boolean function f, to reduce iteratively the degree of f. We first define a pairwise cover. Then, at each iteration one or several products $x_i x_j$ from this cover are chosen from a highest-degree monomial of f. These products are then replaced by new variables y_{ij}. To enforce the equality $y_{ij} = x_i x_j$ at all the optimal solutions, a penalty term is added to the objective function. This penalty term is $P(x_i x_j - 2x_i y_{ij} - 2x_j y_{ij} + 3y_{ij})$ where P is a large positive number (for the minimization case). Rosenberg's procedure can thus lead to several quadratic reformulations depending on the choice of the products $x_i x_j$ that will be linearized. The drawback of a direct resolution using this procedure is that the continuous relaxation bound obtained is often weak since the reformulation involves large positive numbers. We illustrate this quadratization in the following example.

Example 2 (continued). *We want to reformulate (Ex) into a quadratic program using the additional variables $y_1 = x_2 x_3$ and $y_2 = x_1 x_4$. By applying Rosenberg's procedure to (Ex), we obtain*

$$\begin{aligned} g(x,y) &= 2x_1 + 3x_2x_3 - y_1x_4 - 3y_1y_2 + P(x_2x_3 - 2x_2y_1 - 2x_3y_1 + 3y_1) \\ &\quad + P(x_1x_4 - 2x_1y_2 - 2x_4y_2 + 3y_2) \end{aligned}$$

Note that for the penalty term P, it suffices to take $P = \max(\sum_{c_i \leq 0} |c_i|, \sum_{c_i \geq 0} c_i)$ where c_i is the coefficient of the ith monomial. Here we can set $P = 5$.

Quadratization using pairwise covers Once a pairwise cover is defined, the authors of [5] build a quadratization. This quadratization, like Rosenberg's procedure, introduces additional monomials to the objective function to obtain a quadratic function that is equal to f on all the optimal points.

More precisely, if $\mathcal{M}$ and $\mathcal{H}$ are two hypergraphs such that $\mathcal{H} \subset \mathcal{M}$ and $\mathcal{H}$ is a pairwise cover of $\mathcal{M}$, then the authors compute a quadratization using $|\mathcal{H}|$ auxiliary variables y_i. Thus, every pseudo-Boolean function of the form $f(x) = \sum_{M \in \mathcal{M}} a_M \prod_{j \in M} x_j$ is such that

$$\left\{ f(x) = \min_{y \in \{0,1\}^{|\mathcal{H}|}} \sum_{M \in \mathcal{M}} a_M y_{A(M)} y_{B(M)} + \sum_{H \in \mathcal{H}} b_H \left(y_H \left(|H| - \frac{1}{2} - \sum_{j \in H} x_j \right) + \frac{1}{2} \prod_{j \in H} x_j \right) \right. \quad (2.2)$$

where $b_H = 0$ for $H \in \mathcal{M} \backslash \mathcal{H}$ and for $H \in \mathcal{H}$ we have

$$\frac{1}{2}b_H = \sum_{\substack{M\in\mathcal{M}:\\ H\in\{A(M),B(M)\}}} \left(|a_M| + \frac{1}{2}b_M\right),$$

In [5], the authors build a quadratization by recursively "covering" a product of two variables by a new one until the objective function is quadratic. The function defined in 2.2 is thus the resulting quadratization.

Note that a major difference with Rosenberg's procedure is that the equality $y_{A(M)} = \prod_{i\in A(M)} x_i$ is not necessarily satisfied on optimal points.

Example 2 (continued). *We take again the previous example and we introduce a quadratic reformulation using the additional variables $y_1 = x_1x_2$ and $y_2 = x_3x_4$. Note that we have to rewrite the objective function and the set $\mathcal{M}$ as follows*

$$f(x) = 2x_1 + 3x_2x_3 - 2x_2x_3x_4 - 3x_1x_2x_3x_4 + 0x_1x_2 + 0x_3x_4 + 0x_2$$

and

$$\mathcal{M} = \{\{1\}, \{2,3\}, \{2,3,4\}, \{1,2,3,4\}, \{1,2\}, \{3,4\}, \{2\}\}$$

We have to add the monomials x_2, x_1x_2 and x_3x_4, since it is necessary to satisfy the condition $\mathcal{H} \subset \mathcal{M}$.

Next, the pairwise cover $\mathcal{H}$ is defined by initial variables contained in monomials of degree greater than 2:

$$\mathcal{H} = \{\{3,4\}, \{2\}, \{1,2\}, \{3,4\}\} = \{\{1,2\}, \{2\}, \{3,4\}\}$$

We can now compute the coefficients of the objective function.
We have $\mathcal{M}\backslash\mathcal{H} = \{\{1\}, \{2,3\}, \{2,3,4\}, \{1,2,3,4\}\}$ which leads to $b_1 = b_{23} = b_{234} = b_{1234} = 0$.

For other coefficients, we use the formula given in (2.2). Thus, $\{3,4\}$ is in the cover of $\{2,3,4\}$ and $\{1,2,3,4\}$, so

$$\frac{1}{2}b_{34} = |a_{234}| + \frac{1}{2}b_{234} + |a_{1234}| + \frac{1}{2}b_{1234} = |-2| + \frac{1}{2}0 + |-3| + \frac{1}{2}0 = 5$$

which leads to $b_{34} = 10$. Similarly, we get $b_2 = 4$ (as $\{2\}$ is in the cover of $\{2,3,4\}$) and $b_{12} = 6$ (as $\{1,2\}$ is in the cover of $\{1,2,3,4\}$).

Finally, the new quadratic objective function writes

$$
\begin{aligned}
f(x) = \min_{\{y_1,y_2\}\in\{0,1\}^2} \quad & 2x_1 + 3x_2x_3 - 2y_1x_2 - 3y_1y_2 \\
& +10\left[y_2(2 - \frac{1}{2} - x_3 - x_4) + \frac{1}{2}x_3x_4\right] \\
& +4\left[x_2(1 - \frac{1}{2} - x_2) + \frac{1}{2}x_2\right] \\
& +6\left[y_1(2 - \frac{1}{2} - x_1 - x_2) + \frac{1}{2}x_1x_2\right]
\end{aligned}
$$

which can be simplified as

$$
\begin{aligned}
f(x) = \min_{\{y_1,y_2\}\in\{0,1\}^2} \quad & 2x_1 + 3x_2x_3 - 2y_1x_2 - 3y_1y_2 \\
& +10\left[y_2(2 - \frac{1}{2} - x_3 - x_4) + \frac{1}{2}x_3x_4\right] \\
& +6\left[y_1(2 - \frac{1}{2} - x_1 - x_2) + \frac{1}{2}x_1x_2\right]
\end{aligned}
$$

since $|H| = 1$ leads to a null "penalty term".

2.3.2 Quadratizations with additional constraints

In this section, we refer to quadratization as a more general framework where one can introduce constraints during the quadratic reformulation. More precisely, here we call quadratization any equivalent quadratic reformulation to (P).

Quadratization using Fortet's inequalities A constrained quadratization scheme can be derived from the Fortet' inequalities [28]. This set of inequalities is a special case of the McCormick's constraints [60] for the bilinear case x_ix_j. Although originally designed for the linearization of a product of two variables, we can recursively build a quadratization based on these inequalities. More precisely, let x_ix_j be a product of two binary variables. We introduce an additional variable y_{ij} for $i < j$ and the following Fortet's constraints ensure the equality $y_{ij} = x_ix_j$ for every $x \in \{0,1\}^n$.

$$
(F)\begin{cases}
y_{ij} \leq x_i, \\
y_{ij} \leq x_j, \\
y_{ij} \geq x_i + x_j - 1, \\
y_{ij} \geq 0,
\end{cases}
$$

This linearization of a single quadratic term can be generalized for the quadratization of a

whole polynomial. Our main algorithm presented in this thesis relies on a quadratization using recursively this substitution by first defining a pairwise cover. More precisely, considering a binary polynomial f, we can substitute each product of two variables of f by a new one by adding the corresponding set of constraints until we get a quadratic function. Again there are several quadratizations that are possible for a same instance and the choice of the products, i.e. the pairwise cover, is important. We address this problem later in this thesis. We illustrate this quadratization in the following example.

Example 2 (continued). *We present the quadratization of* (Ex) *using Fortet's inequalities with* $x_5 = x_2x_3$ *and* $x_6 = x_1x_4$.

$$\begin{cases} \min g(x) = 2x_1 + 3x_2x_3 - 2x_5x_4 - 3x_5x_6 \\ s.t. \\ x_5 - x_2 \leq 0, \\ x_5 - x_3 \leq 0, \\ -x_5 + x_2 + x_3 \leq 1, \\ x_6 - x_1 \leq 0, \\ x_6 - x_4 \leq 0, \\ -x_6 + x_1 + x_4 \leq 1, \\ \quad x \in \{0,1\}^6 \end{cases}$$

Note that another set of inequalities can be derived similarly when considering a product of a binary variable by a continuous one.

Quadratization and reducibility In [21], Buchheim and Rinaldi aim at computing tight linear relaxations of (P). These relaxations can then be used at the root node of a branch-and-cut algorithm. For this, they focus on a polyhedral description of the quadratization of a binary polynomial program. They first build a quadratization of the initial problem. This is followed by a complete linearization, by associating a new variables for each monomial. Finally they improve this linearization by adding cuts. These cuts are deduced from an interpretation of the equivalent quadratic problem as a max-cut problem.

More precisely, from (P) and its associated monomial set $\mathcal{M}$, the authors derive a first equivalent reformulation with a linear objective function.

$$(R_1) \begin{cases} \min h(z) \\ \text{s.t.} \\ \quad z_i = \prod_{a \in M} z_{\{a\}}, \quad \text{for all } M \in \mathcal{M} \\ \quad z \in \{0,1\}^{\mathcal{M}} \end{cases}$$

Starting from this equivalent program, one can relax the non-linear constraints with the standard linearization constraints. However this relaxation would be rather weak in terms of continuous relaxation value. Instead, the authors define a new polyhedron (P^*). First they define the set $\mathcal{M}^* = \{\{I, J\} | I, J \in \mathcal{M} \text{ and } I \cup J \in \mathcal{M}\}$ which represents product of monomials. Then they associate a variable $y_{\{I,J\}}$ for every element $\{I, J\}$ of $\mathcal{M}^*$ and consider the following new reformulation

$$(R_2) \begin{cases} \min h^*(y) \\ \text{s.t.} \\ \quad y_{\{I,J\}} = y_{\{I\}} y_{\{J\}}, \quad \text{for all } \{I, J\} \in \mathcal{M}^*, \ I \neq J \\ \quad y \in \{0, 1\}^{|\mathcal{M}^*|} \end{cases} \tag{2.3}$$

This problem can be thought as the linearization of a binary quadratic optimization problem. Let F^* be the set of feasible solutions of (2.3) and let (P^*) be the convex hull of F^*. The authors prove that, under a weak assumption on $\mathcal{M}$ (the reducibility property detailed further), (P) is isomorphic to a face of (P^*). Moreover, for unconstrained binary polynomials, (P^*) is known to be a boolean quadric polytope. Finally the main purpose of this approach is to use all the well-known valid inequalities and separation techniques of the quadric polytope for the purpose of binary polynomial optimization.

A sufficient condition for these results to hold is that the set of monomials $\mathcal{M}$ is *reducible*, i.e. any monomial $m \in \mathcal{M}$ is the product of two other monomials in $\mathcal{M}$, both of them being strictly included in $\mathcal{M}$. The authors show that every monomial set $\mathcal{M}$ can be made reducible by adding some suitable sets. This corresponds to adding monomials with coefficient 0 to f. They also discuss two possible ways of making an instance reducible: the upwards-completeness and the downwards-completeness. We will link these particular family of reducibility with our work in Section 4.2.

The connection to other approaches comes out as making an instance reducible amounts to introducing additional variables to reduce the degree of the instance to 2 in a "Rosenberg-like" framework. There exist several ways to attain reductibility, however finding the smallest set of monomials to add is NP-hard [18]. We present below an illustration of this method on a small example.

Example 2 (continued). *The set of monomials corresponding to* (Ex) *is*

$$\mathcal{M} = \{\{1\}, \{2, 3\}, \{2, 3, 4\}, \{1, 2, 3, 4\}\}.$$

$\mathcal{M}$ *is made reducible by adding the singletons* $\{2\}$, $\{3\}$ *and* $\{4\}$. *We denote by* $M_1, ..., M_7$ *the elements of this new reducible set. We introduce a new variable* z_i *representing each monomial in* $\mathcal{M}$. *We then obtain the first reformulation.*

$$(R_1)\begin{cases} \min 2z_1 + 3z_5 - 2z_6 - 3z_7 \\ s.t. \\ \quad z_1 = x_1, \\ \quad z_2 = x_2, \\ \quad z_3 = x_3, \\ \quad z_4 = x_4, \\ \quad z_5 = x_2x_3, \\ \quad z_6 = x_2x_3x_4, \\ \quad z_7 = x_1x_2x_3x_4, \\ \quad z \in \{0,1\}^7 \end{cases}$$

Next we build the set $\mathcal{M}^* = \{M_1, ..., M_7, \{M_1, M_6\}, \{M_1, M_7\}, \{M_2, M_3\}, ..., \{M_6, M_7\}\}$ *and consider the following reformulation*

$$(R_2)\begin{cases} \min 2y_{M_1} + 3y_{M_5} - 2y_{M_6} - 3y_{M_7} \\ s.t. \\ \quad y_{\{I,J\}} = y_{\{I\}}y_{\{J\}}, \quad \textit{for all } \{I,J\} \in \mathcal{M}^*,\ I \neq J \\ \quad y \in \{0,1\}^{|\mathcal{M}^*|} \end{cases}$$

This program gives the polyhedron P^* *of* $\mathbb{R}^{|\mathcal{M}^*|}$. *We can then use a cutting plane algorithm based on any separation algorithm in* P^* *to solve* (P).

Finally note that, for this example, the upwards-completeness and the downwards-completeness give the same reducible set

$$\mathcal{M} = \{\{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\},$$

$$\{3,4\}, \{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}, \{1,2,3,4\}\}.$$

Conclusion In this section, we have presented a sample of different quadratizations that are close to the ones presented in this thesis. Quadratizations have been widely studied and there exist plenty of quadratizations with different purposes such as in [16]. Finally we conclude this section by referring to [17] and [71] for more complete surveys on quadratizations.

Chapter 3

One-step reformulation methods for pseudo-Boolean optimization

Contents

3.1 Linear reformulations **49**

3.1.1 Linearization of a single monomial 49

3.1.2 Linearization of a polynomial 53

3.2 Direct convexification **55**

3.2.1 Approximation of the Hessian matrix 55

3.2.2 Convexification by uniform diagonal perturbation 56

3.2.3 Convexification by non-uniform diagonal perturbation 58

3.3 Conclusion **59**

This chapter introduces two special cases of convex reformulations of (P). The goal is to build a single equivalent reformulation to (P) with a convex continuous relaxation and then solve the resulting equivalent program with a branch-and-bound algorithm.

In Section 3.1, we discuss the well-known case of complete linearization, that is a reformulation of (P) where both the objective function and the constraints are linear. Our study is rather theoretical as we compare the continuous relaxation bound values of several linearizations. In particular, we specify the cases where different linearizations lead to the same bound.

In Section 3.2, we study the direct convexification of (P), which consists in modifying the Hessian matrix of f to make it positive semi-definite. This leads to an equivalent reformulation to (P) without additional variables nor constraints. The main difficulty is that the Hessian matrix is a function of x whenever the degree of f is greater than or equal to 3. We present a general method to tackle this problem.

3.1 Linear reformulations

In this section we discuss the linearization of polynomial programs with binary variables. It consists in rewritting f into an equivalent linear function in an extended space. In the literature, we can find plenty of different linearizations, but in this section we are interested by the ones coming from the work on pseudo boolean optimization [28]. This family of linearizations reduces the degree of f by introducing additional variables and constraints. To describe our approach, we show how we linearize a single monomial of degree d. We further extend the approach to the whole polynomial f.

3.1.1 Linearization of a single monomial

We first focus on the reformulation of the optimization problem of a single monomial subject to binary constraints into an equivalent linear program. Let $m(x)$ be the monomial $\alpha \prod_{i=1}^{d} x_i$ and let (P_0) be the following program with one monomial:

$$(P_0) \begin{cases} \min m(x) = \alpha \displaystyle\prod_{i=1}^{d} x_i \\ \text{s.t.} \\ \quad x \in \{0,1\}^d \end{cases}$$

We introduce a new variable x_L and we write the following inequalities:

$$(F) \begin{cases} x_L \leq x_i, \quad i = 1,..,d \\ x_L \geq \displaystyle\sum_{i=1}^{d} x_i - d + 1 \\ x_L \geq 0 \end{cases}$$

It is well-known that the equality $x_L = \prod_{i=1}^{d} x_i$ is ensured by inequalities (F) and the integrality of the variables. We thus obtain (P_L), a linear program equivalent to (P_0):

$$(P_L) \begin{cases} \min m(x) = \alpha x_L \\ \text{s.t.} \\ \quad x_L \leq x_i, \quad i = 1,..,d & (3.1) \\ \quad x_L \geq \displaystyle\sum_{i=1}^{d} x_i - d + 1 & (3.2) \\ \quad x_L \geq 0 & (3.3) \\ \quad x \in \{0,1\}^{d+1} \end{cases}$$

Let us remark that it is not necessary to enforce x_L to be binary as it is ensured by constraints

(3.1) - (3.3), (P_L) is thus a mixed-integer linear program.

As for any degree d, a monomial can be linearized by a single additional variable as it is stated in (P_L). It can also be decomposed as a product of two monomials of degrees k and l such that $k + l = d$. Each monomial is then linearized by an additional variable. This decomposition can then be done recursively on the two resulting monomials. We call J the set of indices of the additional variables introduced for the linearization. Each different linearization leads to a different set J and a different linear reformulation of (P_0). Considering all the possible ways to linearize such a monomial, we now consider a linearization parameterized by q, the number of additional variables it involves. We denote this parameterized linearization *q-linearization* and in the following, we prove that for any positive integer q, the continuous relaxation bound obtained by a $q-$linearization has the same value.

We first define two sets that will give a characterization of any linearization. The set $\mathcal{E}$ contains indices of the initial variables only. For an additional variable $x_i \in J$, $\mathcal{E}$ gives the indices of the underlying initial variables whose product is equal to x_i. The formal construction of this set is given in the following definition.

Definition 3.1.1. *For all $i \in I \cup J$, we define $\mathcal{E}_i$ as the set of indices of the variables whose product is equal to x_i:*

- *If $i \in I$, i.e. x_i is an initial variable, then we set $\mathcal{E}_i = \{i\}$*
- *If $i \in J$, i.e. x_i is an additional variable, then there exist $(i_1, ..., i_k) \in (I \cup J)^k$ such that x_i replaces $\prod_{j=1}^{k} x_{i_j}$ and we set $\mathcal{E}_i = \bigcup_{j=1}^{k} \mathcal{E}_{i_j}$*

□

Next, we define the set $\mathcal{S}$ that contains the variables forming a direct decomposition of an additional variable x_i.

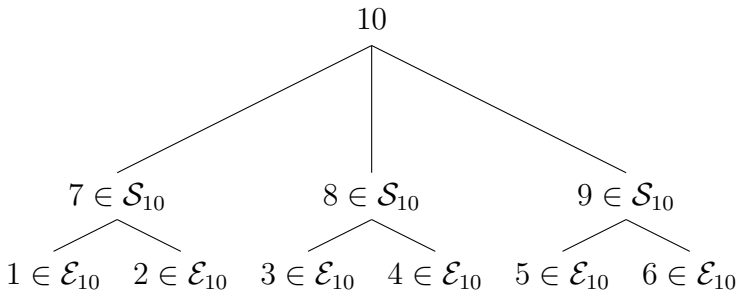
Definition 3.1.2. *For all $i \in I \cup J$, we define $\mathcal{S}_i$ as the set of indices of the variables whose product is equal to x_i:*

- *If $i \in I$, i.e. x_i is an initial variable, then we set $\mathcal{S}_i = \{i\}$*
- *If $i \in J$, i.e. x_i is an additional variable, then there exist $(i_1, ..., i_k) \in (I \cup J)^k$ such that x_i replaces $\prod_{j=1}^{k} x_{i_j}$ and we set $\mathcal{S}_i = \bigcup_{j=1}^{k} \{i_j\}$*

□

The difference between sets $\mathcal{E}$ and $\mathcal{S}$ is rather difficult to picture. Example 3 explicits the description of these sets and illustrates the difference.

Example 3. *Let $x_1x_2x_3x_4x_5x_6$ be a monomial. We introduce 4 new variables x_7, x_8, x_9, x_{10} which represent the product x_1x_2, x_3x_4, x_5x_6 and $x_7x_8x_9$ respectively. The hierarchy of the variables and the sets $\mathcal{E}_{10}$ and $\mathcal{S}_{10}$ are represented in the following tree.*



We have $\mathcal{S}_{10} = \{7, 8, 9\}$ whereas $\mathcal{E}_{10} = \{1, 2, 3, 4, 5, 6\}$. Concerning the additional variables x_7, x_8, x_9 we have $\mathcal{S}_7 = \mathcal{E}_7 = \{1, 2\}$, $\mathcal{S}_8 = \mathcal{E}_8 = \{3, 4\}$ and $\mathcal{S}_9 = \mathcal{E}_9 = \{5, 6\}$.

We can see in Example 3 that $\mathcal{E}$ represents the underlying initial variables involved in a product whereas $\mathcal{S}$ represents the variables (either initial or additional) directly involved in the product that the additional variables models.

With these new sets, we can now properly define a $q-$linearization, that is a linearization parameterized by the number of additional variables added to the equivalent problem.

Definition 3.1.3. *Let q be a positive integer, we call **q-linearization** of the monomial $m(x) = \alpha \prod_{i=1}^{d} x_i$, a Fortet's linearization of m using exactly q additional variables. The following program (P_q) coming from such a $q-$linearization is equivalent to (P_0) in the $\{0, 1\}$ space:*

$$
(P_q) \begin{cases}
\min m(x) = \alpha x_{d+q} \\
s.t. \\
\quad x_{d+k} \leq x_i, \quad \forall i \in \mathcal{S}_{d+k},\ \forall k \in \{1, .., q\} & (3.4) \\
\quad x_{d+k} \geq \sum_{i \in \mathcal{S}_{d+k}} x_i - |\mathcal{S}_{d+k}| + 1, \quad \forall k \in \{1, .., q\} & (3.5) \\
\quad x_{d+k} \geq 0, \quad \forall k \in \{1, .., q\} & (3.6) \\
\quad x \in \{0, 1\}^d
\end{cases}
$$

The special case of (P_L) corresponding to $q = 1$ is called *standard linearization.* With the definition of a $q-$linearization, we can now consider several ways to linearize a monomial and each linearization can lead to a different continuous relaxation bound. The following lemma states that, in the case of a single monomial, every $q-$linearization has the same continuous relaxation bound. We show this result by comparing the bounds obtained by every $q-$linearization to the one obtained by the standard linearization.

Lemma 3.1.1. *For any $1 \leq q \leq n$, the continuous relaxation bound value of (P_q) (denoted as $v(\bar{P}_q)$) is equal to the continuous relaxation bound value of (P_L) (denoted as $v(\bar{P}_L)$).*

Proof.

1. We first show that $v(\bar{P}_L) \leq v(\bar{P}_q)$. Let $(\bar{x}_1, .., \bar{x}_n, \bar{x}_{d+1}, .., \bar{x}_{d+q})$ be a feasible solution of $(\bar{P}_q)$.

   (a) Building a new feasible solution: we build $(x_1, .., x_d, x_L)$ a feasible solution of $(\bar{P}_L)$. We set $x_i = \bar{x}_i$ for all $i = 1, .., d$ and $x_L = \bar{x}_{d+q}$. We now prove that the constraints of $(\bar{P}_L)$ are satisfied.

      i. Constraint (3.1): by transitivity of (3.4), we have $\forall i = 1, .., n,\ x_L \leq x_i$.

      ii. Constraint (3.2): by transitivity of (3.5) and as $\sum_{k=1}^{q} |\mathcal{S}_{d+k}| = d + q - 1$, we have $x_L \geq \sum_{i=1}^{d} x_i - d + 1$.

      iii. Constraint (3.3): by (3.6), we have $x_L \geq 0$.

   (b) Comparison of the value of the objective functions: trivially, as $x_L = \bar{x}_{d+q}$, the equality holds and thus $v(\bar{P}_L) \leq v(\bar{P}_q)$.

2. We now show that $v(\bar{P}_L) \geq v(\bar{P}_q)$. Let $(\bar{x}_1, .., \bar{x}_d, \bar{x}_L)$ be an optimal solution of $(\bar{P}_L)$.

   (a) Building a new feasible solution: We consider two different cases

   - If $\alpha \leq 0$, we have $\bar{x}_L = \min(\bar{x}_i)$ and we set

$$\begin{cases} x_i = \bar{x}_i, \quad \forall i \in \{1, .., d\} \\ x_{d+k} = \min\limits_{j \in \mathcal{S}_{d+k}} (\bar{x}_j), \quad \forall k \in \{1, .., q-1\} \\ x_{d+q} = \bar{x}_L \end{cases}$$

     The solution $(\bar{x}_1, .., \bar{x}_d, x_{d+1}, .., x_{d+q-1}, \bar{x}_L)$ satisfies Constraints (3.4) and (3.6) by construction.
     For constraints (3.5), let $k \in J$ and let $j_0 = \text{argmin}_{j \in \mathcal{S}_{d+k}} \{\bar{x}_j\}$. We have $x_{d+k} = \bar{x}_{j_0}$. It follows that $\bar{x}_{j_0} - \sum_{j \in \mathcal{S}_{d+k}} x_j + |\mathcal{S}_{d+k}| - 1 = -\sum_{j \in \mathcal{S}_{d+k} \setminus \{j_0\}} x_j + |\mathcal{S}_{d+k}| - 1 = \sum_{j \in \mathcal{S}_{d+k} \setminus \{j_0\}} (1 - x_j) \geq 0$.

   - If $\alpha \geq 0$, we have $\bar{x}_L = \max(0, \sum\limits_{i=1}^{d} \bar{x}_i - d + 1)$ and we set

$$\begin{cases} x_i = \bar{x}_i, \quad \forall i \in \{1, .., d\} \\ x_{d+k} = \max(0, \sum\limits_{i \in \mathcal{S}_{d+k}} x_i - |\mathcal{S}_{d+k}| + 1), \quad \forall k \in \{1, .., q-1\} \\ x_{d+q} = \bar{x}_L \end{cases}$$

     It is easy to see that the solution $(\bar{x}_1, .., \bar{x}_d, x_{d+1}, .., x_{d+q-1}, \bar{x}_L)$ satisfies constraints (3.5) and (3.6) of $(\bar{P}_q)$ by construction. We show that constraints (3.4) are also satisfied. Let $k \in J$, if $x_{d+k} = 0$, then (3.4) is satisfied. Else, $x_{d+k} = \sum\limits_{i=1}^{d} \bar{x}_i - d + 1$. Let $j \in I$, $\sum_{i \in I \setminus \{j\}} (\bar{x}_i - 1) \leq 0$, by adding x_j to both sides we obtain $\sum_{i \in I} (\bar{x}_i - 1) + 1 \leq x_j$ which is exactly $x_{d+k} \leq x_j$.

(b) Comparison of the value of the objective functions: here again the equality trivially holds and thus $v(\bar{P}_L) \geq v(\bar{P}_q)$.

We have proven that from an optimal solution of $(\bar{P}_q)$, we can build a feasible solution of $(\bar{P}_L)$ with the same objective value and vice versa. We thus have $v(\bar{P}_L) = v(\bar{P}_q)$. □

This shows that when linearizing a single monomial, the number of additional variables and the order of product decompositions do not influence the bound obtained by continuous relaxation. Moreover, the link we had made between $q-$linearizations and the standard linearization allows the $q-$linearization to benefit from the nice polyhedral properties of the standard linearization of a single monomial. Especially, it is well-known (see for example [7]) that in the case of a single binary monomial, the standard linearization yields to the integer polytope. As a corollary, any $q-$linearization set of constraints also describes the integer polytope. In the next paragraph, we show that this result can be generalized, with some assumptions, to the linearization of a whole polynomial.

3.1.2 Linearization of a polynomial

We have seen previously that different $q-$linearizations of a single monomial lead to the same continuous relaxation bound. One simple idea to generalize this result to a whole polynomial is to consider each monomial of this polynomial and linearize them separately. In other words, if we apply a $q-$linearization to each monomial of a polynomial, then by Lemma 3.1.1, the continuous relaxation bound obtained by any $q-$linearization would be the same. This result is valid as long as an additional variable is used exclusively in one monomial. In this case, Lemma 3.1.1 holds for each monomial as it is stated in the following corollary.

Corollary 3.1.1. *For $q \geq 1$, all $q-$linearizations of a polynomial where monomials are linearized independently lead to the same bound by continuous relaxation.*

Proof. When using an additional variable exclusively in one monomial, we can apply the previous proof on each monomial separately. □

In light of these results, we remark that reusing additional variables in several monomials may lead to different bounds. For now, we do not know the exact impact of re-usability of additional variables on continuous relaxation bound. Indeed, when considering re-usability, we can not build a partially ordered set of the continuous relaxation bounds obtained by every $q-$linearization. In other words, it is difficult to compare the bound obtained by two $q-$linearizations and this remains an open question. However, in some other linearizations frameworks (LP hierarchy, valid inequalities, etc.) it is possible to compute the best linearization, see for example [67]. In the following example, we present a small example to illustrate Corollary 3.1.1 and show how re-usability can impact the bound.

Example 2 (continued). *We recall program* (Ex)*:*

$$(Ex)\left\{ \min_{x\in\{0,1\}^4} 2x_1 + 3x_2x_3 - 2x_2x_3x_4 - 3x_1x_2x_3x_4 \right.$$

We consider three different $q-$*linearizations:*

- $l_1(x) = 2x_1 + 3\underbrace{x_2x_3}_{x_5} - 2\underbrace{x_2x_3x_4}_{x_6} - 3\underbrace{x_1x_2x_3x_4}_{x_7}$
- $l_2(x) = 2x_1 + 3\underbrace{x_2x_3}_{x_5} - 2\underbrace{\underbrace{x_2x_3}_{x_6}x_4}_{x_8} - 3\underbrace{\underbrace{x_1x_2x_3}_{x_7}x_4}_{x_9}$
- $l_3(x) = 2x_1 + 3\underbrace{x_2x_3}_{x_5} - 2\underbrace{\underbrace{x_2x_3}_{x_5}x_4}_{x_7} - 3\underbrace{\underbrace{x_1x_2x_3}_{x_6}x_4}_{x_8}$

Observe that l_1 *represents the standard linearization. By Corollary 3.1.1, we know that linearizations* l_1 *and* l_2 *have the same continuous relaxation bounds, that is* -1.5*. However, we can not say anything a priori concerning the continuous relaxation bound of* l_3 *since it uses the additional variable* x_5 *twice. For this linearization, the bound value is* -0.5 *which is tighter than the other linearizations. Intuitively, if we add the equality* $x_5 = x_6$ *in* l_2*, then we get* l_3*. In this case we can justify that reusing variables can improve the bound as it amounts to adding a valid equality to* l_2*.*

These valid equalities are closely related to the $2-$*links cut presented in [24]. More precisely, when writing the* $2-$*link cuts for* $S = \{2, 3\}$ *and* $T = \{2, 3, 4\}$*, we obtain exactly* $x_5 = x_6$*. In terms of polyhedral characterization, the projection of the polyhedron corresponding to* l_3 *on the space spanned by* $(x_1, x_2, x_3, x_4, x_5, x_7, x_8)$ *is exactly the same as the 2-link polyhedra for the product* x_2x_3 *and* $x_2x_3x_4$*.*

Finally, it seems that the number of additional variables added does not affect the quality of the bound, since l_3 *has less variables than* l_1 *but more than* l_2*.*

Degree	*Method*	*Continuous relaxation bound value*
1	`Standard linearization`	-1.5

The proven property of equivalence between the linearizations is a nice and important result when considering linearization of a polynomial program from degree d to degree 1. We have shown that linearizing each monomial of a polynomial separately leads to the same continuous relaxation bound value for any $q-$linearization.

3.2 Direct convexification

The previous method for solving (P) was based on a first phase of reformulation to reduce the degree of the objective function. More precisely, we projected (P) in a higher dimensional space by adding additional variables. The resulting program was linear and thus convex. Here we focus on finding another convex reformulation of (P) with the same degree and the same number of variables than (P). For this, we follow the ideas of the $\alpha-$branch-and-bound algorithm [2]. We first introduce an approximation of the non-constant Hessian matrix of f with an *interval matrix* and then we use some eigenvalues results on these matrices. We obtain an equivalent program to (P) with a convex continuous relaxation. This relaxation is then used at each node of a branch-and-bound algorithm.

3.2.1 Approximation of the Hessian matrix

In this section, we present an algorithm to compute an equivalent convex reformulation to any binary polynomial. The goal here is to make f convex which is equivalent to its Hessian matrix H being positive semi-definite. For this reason, one solution is to perturb H so that its smallest eigenvalue becomes positive. The difficulty here is that, as soon as the degree of the polynomial f is strictly greater than 2, the Hessian matrix $H(x)$ is no longer constant and is a function of x. Thus finding the smallest eigenvalue of such a matrix is NP-hard as it amounts to find the minimum and maximum of each entry of H. The first step of our approach is to build a constant matrix that gathers all the information of the non-constant matrix $H(x)$. For this we use the concept of *interval matrix*.

Definition 3.2.1. *An interval matrix $[H]$ is a matrix whose entries are real intervals of the form $[h]_{ij} = [\underline{h}_{ij}, \overline{h}_{ij}]$.*

The purpose of the interval matrix is to find such a matrix $[H]$ so that, for each entry h_{ij} of $H(x)$ we have $h_{ij} \in [h]_{ij}$. Since the variables are binary (and hence bounded), it is possible to compute bounds for each entry of $H(x)$. Note that we only approximate the bounds of the interval matrix. Indeed, computing the optimal interval matrix of the Hessian matrix of (P) is difficult starting from degree 4, i.e. the computation of the maximum and minimum for any entry of the Hessian. To do this, it would be necessary to find the global optimums of a 0-1 function for each term of the interval matrix, and thus solve approximately n^2 integer programs. As a consequence, rather than computing an optimal interval matrix $[H^*]$, we will compute an approximate interval matrix $[H]$ (i.e. such that for all entry (i, j), $[H^*]_{ij} \subset [H]_{ij}$) and we present in the following a possible approximation of the bounds of the Hessian matrix. More precisely, we compute upper and lower bounds for each term of the Hessian matrix, $\frac{\partial^2 f}{\partial x_i x_j}$.

Proposition 3.2.1. *Let c_m be the coefficient of monomial m in f. The following inequalities hold*

$$\underline{c}_{ij} = \sum_{\substack{\text{monomial } m:\\ x_i \wedge x_j \in m\\ \text{and } c_m < 0}} c_m \leq \frac{\partial^2 f}{\partial x_i x_j} \leq \sum_{\substack{\text{monomial } m:\\ x_i \wedge x_j \in m\\ \text{and } c_m > 0}} c_m = \overline{c}_{ij}$$

Proof. We consider a monomial $m = c \prod_{k=1}^{n} x_k$. If $x_i \notin$ monomial m, or $x_j \notin$ monomial m, then $\frac{\partial^2 m}{\partial x_i x_j} = 0$. Else, $\frac{\partial^2 m}{\partial x_i x_j} = c \prod_{\substack{k \neq i \\ k \neq j}} x_k$. In this case, the variables are contained in $[0, 1]$, and we have $0 \leq \frac{\partial^2 m}{\partial x_i x_j} \leq c$ if $c > 0$, or $c \leq \frac{\partial^2 m}{\partial x_i x_j} \leq 0$ else. By summing up on the monomials we have

$$\sum_{\substack{\text{monomial m:}\\ x_i \wedge x_j \in m\\ \text{and } c_m < 0}} c_m \leq \frac{\partial^2 f}{\partial x_i x_j} \leq \sum_{\substack{\text{monomial m:}\\ x_i \wedge x_j \in m\\ \text{and } c_m > 0}} c_m$$

□

Thanks to this result, we can build an interval matrix that approximates the Hessian matrix of f. For every entry (i, j), we have $[H]_{ij} = [\underline{c}_{ij}, \overline{c}_{ij}]$ and $h_{ij} \in [H]_{ij}$.

Example 2 (continued). *The Hessian matrix of f is written as follows*

$$H_f(x) = \begin{pmatrix} 0 & -3x_3x_4 & -3x_2x_4 & -3x_2x_3 \\ -3x_3x_4 & 0 & 3 - 3x_1x_4 - 2x_4 & -2x_3 - 3x_1x_3 \\ -3x_2x_4 & 3 - 3x_1x_4 - 2x_4 & 0 & -2x_2 - 3x_1x_2 \\ -3x_2x_3 & -2x_3 - 3x_1x_3 & -2x_2 - 3x_1x_2 & 0 \end{pmatrix}$$

Its associated interval matrix $[H]$ writes

$$M = \begin{pmatrix} [0,0] & [-3,0] & [-3,0] & [-3,0] \\ [-3,0] & [0,0] & [-2,3] & [-5,0] \\ [-3,0] & [-2,3] & [0,0] & [-5,0] \\ [-3,0] & [-5,0] & [-5,0] & [0,0] \end{pmatrix}$$

Note that the diagonal terms of the Hessian matrix as well as the interval matrix are always null as we consider binary variables. Indeed the binary identity $x_i^2 = x_i$ implies that $\frac{\partial^2 f}{\partial x_i^2} = 0$.

Finally, $[H]$ positive semi-definite implies $H(x)$ positive semi-definite for all $x \in [0, 1]^n$ as values of $H(x)$ are contained in the interval matrix $[H]$. In the next section we modify selected entries of $[H]$ to find a convex reformulation of (P).

3.2.2 Convexification by uniform diagonal perturbation

In the last section we built a constant interval matrix that approximates the Hessian matrix of f. Here we present some theoretical results used in the $\alpha-$branch-and-bound algorithm [2]

concerning the eigenvalues of interval matrices. These results will lead to a convex reformulation by making the interval matrix $[H]$ positive semi-definite. We first precise the notion of positive semi-definiteness for an interval matrix.

Definition 3.2.2. *An interval matrix $[H]$ is positive semi-definite if every real matrix contained in $[H]$ is positive semi-definite.*

We then write an equivalent program to (P). Observe that the equality $x_i^2 - x_i = 0$ is valid for all $x_i \in \{0,1\}^n$. Thus the function $f_{\bar{\lambda}} : x \mapsto f(x) - \bar{\lambda}\sum_i (x_i^2 - x_i)$ parameterized by $\bar{\lambda}$ is equal to f on $\{0,1\}^n$. We then consider the following equivalent program to (P).

$$(P_{\bar{\lambda}}) \begin{cases} \min f_{\bar{\lambda}}(x) = f(x) - \bar{\lambda}\sum_i (x_i^2 - x_i) \\ \text{s.t.} \\ \quad x \in \{0,1\}^n \end{cases}$$

We denote by $[H_{\bar{\lambda}}]$ the corresponding interval matrix of $f_{\bar{\lambda}}$ which is equal to $[H] + diag(\bar{\lambda})$. The parameter $\bar{\lambda}$ can be thought as a lever to the convexity of f. Indeed, for large positive values of $\bar{\lambda}$, we have $[H_{\bar{\lambda}}] \succeq 0$ and thus $f_{\bar{\lambda}}$ is convex. However, recall that the resulting continuous relaxation will be solved through a branch-and-bound algorithm and one has to provide tight continuous relaxations at the root node of the branch-and-bound in order to improve its efficiency. The difficulty lies in finding $\bar{\lambda}$ such that $f_{\bar{\lambda}}$ is convex and at the same time the continuous relaxation bound value should be as tight as possible. The Gerschgorin Theorem [32] gives a possible value for $\bar{\lambda}$.

Theorem 3.2.1 (Scaled Gerschgorin)**.** *[32] The smallest eigenvalue λ_{min} of $[H]$ is bounded by*

$$\lambda_{min} \geq \bar{\lambda} = \min_i \left[-\sum_{j \neq i} \max(|\underline{c}_{ij}|, |\overline{c}_{ij}|) \right]$$

The convexification parameter $\bar{\lambda}$ is based on two successive approximations. Indeed, a first approximation is made on the Hessian matrix H which leads to the construction of the interval matrix $[H]$. Then a second approximation is made on the smallest eigenvalue of $[H]$. The resulting $\bar{\lambda}$ suffices to make $f_{\bar{\lambda}}$ convex given its associated interval matrix. The more accurate the interval matrix is, the tighter the continuous relaxation will be. This leads to a trade-off between the tightness of $[H]$ and the computational speed.

We illustrate the approach presented in this section in a small example.

Example 2 (continued). *After applying Theorem 3.2.1 to our example, we obtain $\bar{\lambda} = -13$. Thus the equivalent program $(P_{\bar{\lambda}})$ is*

$$(P_{\bar{\lambda}})\begin{cases} \min f_{\bar{\lambda}}(x) = 2x_1 + 3x_2x_3 - 2x_2x_3x_4 - 3x_1x_2x_3x_4 + 6.5\sum_i (x_i^2 - x_i) \\ s.t. \\ \quad x \in \{0,1\}^4 \end{cases}$$

Finally the Hessian of $f_{\bar{\lambda}}$ writes

$$H_{f_{\bar{\lambda}}}(x) = \begin{pmatrix} 13 & -3x_3x_4 & -3x_2x_4 & -3x_2x_3 \\ -3x_3x_4 & 13 & 3 - 3x_1x_4 - 2x_4 & -2x_3 - 3x_1x_3 \\ -3x_2x_4 & 3 - 3x_1x_4 - 2x_4 & 13 & -2x_2 - 3x_1x_2 \\ -3x_2x_3 & -2x_3 - 3x_1x_3 & -2x_2 - 3x_1x_2 & 13 \end{pmatrix}$$

Degree	*Method*	*Continuous relaxation bound value*
1	`Standard linearization`	-1.5
d	`Uniform interval`	-5.3

3.2.3 Convexification by non-uniform diagonal perturbation

In the previous convexification, we perturbed the diagonal of the interval matrix $[H]$ by adding a constant term on the diagonal. This parameter $\bar{\lambda}$ approximates the smallest eigenvalue of the interval matrix. In this section, we will refine our convexification by perturbing the diagonal of $[H]$ with a vectorial parameter $\bar{\sigma}$. The corresponding equivalent problem $(P_{\bar{\sigma}})$ is

$$(P_{\bar{\sigma}})\begin{cases} \min f_{\bar{\sigma}}(x) = f(x) - \sum_i \bar{\sigma}_i(x_i^2 - x_i) \\ \text{s.t.} \\ \quad x \in \{0,1\}^n \end{cases}$$

The new interval matrix $[H_\sigma]$ satisfies $[H_\sigma] = [H] + diag(\sigma)$. Similarly to the previous section, it is possible to choose specific values for $\bar{\sigma}$ such that $[H_\sigma] \succeq 0$ and thus $f_{\bar{\sigma}}$ is convex. The following Theorem gives an example of such a vector $\bar{\sigma}$.

Theorem 3.2.2 (Scaled Gerschgorin)**.** *Let $\bar{\sigma}$ be the $n-$dimensional vector defined by*

$$\bar{\sigma}_i = \min\left[0, \left(\sum_{j\neq i} - \max\left(\left|\underline{c}_{ij}\right|, \left|\bar{c}_{ij}\right|\right)\right)\right] \quad \forall i \in \{1, ..., n\} \tag{3.7}$$

Then $H - diag(\bar{\sigma})$ is positive semi-definite.

Note that this convexification is better than the previous one. Indeed, $\min_i(\bar{\sigma}_i) = \bar{\lambda}$ and thus for all $i \in \{1, ..., \}$, $\bar{\sigma}_i \geq \bar{\lambda}$. A more refined convexification is obtained, in the sense that the value of the continuous relaxation bound of $(P_{\bar{\sigma}})$ is greater or equal than the one of $(P_{\bar{\lambda}})$. We illustrate the comparison between the two programs in the following example.

Example 2 (continued). *After applying Theorem 3.7 to our example, we obtain* $\bar{\sigma} = (-9, -11, -11, -13)$. *Thus the equivalent program* $(P_{\bar{\sigma}})$ *is*

$$(P_{\bar{\sigma}}) \begin{cases} \min f_{\bar{\sigma}}(x) = 2x_1 + 3x_2x_3 - 2x_2x_3x_4 - 3x_1x_2x_3x_4 \\ +4.5(x_1^2 - x_1) + 5.5(x_2^2 - x_2) + 5.5(x_3^2 - x_3) + 6.5(x_4^2 - x_4) \\ s.t. \\ \quad x \in \{0,1\}^4 \end{cases}$$

Finally the Hessian of $f_{\bar{\sigma}}$ *writes*

$$H_{f_{\bar{\sigma}}}(x) = \begin{pmatrix} 9 & -3x_3x_4 & -3x_2x_4 & -3x_2x_3 \\ -3x_3x_4 & 11 & 3 - 3x_1x_4 - 2x_4 & -2x_3 - 3x_1x_3 \\ -3x_2x_4 & 3 - 3x_1x_4 - 2x_4 & 11 & -2x_2 - 3x_1x_2 \\ -3x_2x_3 & -2x_3 - 3x_1x_3 & -2x_2 - 3x_1x_2 & 13 \end{pmatrix}$$

Degree	*Method*	*Continuous relaxation bound value*
1	`Standard linearization`	-1.5
d	`Uniform interval`	-5.3
d	`Non-uniform interval`	-4.3

3.3 Conclusion

In this Chapter we have presented two specific cases of convex reformulation of (P). In Section 3.1, we introduced the concept of $q-$linearizations and proved that the continuous relaxation bounds obtained by every $q-$linearization are the same when using an additional variable only once. Especially, the standard linearization has the same continuous relaxation bound value than any other $q-$linearization in this case. In section 3.2, we approximated the non-constant Hessian matrix $H(x)$ of f with a constant interval matrix $[H]$. The Gerschgorin Theorems 3.2.1 and 3.2.2 provide feasible values for diagonal perturbation of $[H]$. The first Theorem leads to a uniform perturbation of the diagonal of H, whereas the second lead to a non-uniform perturbation with a slightly better bound. However, the continuous relaxation bound values obtained by this method

are not enough tight as they are far from the optimal value. For this reason, in the next Chapter, we present a general framework to solve (P) with tight continuous relaxation bounds. It consists in a double reformulation phase before applying a branch-and-bound algorithm.

Chapter 4

PQCR: Polynomial optimization in binary variables through Quadratic Convex Reformulation

Contents

4.1 A quadratic convex reformulation framework 62

4.1.1 Phase 1: Quadratization of polynomial program (P) into (QP) 63

4.1.2 Phase 2: Optimal quadratic convex reformulation of (QP) 65

4.2 Discussion on the impact of the chosen 2×2 quadratization 76

4.2.1 2×2 Full quadratization 82

4.2.2 2×2 Partial quadratization 84

4.2.3 Replacing PQCR's quadratization scheme with other pairwise covers . . . 86

4.3 Link with the Lasserre's hierarchy 88

4.3.1 A hierarchy of exact quadratic convex reformulations 88

4.4 Conclusion 91

In this chapter, we are interested in building an equivalent program to (P), defined in 2.1, using two successive reformulations, a first quadratic reformulation and then a convex reformulation. We then solve the resulting quadratic convex program by a branch-and-bound algorithm. We introduce our main algorithm to solve (P), called PQCR (Polynomial optimization in binary variables through Quadratic Convex Reformulation).

The outline of the section is the following. In Section 4.1, we define and present our algorithm PQCR. We define the quadratic reformulation and the convex reformulation. We then compare the results obtained by PQCR with the ones obtained by state-of-the-art solvers. In Section 4.2, we

focus on the quadratization phase. Indeed, PQCR is built on an optimal convex reformulation but this reformulation highly depends on the quadratization phase. This is why, this section proposes to compare some quadratizations in terms of continuous relaxation bound values. In Section 4.3, we theoretically compare our approach to the Moment/SOS method. Finally, Section 4.4 draws a conclusion on quadratic convex reformulations applied to (P).

4.1 A quadratic convex reformulation framework

Quadratic convex reformulation methods [10, 11] were introduced for the specific case where $d = 2$. The idea of these approaches is to build tight equivalent reformulations to (P) that have a convex objective function. This equivalent problem can be built using the dual solution of a semi-definite relaxation of (P), and further solved by a branch-and-bound algorithm based on quadratic convex relaxation. Here, we consider the more general case where $d \geq 3$, and we propose to compute an equivalent convex formulation to (P). Hence, we present an exact solution method for problem (P) that can be split in three phases. The first phase consists in building an equivalent formulation to (P) where both objective function and constraints are at most quadratic. For this, we need to add some auxiliary variables. We then obtain problem (QP) that has a quadratic objective function and linear inequalities.

Then in the second phase, we focus on the convexification of the obtained problem. As illustrated in the experiments in Section 7.3.1, the original QCR and MIQCR methods are not able to handle (QP). Indeed, QCR leads to a reformulation with a weak bound, and in method MIQCR the semi-definite program that we need to solve is too large. This is why we introduce a tailored convexification phase. The idea is to apply convex quadratic reformulation to any quadratization of (P). For this, we need null quadratic functions on the domain of (QP) so as to perturb the Hessian matrix of the new quadratic objective function. One of these null functions comes from the classic binary identity, $x_i^2 = x_i$. One contribution of this section is the introduction of new null quadratic functions on the domain of (QP). This set of functions varies according to the quadratization used in phase 1. Adding these functions to the new objective function, we get a family of convex equivalent formulations to (QP) that depend on some parameters. We then want to choose these parameters such that the continuous relaxation bound of the convexified problem is maximized. We show that they can be computed thanks to a semi-definite program. Finally, the last phase consists in solving the convexified problem using general-purpose optimization softwares.

Our experiments show that PQCR is able to solve to global optimality 6 unsolved instances of the low auto-correlation binary sequence problem and improves lower and/or upper bounds of 27 of the 45 instances available at the minlplib website, in comparison to the other available solvers.

4.1.1 Phase 1: Quadratization of polynomial program (P) into (QP)

We present how we build equivalent quadratic formulations to (P). The basic idea is to reduce the degree of f to 2. For this, in each monomial of degree 3 or greater, we simply recursively replace each product of two variables by an additional variable.

More formally, we define the set of indices of the additional variables $J = \{n+1, .., N\}$, where N is the total number of initial and additional variables. We also refer to the subsets $\mathcal{E}_i$ defined in Definition 3.1.1 for the initial or additional variable i. Using these sets, we define a *valid* quadratization as a reformulation with N variables where any monomial of degree at least 3 is replaced by the product of two variables.

Definition 4.1.1. *The sets $J = \{n+1, .., N\}$ and $\{\mathcal{E}_i,\ i \in I \cup J\}$ define a valid quadratization with N variables if, for any monomial p of degree greater than or equal to 3 (i.e. $|\mathcal{M}_p| \geq 3$), there exist $(j,k) \in (I \cup J)^2$ such that $\mathcal{M}_p = \mathcal{E}_j \cup \mathcal{E}_k$ and then $\prod_{i \in \mathcal{M}_p} x_i = x_j x_k$. Then the monomial p is replaced by a quadratic term.*

□

With this definition of a quadratization, we reformulate (P) as a non-convex quadratically constrained quadratic program $(QCQP)$ with N variables.

$$(QCQP) \begin{cases} \min g(x) = \sum\limits_{\substack{|\mathcal{M}_p| \geq 3 \\ \mathcal{M}_p = \mathcal{E}_j \cup \mathcal{E}_k}} c_p x_j x_k + \sum\limits_{|\mathcal{M}_p| \leq 2} c_p \prod\limits_{i \in \mathcal{M}_p} x_i \\ \text{s.t.} \\ \quad x_i = x_{i_1} x_{i_2} \quad \forall (i, i_1, i_2) \in J \times (I \cup J)^2 : \ \mathcal{E}_i = \mathcal{E}_{i_1} \cup \mathcal{E}_{i_2} \\ \quad x \in \{0,1\}^N \end{cases} \tag{4.1}$$

Note that the reformulation of f into a quadratic function is not unique and, given a polynomial f and a set $\mathcal{E}$, there exist several corresponding quadratic functions g. This is not impacting our study as our approach applies to any quadratization. Further discussion on this issue are provided in Section 4.2.

As the variables are binary, Constraints (6.1) are equivalent to the classical set of Fortet inequalities [28]:

$$(C^i_{i_1,i_2}) \begin{cases} x_i - x_{i_1} \leq 0, \\ x_i - x_{i_2} \leq 0, \\ -x_i + x_{i_1} + x_{i_2} \leq 1, \\ -x_i \leq 0, \end{cases}$$

We now define set $\mathcal{F}_{\mathcal{E}}$:

$$\mathcal{F}_{\mathcal{E}} = \{x \in \{0,1\}^N : C^i_{i_1,i_2} \text{ is satisfied } \forall (i,i_1,i_2) \in J \times (I \cup J)^2 : \ \mathcal{E}_i = \mathcal{E}_{i_1} \cup \mathcal{E}_{i_2}\}.$$

We denote by $M = 4(N-n)$ the number of constraints of $\mathcal{F}_{\mathcal{E}}$. We thus obtain the following linearly constrained quadratic formulation that is equivalent to (P) and has N variables and M constraints:

$$(QP) \begin{cases} \min g(x) \equiv x^T Q x + c^t x \\ \text{s.t.} \\ \quad x \in \mathcal{F}_{\mathcal{E}} \end{cases}$$

where $Q \in S_N$ (the set of $N \times N$ real symmetric matrices), and $c \in \mathbb{R}^N$.

In the following, we will focus on the solution of problem (QP), that is a binary quadratic program with linear constraints. It is equivalent to (P). Let us observe that (QP), as well as $(QCQP)$, are parameterized by the quadratization defined by sets $\mathcal{E}$. Indeed, several valid quadratizations can be applied to (P), each of them leading to different sets $\mathcal{E}_i$.

Different valid quadratizations were introduced and compared from the size point of view in [5]. In our case the comparison criterion is the continuous relaxation bound value from which we present our experimental comparison.

Example 2 (continued). *[Different valid quadratizations]*
For instance, we can build three different equivalent functions to f:

- $g_1(x) = 2x_1 + 3x_2x_3 - 2\underbrace{x_2x_4}_{x_5}x_3 - 3\underbrace{x_1x_4}_{x_6}\underbrace{x_2x_3}_{x_7}$
- $g_2(x) = 2x_1 + 3x_2x_3 - 2\underbrace{x_3x_4}_{x_6}x_2 - 3\underbrace{x_1x_2}_{x_5}\underbrace{x_3x_4}_{x_6}$
- $g_3(x) = 2x_1 + 3x_2x_3 - 2\underbrace{x_2x_4}_{x_5}x_3 - 3\underbrace{x_1x_2}_{x_6}\underbrace{x_3x_4}_{x_7}$

$$(QEx_1) \begin{cases} \min g_1(x) \\ s.t. \\ \quad (x_2,x_4,x_5) \in C^5_{2,4} \\ \quad (x_1,x_4,x_6) \in C^6_{1,4} \\ \quad (x_2,x_3,x_7) \in C^7_{2,3} \\ \quad x \in \{0,1\}^7 \end{cases} \qquad (QEx_2) \begin{cases} \min g_2(x) \\ s.t. \\ \quad (x_3,x_4,x_6) \in C^6_{3,4} \\ \quad (x_1,x_2,x_5) \in C^5_{1,2} \\ \quad x \in \{0,1\}^6 \end{cases} \qquad (QEx_3) \begin{cases} \min g_3(x) \\ s.t. \\ \quad (x_2,x_4,x_5) \in C^5_{2,4} \\ \quad (x_1,x_2,x_6) \in C^6_{1,2} \\ \quad (x_3,x_4,x_7) \in C^7_{3,4} \\ \quad x \in \{0,1\}^7 \end{cases}$$

Here we obtain 3 different quadratizations of (Ex) with different sets $\mathcal{E}$. They have different sizes: (QEx_1) and (QEx_3) have 7 variables and 12 constraints, while (QEx_2) has 6 variables and 8 constraints.

We have reduced the degree of the polynomial program (P) by building an equivalent quadratic program to (P). However, the solution of (QP) still has two difficulties, the non-convexity of the objective function g and the integrality of the variables.

Some state-of-the-art solvers can solve (QP) to global optimality (e.g. `Cplex 12.7` [42]). Unfortunately, these solvers may not be enough efficient for solving dense instances of (P). Here, we propose to compute an equivalent quadratic convex formulation to (QP). There exist several convexification methods devoted to quadratic programming (see, for example [11, 13, 23, 39, 59]). These approaches can be directly applied to (QP). For instance, one can use the `QCR` method, described in [13], that consists in computing an equivalent convex formulation to (QP) using semi-definite programming. The convexification is obtained thanks to a non uniform perturbation of the diagonal of the Hessian matrix. The semi-definite relaxation used can be easily solved due to its reasonable size. However, the bound obtained by continuous relaxation of the reformulation is very weak. As a consequence, for the considered instances of Section 4, the branch-and-bound used to solve the reformulation failed as soon as $n \geq 20$. Another alternative is to apply the `MIQCR` method [11]. In this method, the perturbation is generalized to the whole Hessian matrix and hence is more refined than the previous one. This leads to a reformulation with a significantly sharper bound. Unfortunately, the semi-definite relaxation used in this approach is too large and its computation failed even with instances of (P) containing only 10 variables. In the next section, we present a new convexification that leads to sharper bounds than `QCR` but with a better tractability than `MIQCR`.

4.1.2 Phase 2: Optimal quadratic convex reformulation of (QP)

We consider the problem of reformulating (QP) by an equivalent quadratic 0-1 program with a convex objective function. To do this, we define a new convex function which value is equal to the value of $g(x)$, but which Hessian matrix is positive semi-definite. More precisely, we first add to $g(x)$ sets of functions that vanish on the feasible set $\mathcal{F}_{\mathcal{E}}$. Then we focus on computing the best parameters that lead to a convex function and that maximize the optimal value of the continuous relaxation of the obtained problem. For each function we introduce a scalar parameter. We present two state-of-the-art convexification methods and then we introduce our main convexification algorithm.

4.1.2.1 Smallest eigenvalue convexification [39]

We begin by one of the first convexification chronologically speaking. This method was introduced by Hammer and Rubin in [39] and it consists in modifying diagonal entries of the Hessian matrix

of f by adding null functions to it. More precisely, let $\lambda \in \mathbb{R}$ be a real number. We add the function $x \mapsto \lambda \sum_{i=1}^{N}(x_i^2 - x_i)$ which is null over $\{0,1\}^N$. We thus get the following equivalent program to (QP) and parameterized by λ:

$$(QP_\lambda) \begin{cases} \min g_\alpha(x) = g(x) + \lambda \displaystyle\sum_{i=1}^{N}(x_i^2 - x_i) \\ \text{s.t.} \\ \quad x \in \mathcal{F}_\mathcal{E} \end{cases}$$

Observe that $g_\lambda(x) = g(x)$ for all $x \in \mathcal{F}_\mathcal{E}$. Hammer and Rubin [39] have proved that for any $\lambda \leq \lambda_{min}(Q)$, g_λ is convex. In other words, the Hessian matrix Q_λ of g_λ is such that $Q_\lambda = Q + diag(\lambda) \succeq 0$. Moreover, for a given vector $x \in [0,1]^N$, $\lambda \mapsto f_\lambda(x)$ is a decreasing function. Thus the optimal value of the continuous relaxation of (QP_λ) is attained when $\lambda^* = -\lambda_{min}(Q)$.

Example 2 (continued). *We focus on the quadratic reformulation (QEx_2) previously introduced:*

$$g_2(x) = 2x_1 + 3x_2x_3 - 2\underbrace{x_3x_4}_{x_6}x_2 - 3\underbrace{x_1x_2}_{x_5}\underbrace{x_3x_4}_{x_6}$$

The Hessian matrix Q of g_2 is the following

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.5 & 0 & 0 & -1 \\ 0 & 1.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1.5 \\ 0 & -1 & 0 & 0 & -1.5 & 0 \end{pmatrix}$$

For this example, the smallest eigenvalue of Q is $\lambda_{min} = 2.08$ and the equivalent program (QP_λ) is written as follows

$$(QP_\lambda) \begin{cases} \min g_\alpha(x) = 2x_1 + 3x_2x_3 - 2x_2x_6 - 3x_5x_6 + 1.04 \displaystyle\sum_{i=1}^{N}(x_i^2 - x_i) \\ s.t. \\ \quad x \in \mathcal{F}_\mathcal{E} \end{cases}$$

The Hessian matrix of g_λ after the diagonal perturbation is

$$Q_\lambda = \begin{pmatrix} 2.08 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2.08 & 1.5 & 0 & 0 & -1 \\ 0 & 1.5 & 2.08 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.08 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.08 & -1.5 \\ 0 & -1 & 0 & 0 & -1.5 & 2.08 \end{pmatrix}$$

Note that it is not necessary to modify the entries $(1,1)$ *and* $(4,4)$ *of* Q_λ *as the second order partial derivatives of* x_1 *and* x_4 *are null.*

Degree	*Method*	*Continuous relaxation bound value*
1	`Standard linearization`	-1.5
d	`Uniform interval`	-5.3
d	`Non-uniform interval`	-4.3
2	`Smallest eigenvalue`	-1.7

4.1.2.2 Non-uniform diagonal convexification: `QCR` [13]

It is possible to derive a more accurate convex reformulation than previously by taking a vectorial parameter $\alpha \in \mathbb{R}^N$. Thus the perturbation concerns every diagonal entry of the Hessian matrix independently. More precisely, we consider the following equivalent program to (P) parameterized by $\alpha \in \mathbb{R}^N$.

$$(QP_\alpha) \begin{cases} \min g_\alpha(x) = g(x) + \sum_{i=1}^{N} \alpha_i (x_i^2 - x_i) \\ \text{s.t.} \\ \quad x \in \overline{\mathcal{F}}_\mathcal{E} \\ \quad x \in \{0,1\}^N \end{cases}$$

where $\overline{\mathcal{F}}_\mathcal{E}$ is the set $\mathcal{F}_\mathcal{E}$ where the integrality constraints are relaxed, i.e. $x \in [0,1]^N$. We are interested in the computation of a vector α such that the continuous relaxation bound value of (QP_α) is maximized for g_α convex, i.e. for $Q_\alpha = Q + diag(\alpha) \succeq 0$. This amounts to solve the following program:

$$(CP_{NU}) : \max_{\substack{\alpha \in \mathbb{R}^N, \\ Q_\alpha \succeq 0}} \left\{ \min_{x \in \overline{\mathcal{F}}_\mathcal{E}} \quad g_\alpha(x) \right\}$$

Billonnet et al. proved in [13] that (CP_{NU}) is a convex problem and optimal values of α can

be extracted from a semi-definite relaxation of (QP_α).

Theorem 4.1.1 ([13]). *The optimal value of (CP_{NU}) is equal to the optimal value of the following semi-definite program (SDP_α):*

$$(SDP_\alpha)\begin{cases} \min < Q, X > + c^T x \\ s.t. \\ \quad X_{ii} - x_i = 0 \qquad\qquad i \in I \cup J \\ \quad x \in \overline{\mathcal{F}}_{\mathcal{E}} \\ \quad \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 \\ \quad x \in \mathbb{R}^N,\ X \in S^N \end{cases} \tag{4.2}$$

The optimal values of α^* of problem (CP_{NU}) are given by the optimal values of the dual variables associated with constraints (4.2) .

Example 2 (continued). *For this example, the resolution of (SDP_α) leads to $\alpha_1^* = 0$, $\alpha_2^* = 2.5$, $\alpha_3^* = 1.5$, $\alpha_4^* = 0$, $\alpha_5^* = 1.5$ and $\alpha_6^* = 2.5$. The equivalent program (QP_α) is written as follows*

$$(QP_\alpha)\begin{cases} \min g_\alpha(x) = 2x_1 + 3x_2x_3 - 2x_2x_6 - 3x_5x_6 + 1.25(x_2^2 - x_2) \\ +0.75(x_3^2 - x_3) + 0.75(x_5^2 - x_5) + 1.25(x_6^2 - x_6) \\ s.t. \\ \quad x \in \mathcal{F}_{\mathcal{E}} \end{cases}$$

The Hessian matrix of g_α after the diagonal perturbation is

$$Q_\alpha = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2.5 & 1.5 & 0 & 0 & -1 \\ 0 & 1.5 & 1.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.5 & -1.5 \\ 0 & -1 & 0 & 0 & -1.5 & 2.5 \end{pmatrix}$$

Degree	*Method*	*Continuous relaxation bound value*
1	`Standard linearization`	-1.5
d	`Uniform interval`	-5.3
d	`Non-uniform interval`	-4.3
2	`Smallest eigenvalue`	-1.7
2	`QCR`	-1.6

4.1.2.3 General convexification framework

We present the convexification algorithm that will be used within the PQCR framework. Unlike the previous convex reformulations, this one takes advantage of the quadratization phase to derive valid equalities. The null functions corresponding to these equalities are then added to g in order to compute a new convexification. For a quadratization characterized by $\mathcal{E}$, we first introduce these null quadratic functions over the set $\mathcal{F}_{\mathcal{E}}$.

Lemma 4.1.1. *The following quadratic equalities characterize null functions over $\mathcal{F}_{\mathcal{E}}$:*

$$(S_{\mathcal{E}})\begin{cases} x_i^2 - x_i = 0 & i \in I \cup J & (4.3)\\ x_i - x_i x_j = 0 & (i,j) \in J \times (I \cup J) : \ \mathcal{E}_j \subset \mathcal{E}_i & (4.4)\\ x_i - x_j x_k = 0 & (i,j,k) \in J \times (I \cup J)^2 : \ \mathcal{E}_i = \mathcal{E}_j \cup \mathcal{E}_k & (4.5)\\ x_i x_j - x_k x_l = 0 & (i,j,k,l) \in (I \cup J)^4 : \ \mathcal{E}_i \cup \mathcal{E}_j = \mathcal{E}_k \cup \mathcal{E}_l & (4.6)\end{cases}$$

Proof. Constraints (4.3) trivially hold since $x_i \in \{0,1\}$. Constraints (4.5) come from Definition 1. We then prove the validity of the Constraints (4.4) and (4.6).

- *Constraints (4.4):* we have $x_i = \prod_{i' \in \mathcal{E}_i} x_{i'}$ and $x_j = \prod_{j' \in \mathcal{E}_j} x_{j'}$, then:

$$\begin{aligned} x_i x_j &= \prod_{i' \in \mathcal{E}_i} x_{i'} \prod_{j' \in \mathcal{E}_j} x_{j'} \\ &= \prod_{j' \in \mathcal{E}_j} x_{j'}^2 \prod_{i' \in \mathcal{E}_i \backslash \mathcal{E}_j} x_{i'} \text{ since } \mathcal{E}_j \subset \mathcal{E}_i \\ &= \prod_{i' \in \mathcal{E}_i} x_{i'} \text{ since } x_{j'}^2 = x_{j'} \text{ and } \mathcal{E}_j \cup (\mathcal{E}_i \backslash \mathcal{E}_j) = \mathcal{E}_i \\ &= x_i \end{aligned}$$

- *Constraints (4.6):* by definition we have:

$$\begin{aligned} x_i x_j &= \prod_{i' \in \mathcal{E}_i} x_i' \prod_{j' \in \mathcal{E}_j} x_{j'} \\ &= \prod_{i' \in \mathcal{E}_i \cup \mathcal{E}_j} x_{i'} \prod_{j' \in \mathcal{E}_i \cap \mathcal{E}_j} x_{j'} \\ &= \prod_{i' \in (\mathcal{E}_i \cup \mathcal{E}_j) \backslash (\mathcal{E}_i \cap \mathcal{E}_j)} x_{i'} \prod_{j' \in \mathcal{E}_i \cap \mathcal{E}_j} x_{j'}^2 \\ &= \prod_{i' \in \mathcal{E}_i \cup \mathcal{E}_j} x_{i'} \text{ since } x_{j'}^2 = x_{j'} \text{ and } (\mathcal{E}_i \cup \mathcal{E}_j) \backslash (\mathcal{E}_i \cap \mathcal{E}_j) \cup (\mathcal{E}_i \cap \mathcal{E}_j) = (\mathcal{E}_i \cup \mathcal{E}_j) \\ &= \prod_{k' \in \mathcal{E}_k \cup \mathcal{E}_l} x_{k'} \text{ since } \mathcal{E}_i \cup \mathcal{E}_j = \mathcal{E}_k \cup \mathcal{E}_l \\ &= x_k x_l \end{aligned}$$

□

We now compute a quadratic convex reformulation of (QP) and thus of (P). For this, we add to the objective function g the null quadratic forms in (4.3)–(4.6). For each of them, we associate a real scalar parameter: α_i for Constraints (4.3), β_{ij} for Constraints (4.4), δ_{ijk} for Constraints (4.5), and λ_{ijkl} for Constraints (4.6). We get the following parameterized function:

$$\begin{aligned} g_{\alpha,\beta,\delta,\lambda}(x) &= g(x) + \sum_{i \in I \cup J} \alpha_i (x_i^2 - x_i) + \sum_{\substack{(i,j) \in J \times (I \cup J) \\ \mathcal{E}_j \subset \mathcal{E}_i}} \beta_{ij}(x_i - x_i x_j) \\ &\quad + \sum_{\substack{(i,j,k) \in J \times (I \cup J)^2 \\ \mathcal{E}_i = \mathcal{E}_j \cup \mathcal{E}_k}} \delta_{ijk}(x_i - x_j x_k) + \sum_{\substack{(i,j,k,l) \in (I \cup J)^4 \\ \mathcal{E}_i \cup \mathcal{E}_j = \mathcal{E}_k \cup \mathcal{E}_l}} \lambda_{ijkl}(x_i x_j - x_k x_l) \end{aligned}$$

Obviously $g_{\alpha,\beta,\delta,\lambda}(x)$ has the same value as $g(x)$ for any $x \in \mathcal{F}_{\mathcal{E}}$. Moreover, there exist vector parameters α, β, δ and λ such that $g_{\alpha,\beta,\delta,\lambda}$ is a convex function. Take for instance, α equals to the opposite of the smallest eigenvalue of Q, and $\beta = \delta = \lambda = 0$, which amounts to apply the smallest eigenvalue convexification.

By replacing g by the new function, we obtain the following family of quadratic convex equivalent formulation to (QP):

$$(QP_{\alpha,\beta,\delta,\lambda}) \begin{cases} \min g_{\alpha,\beta,\delta,\lambda}(x) \equiv x^T Q_{\alpha,\beta,\delta,\lambda} x + c^T_{\alpha,\beta,\delta,\lambda} x \\ \text{s.t.} \\ \quad x \in \mathcal{F}_{\mathcal{E}} \end{cases} \tag{4.7}$$

where $Q_{\alpha,\beta,\delta,\lambda} \in S_N$ is the Hessian matrix of $g_{\alpha,\beta,\delta,\lambda}(x)$, and $c_{\alpha,\beta,\delta,\lambda} \in \mathbb{R}^N$ is the vector of linear coefficients of $g_{\alpha,\beta,\delta,\lambda}(x)$.

In order to use $(QP_{\alpha,\beta,\delta,\lambda})$ within a branch-and-bound procedure, we are interested by parameters $(\alpha, \beta, \delta, \lambda)$ such that $g_{\alpha,\beta,\delta,\lambda}$ is a convex function. Moreover, in order to have a good behavior of the branch-and-bound algorithm, we want to find parameters that give the tightest continuous relaxation bound. More formally, we want to solve the following optimization problem:

$$(CP) : \max_{\substack{\alpha \in \mathbb{R}^N, \beta \in \mathbb{R}^{T_1}, \delta \in \mathbb{R}^{T_2}, \lambda \in \mathbb{R}^{T_3} \\ Q_{\alpha,\beta,\delta,\lambda} \succeq 0}} \left\{ \min_{x \in \overline{\mathcal{F}}_{\mathcal{E}}} \; g_{\alpha,\beta,\delta,\lambda}(x) \right\}$$

where T_1, T_2 and T_3 are the number of Constraints (4.4), (4.5), and (4.6), respectively, and $\overline{\mathcal{F}}_{\mathcal{E}}$ is the set $\mathcal{F}_{\mathcal{E}}$ where the integrality constraints are relaxed, i.e. $x \in [0,1]^N$.

In the rest of the section we will focus on solving (CP). For this, we build a compact semidefinite relaxation that uses our new valid equalities and prove that its optimal dual variables provide an optimal solution to (CP).

4.1.2.4 Computing an optimal solution to (CP)

The following theorem shows that problem (CP) is equivalent to the dual of a semi-definite relaxation of (QP).

Theorem 4.1.2. *The optimal value of (CP) is equal to the optimal value of the following semi-definite program (SDP):*

$$(SDP)\begin{cases} \min < Q, X > + c^T x \\ s.t. \\ \quad X_{ii} - x_i = 0 & i \in I \cup J & (4.8) \\ \quad -X_{ij} + x_i = 0 & (i,j) \in J \times (I \cup J) : \ \mathcal{E}_j \subset \mathcal{E}_i & (4.9) \\ \quad -X_{jk} + x_i = 0 & (i,j,k) \in J \times (I \cup J)^2 : \ \mathcal{E}_i = \mathcal{E}_j \cup \mathcal{E}_k & (4.10) \\ \quad X_{ij} - X_{kl} = 0 & (i,j,k,l) \in (I \cup J)^4 : \ \mathcal{E}_i \cup \mathcal{E}_j = \mathcal{E}_k \cup \mathcal{E}_l & (4.11) \\ \quad \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 & & (4.12) \\ \quad x \in \mathbb{R}^N, \ X \in S^N & & (4.13) \end{cases}$$

The optimal values $(\alpha^*, \beta^*, \delta^*, \lambda^*)$ of problem (CP) are given by the optimal values of the dual variables associated with constraints (4.8)–(4.11) respectively.

Proof. For simplicity, we rewrite $\mathcal{F}_\mathcal{E}$ as follows: $\mathcal{F}_\mathcal{E} = \{x \in \{0,1\}^N : Ax \leq b\}$ where A is a $M \times N$-matrix, $b \in \mathbb{R}^M$, and we introduce $T = N + T_1 + T_2 + T_3$ the number of Constraints (4.3)–(4.6) respectively.

We start by observing that $x \in [0,1]^N$ is equivalent to $x^2 \leq x$, thus, (CP) is equivalent to $(Q1)$:

$$(Q1) : \max_{\substack{\alpha \in \mathbb{R}^N, \beta \in \mathbb{R}^{T_1}, \delta \in \mathbb{R}^{T_2}, \lambda \in \mathbb{R}^{T_3} \\ Q_{\alpha,\beta,\delta,\lambda} \succeq 0}} \left\{ \min_{x \in \mathbb{R}^N, \ x^2 \leq x, \ Ax \leq b} g_{\alpha,\beta,\delta,\lambda}(x) \right\}$$

$(Q1)$ is a convex optimization problem over a convex set. If we consider the solution $\tilde{x}_i = 0.5$ $\forall i \in I$ and $\tilde{x}_i = \tilde{x}_j \tilde{x}_k$ $\forall (i,j,k) \in J \times (I \cup J)^2, \ \mathcal{E}_i = \mathcal{E}_j \cup \mathcal{E}_k$, the obtained $\tilde{x}$ is an interior point and the Slater's conditions are satisfied for the minimization sub-problem. Then, by Lagrangian duality, we have $(Q1)$ equivalent to $(Q2)$:

$$(Q2) : \max_{\substack{\alpha \in \mathbb{R}^N, \beta \in \mathbb{R}^{T_1}, \delta \in \mathbb{R}^{T_2}, \lambda \in \mathbb{R}^{T_3}, \omega \in \mathbb{R}^N_+, \gamma \in \mathbb{R}^M_+ \\ Q_{\alpha,\beta,\delta,\lambda} \succeq 0}} \left\{ \min_{x \in \mathbb{R}^N} g_{\alpha,\beta,\delta,\lambda}(x) + \omega^T (x^2 - x) + \gamma^T (Ax - b) \right\}$$

Due to Constraints (4.3), it holds that $(Q2)$ is equivalent to $(Q3)$:

$$(Q3) : \max_{\substack{\alpha \in \mathbb{R}^N, \beta \in \mathbb{R}^{T_1}, \delta \in \mathbb{R}^{T_2}, \lambda \in \mathbb{R}^{T_3}, \gamma \in \mathbb{R}^M_+ \\ Q_{\alpha,\beta,\delta,\lambda} \succeq 0}} \left\{ \min_{x \in \mathbb{R}^N} g_{\alpha,\beta,\delta,\lambda}(x) + \gamma^T (Ax - b) \right\}$$

It is well known that a necessary condition for the quadratic function $g_{\alpha,\beta,\delta,\lambda,\gamma}(x)+\gamma^T(Ax-b)$ to have a minimum not equal to $-\infty$ is that matrix $Q_{\alpha,\beta,\delta,\lambda}$ is positive semi-definite. Therefore $(Q3)$ is equivalent to $(Q4)$:

$$(Q4): \max_{\alpha\in\mathbb{R}^N,\beta\in\mathbb{R}^{T_1},\delta\in\mathbb{R}^{T_2},\lambda\in\mathbb{R}^{T_3},\gamma\in\mathbb{R}^M_+} \left\{ \min_{x\in\mathbb{R}^N} g_{\alpha,\beta,\delta,\lambda,\gamma}(x)+\gamma^T(Ax-b) \right\}$$

We know from [56] that $(Q4)$ is equivalent to problem (D):

$$(D)\begin{cases} \max t \\ \text{s.t.} \\ \begin{pmatrix} -\gamma^T b - t & \frac{1}{2}(c^T_{\alpha,\beta,\delta,\lambda}+\gamma^T A) \\ \frac{1}{2}(c_{\alpha,\beta,\delta,\lambda}+A^T\gamma) & Q_{\alpha,\beta,\delta,\lambda} \end{pmatrix} \succeq 0 \\ \quad t\in\mathbb{R},\ \alpha\in\mathbb{R}^N,\ \beta\in\mathbb{R}^{T_1},\ \delta\in\mathbb{R}^{T_2},\ \lambda\in\mathbb{R}^{T_3},\ \gamma\in\mathbb{R}^M_+ \end{cases}$$

By semi-definite duality of program (D), and with $\alpha,\beta,\delta,\lambda$ the dual variables associated with Constraints (4.8)–(4.11) respectively, we get (SDP'):

$$(SDP')\begin{cases} \min < Q, X > + c^T x \\ \text{s.t.} \\ \quad (4.8)-(4.13) \\ \quad Ax \le b \end{cases}$$

We now prove that there is no duality gap between (D) and (SDP'), which holds since:

(i) The feasible domain of (SDP') is nonempty, as $(QP_{\alpha,\beta,\delta,\lambda})$ contains 0 as a feasible solution and (D) is bounded

(ii) (D) satisfies Slater's condition. It is sufficient to take β, δ and λ equal to 0, α large enough so that $Q_{\alpha,\beta,\delta,\lambda}\succeq 0$ holds, and t a large negative number that ensures the diagonal dominance of the first row and the first column of matrix $\begin{pmatrix} -\gamma^T b - t & \frac{1}{2}(c^T_{\alpha,\beta,\delta,\lambda}+\gamma^T A) \\ \frac{1}{2}(c_{\alpha,\beta,\delta,\lambda}+A^T\gamma) & Q_{\alpha,\beta,\delta,\lambda} \end{pmatrix}$.

From these equivalences, we know that we can build an optimal solution of (CP) from the optimal dual variables of (SDP'). However, constraints $Ax\le b$ are redundant in (SDP') and we thus prove in Lemma 4.1.2 that (SDP') and (SDP) are equivalent. As a consequence, an optimal solution to (CP) can be deduced from the optimal dual variables of (SDP).

Lemma 4.1.2. *Due to Constraints (4.8)–(4.10) and (4.12), inequalities $Ax\le b$ are redundant in (SDP').*

Proof. Recall that $Ax \leq b$ are the inequalities of $(C^i_{j,k})$, $\forall (i,j,k) \in J \times (I \cup J)^2 : \ \mathcal{E}_i = \mathcal{E}_j \cup \mathcal{E}_k$, i.e $x_i \geq 0$ (a), $x_i \leq x_j$ (b), $x_i \leq x_k$ (c), and $x_i \geq x_j + x_k - 1$ (d).

The basic idea used here is that, since matrix $\begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix}$ is positive semi-definite, all its symmetric minors are non-negative.

- Constraint (a): $x_i \geq 0$. We consider the determinant $\begin{vmatrix} 1 & x_i \\ x_i & X_{ii} \end{vmatrix}$, which implies $X_{ii} - x_i^2 \geq 0$. By (4.8) we obtain $x_i - x_i^2 \geq 0$ and thus $x_i \geq 0$.

- Constraint (b): $x_i \leq x_j$. Considering the determinant of the symmetric minor $\begin{vmatrix} X_{jj} & X_{ji} \\ X_{ij} & X_{ii} \end{vmatrix}$ implies $X_{ii}X_{jj} - X_{ij}^2 \geq 0$. By (4.8) we have $x_j x_i - X_{ij}^2 \geq 0$ and by (4.9) we obtain $x_i x_j - x_i^2 \geq 0$. Either $x_i > 0$ and then we have $x_j - x_i \geq 0$, or $x_i = 0$ and the inequality comes from $x_j \geq 0$.

- Constraint (c): $x_i \leq x_k$. By symmetry, i.e. considering the determinant $\begin{vmatrix} X_{kk} & X_{ki} \\ X_{ik} & X_{ii} \end{vmatrix}$, the inequality holds.

- Constraint (d): $x_i \geq x_j + x_k - 1$. By definition (4.12) implies $z^T \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} z \geq 0$, $\forall z \in \mathbb{R}^{N+1}$. By taking $\bar{z} = (1, 0, .., 0, \underbrace{-1}_{j}, 0, .., 0, \underbrace{-1}_{k}, 0, .., 0, \underbrace{1}_{i}, 0, .., 0)$, we have:

$$\begin{aligned} 0 \leq \bar{z}^T \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \bar{z} &= (x_i + 1 - x_j - x_k) - (x_j - X_{jj} - X_{kj} + X_{ij}) \\ &\quad -(x_k - X_{kk} - X_{jk} + X_{ik}) + (x_i - X_{ji} - X_{ki} + X_{ii}) \\ &= (x_i + 1 - x_j - x_k) \text{ by (4.8), (4.9) and (4.10).} \end{aligned}$$

□

Let us state Corollary 1 that shows that from an optimal dual solution to (SDP) we can build an optimal solution to (CP).

Corollary 4.1.1. *We have $v(CP) = v(SDP)$. Consequently, an optimal solution $(\alpha^*, \beta^*, \delta^*, \lambda^*)$ of (CP) corresponds to the optimal values of the dual variables associated with constraints (4.8)–(4.11) of (SDP) respectively.*

Proof. We have:

(i) $v(CP) = v(D)$

(ii) since there is no duality gap between (D) and (SDP'), we have $v(D) = v(SDP')$

(iii) by Lemma 2, we get $v(CP) = v(D) = v(SDP') = v(SDP)$

□

□

To sum up, we obtain (QP^*), the best equivalent convex formulation to (QP):

$$(QP^*) \begin{cases} \min g_{\alpha^*,\beta^*,\delta^*,\lambda^*}(x) \\ \text{s.t.} \\ \quad x \in \mathcal{F}_{\mathcal{E}} \end{cases}$$

From Theorem 1, we deduce the Algorithm 1 to solve (P).

Algorithm 4.1.1 `PQCR` an exact solution method for (P)

Step 1: Apply a quadratization $\mathcal{E}$ to (P) and thus generate sets $\mathcal{F}_{\mathcal{E}}$ and $\mathcal{S}_{\mathcal{E}}$.
Step 2: Solve (SDP), deduce optimal values α^*, β^*, δ^*, λ^*, and build (QP^*).
Step 3: Solve (QP^*) by a standard quadratic convex programming solver.

Example 2 (continued). *We focus on the quadratic reformulation (QEx_2):*

$$g_2(x) = 2x_1 + 3x_2x_3 - 2\underbrace{x_3x_4}_{x_6}x_2 - 3\underbrace{x_1x_2}_{x_5}\underbrace{x_3x_4}_{x_6}$$

For this example, the semi-definite relaxation is

$$
(SDPEx)\begin{cases}
\min 2x_1 + 3X_{3,2} - 2X_{5,2} - 3X_{6,5} \\
s.t. \\
\quad X_{1,1} = x_1 \\
\quad X_{2,2} = x_2 \\
\quad X_{3,3} = x_3 \\
\quad X_{4,4} = x_4 \\
\quad X_{5,5} = x_5 \\
\quad X_{6,6} = x_6 \\
\quad X_{3,5} = x_5 \\
\quad X_{4,5} = x_5 \\
\quad X_{1,6} = x_6 \\
\quad X_{2,6} = x_6 \\
\quad X_{3,4} = x_5 \\
\quad X_{1,2} = x_6 \\
\quad \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 \\
\quad x \in \mathbb{R}^6,\ X \in S^6
\end{cases}
$$

and its solution leads to the following dual variables

- $\alpha^* = (2, 2, 1.5, 0.18, 4.47, 2.68)$
- $\beta^*_{1,5} = \beta^*_{5,1} = -1.98$
- $\beta^*_{2,5} = \beta^*_{5,2} = -0.98$
- $\beta^*_{3,6} = \beta^*_{6,3} = -1.59$
- $\beta^*_{4,6} = \beta^*_{6,4} = -0.18$
- $\gamma^*_{1,2} = \gamma^*_{2,1} = -0.02$
- $\gamma^*_{3,4} = \gamma^*_{4,3} = 0.09$

The equivalent program $(QP_{\alpha,\beta,\delta,\lambda})$ *is written as follows*

$$
(QP_{\alpha,\beta,\delta,\lambda}) \begin{cases} \min g_{\alpha,\beta,\delta,\lambda}(x) = 2x_1 + 3x_2x_3 - 2x_2x_6 - 3x_5x_6 + 1(x_1^2 - x_1) + 1(x_2^2 - x_2) \\ +0.7(x_3^2 - x_3) + 0.09(x_4^2 - x_4) + 2.2(x_5^2 - x_5) + 1.3(x_6^2 - x_6) \\ -3.96(x_1x_5 - x_5) - 1.96(x_2x_5 - x_5) - 3.18(x_3x_6 - x_6) - 0.36(x_4x_6 - x_6) \\ -0.04(x_1x_2 - x_5) + 0.18(x_3x_4 - x_6) \\ s.t. \\ \quad x \in \mathcal{F}_{\mathcal{E}} \end{cases}
$$

The Hessian matrix of $g_{\alpha,\beta,\delta,\lambda}$ after the diagonal perturbation is

$$
Q_{\alpha,\beta,\delta,\lambda} = \begin{pmatrix} 2 & -0.02 & 0 & 0 & -1.98 & 0 \\ -0.02 & 2 & 1.5 & 0 & -0.98 & -1 \\ 0 & 1.5 & 1.5 & 0.09 & 0 & -1.59 \\ 0 & 0 & 0.09 & 0.18 & 0 & -0.18 \\ -1.98 & -0.98 & 0 & 0 & 4.47 & -1.5 \\ 0 & -1 & -1.59 & -0.18 & -1.5 & 2.68 \end{pmatrix}
$$

Degree	*Method*	*Continuous relaxation bound value*
1	*Standard linearization*	-1.5
d	*Uniform interval*	-5.3
d	*Non-uniform interval*	-4.3
2	*Smallest eigenvalue*	-1.7
2	*QCR*	-1.6
2	*PQCR*	-0.6

4.2 Discussion on the impact of the chosen 2×2 quadratization

In the general framework of PQCR described in the previous section, we arbitrarily choose a 2×2 quadratization for the first phase of the method. Moreover, the set of valid equalities $S_{\mathcal{E}}$ involved in the convexification phase is derived from the quadratization chosen in the first phase. We experimentally observe that this choice impacts the quality of the continuous relaxation bound after convexification. A comparative computational study can be found in Chapter 7.

In this section, we concentrate ourselves on the first phase of the PQCR method, i.e. we study different quadratizations of (P) and more particularly the quadratizations that are *stable*

by convexification. By stable, we mean that we are looking for families of quadratizations that lead to the same continuous relaxation bound after convexification. The purpose of this study can be seen as an attempt to classify the quadratizations depending on the bound obtained after convexification. Indeed, we can identify quadratizations that attain the same bound after convexification, leading to a *family of quadratizations*. These families can then be compared in terms of continuous relaxation bound. Nevertheless, we do not pretend to draw a totally ordered set, i.e. not every pair of quadratizations can be compared as it will be presented in the final example of this section. Instead the idea is rather to underline special cases where we can compare the continuous relaxation bounds of two quadratizations. Even if we are not able to answer the ultimate question of finding the 2×2 quadratization that gives the best bound with the `PQCR` framework, the following study provides partial answers.

We start this section by redefining the concept of quadratization. Indeed, in Definition 4.1.1, we did not take into account the multiple ways of defining quadratic reformulations given a set $\mathcal{E}$ and a polynomial f. As we discuss on the impact of quadratization, we have to distinguish between all the quadratic functions g obtained after the quadratization phase, given $\mathcal{E}$ and f.

Definition 4.2.1. *Given sets $J = \{n+1, .., N\}$ and $\{\mathcal{E}_i,\ i \in I \cup J\}$ we can define several quadratic reformulations $f^r_{\mathcal{E}}$ of the initial objective function f. A valid quadratization with N variables is a reformulation $f^r_{\mathcal{E}}$ such that, for any monomial p of f of degree greater than or equal to 3 (i.e. $|\mathcal{M}_p| \geq 3$), there exist $(j,k) \in (I \cup J)^2$ such that $\mathcal{M}_p = \mathcal{E}_j \cup \mathcal{E}_k$ and $\prod_{i \in \mathcal{M}_p} x_i = x_j x_k$. Then the monomial p is replaced by a quadratic term in $f^r_{\mathcal{E}}$.*

This definition allows us to consider all the possible decompositions of the objective function into a quadratic one. Indeed, given a set $\mathcal{E}$, there are several ways to reformulate the objective function. The choices are made on the variables that will appear in the objective function.

We then properly define a family of quadratizations.

Definition 4.2.2. *We consider B sets $(\mathcal{E}^1, ..., \mathcal{E}^B)$ (not necessarily different) and B corresponding decompositions of the objective function f $(r_1, ..., r_B)$. Let $f^{r_1}_{\mathcal{E}^1}, ..., f^{r_B}_{\mathcal{E}^B}$ be B different quadratizations of f. We then define a family of quadratizations as the finite sequence $f^{r_i}_{\mathcal{E}^i},\ (i = 1, ..., B)$.*

This definition of family of quadratizations allows us to consider all the possible decompositions of the objective function into a quadratic one and to cluster quadratizations that have the same characteristic, namely the continuous relaxation bound value. We say that a family of quadratization is stable if any quadratization of this family leads to the same continuous relaxation bound value after convexification by `PQCR` algorithm. We state more formally this definition in the following.

Definition 4.2.3. *A family of quadratizations $f^{r_i}_{\mathcal{E}^i},\ (i = 1, ..., B)$ is stable by convexification if, for each pair (i,j) of $\{1, ..., B\}^2$, we have $v(\overline{QP}_{f^{r_i}_{\mathcal{E}^i}}) = v(\overline{QP}_{f^{r_j}_{\mathcal{E}^j}})$.*

In the rest of the section, we are searching for quadratizations that can be compared to others. In particular, two quadratizations that give the same bound after convexification can be qualified as "equivalent". Then, we can decide to choose the one that has the smallest number of additional variables for example to fasten computations. Establishing stable families of convexification can be useful to compare quadratizations based on other criterion than the bound criterion. We next bring these definitions to light with an example comparing different quadratizations.

Example 2 (continued). *We present two different sets $\mathcal{E}$ for (Ex).*

- *$\mathcal{E}^1$ is defined by $\mathcal{E}^1_5 = \{2,3\}$, $\mathcal{E}^1_6 = \{1,2\}$, $\mathcal{E}^1_7 = \{3,4\}$, $\mathcal{E}^1_8 = \{1,4\}$, $\mathcal{E}^1_9 = \{1,3\}$ and $\mathcal{E}^1_{10} = \{2,4\}$.*
- *$\mathcal{E}^2$ is defined by $\mathcal{E}^1_5 = \{2,3\}$ and $\mathcal{E}^1_6 = \{1,4\}$.*

From $\mathcal{E}^1$, we can derive at least two different quadratizations,

- $f^1_{\mathcal{E}^1}(x) = 2x_1 + 3x_2x_3 - 2x_5x_4 - 3x_6x_7$
- $f^2_{\mathcal{E}^1}(x) = 2x_1 + 3x_2x_3 - 2x_2x_7 - 3x_5x_8$

  Then from $\mathcal{E}^2$ we can derive only one valid quadratization

- $f^1_{\mathcal{E}^2}(x) = 2x_1 + 3x_2x_3 - 2x_4x_5 - 3x_5x_6$

The optimal solution in binary variables for (Ex) is 0. The continuous relaxation bound value obtained after convexification is -0.015 for $f^1_{\mathcal{E}^1}$ and $f^2_{\mathcal{E}^1}$, whereas it is -0.125 for $f^1_{\mathcal{E}^2}$. We can conclude that the family of quadratizations $(f^1_{\mathcal{E}^1}, f^2_{\mathcal{E}^1})$ is stable by convexification.

From the example, it appears intuitively that for a given set $\mathcal{E}$, all the quadratizations $f^i_{\mathcal{E}}$ using this set of additional variables have the same bound.

In the same way, the difference between the bound values for the quadratizations in the example seems rather intuitive. Indeed, adding more additional variables means adding more valid equalities to the convexification. One can ask whether a quadratization introducing more additional variables has always a better bound than a quadratization introducing less variables. Unfortunately, this does not hold in the general case as it will be seen in a counter example. However, we can formalize this result in the special case of one quadratization being included in an other.

We state and prove a first theoretical result on the bounds obtained by different quadratizations based on the same set $\mathcal{E}$. Given a set $\mathcal{E}$, all the quadratizations using this set have the same continuous relaxation bound. In other words, the choice of the variables that appear in the objective function does not impact the continuous relaxation bound value.

Proposition 4.2.1. *Let $\mathcal{E}$ be a set representing the decomposition of additional variables and let $f^i_{\mathcal{E}}$ and $f^j_{\mathcal{E}}$ be two quadratizations based on $\mathcal{E}$. Then $v(\overline{QP}_{f^i_{\mathcal{E}}}) = v(\overline{QP}_{f^j_{\mathcal{E}}})$*

Proof. We focus on semi-definite programs $(SDP^i_{\mathcal{E}})$ and $(SDP^j_{\mathcal{E}})$ as $v(\overline{QP}_{f^i_{\mathcal{E}}}) = v(SDP_{f^i_{\mathcal{E}}})$. The set of constraints is the same for both programs. We prove that the value of the bijective functions are the same. Let us consider a monomial αX_{ij} in the objective function of $(SDP_{f^i_{\mathcal{E}}})$ which corresponds to the monomial αX_{kl} in the objective function of $(SDP_{f^i_{\mathcal{E}}})$. Constraints (4.11) ensures the equality $X_{ij} = X_{kl}$ when $\mathcal{E}_i \cup \mathcal{E}_j = \mathcal{E}_k \cup \mathcal{E}_l$, which is the case here. We thus have the equality between the two monomials. By iterating this observation for all the monomials, we obtain the equality between $v(SDP_{f^i_{\mathcal{E}}})$ and $v(SDP_{f^j_{\mathcal{E}}})$.

□

This result gives a first example of family of stable quadratizations. Indeed, given a set $\mathcal{E}$, we can equally consider all the possible decompositions of the objective function in terms of continuous relaxation bound. This is comforting as the choice of variables in the quadratic objective function does not add unnecessary complexity to find stable reformulations. Thus the concept of quadratization proposed in definition 4.1.1 is sufficient. In the following we can now consider quadratic reformulations $f_{\mathcal{E}}$ instead of $f^r_{\mathcal{E}}$ since the decomposition order does not play a role neither in the size of the problem nor in the quality of the continuous relaxation bound.

Next, we present a result that orders some family of quadratizations according to the continuous relaxation bound. More precisely, we establish a domination result on quadratizations. First, we introduce a notation that allows a comparison between the size (number of variables) of different quadratizations in the following definition.

Definition 4.2.4. *We consider two sets $\mathcal{E}^1$ and $\mathcal{E}^2$ involving N_1 and N_2 variables respectively, with $N_1 \leq N_2$. We say that the quadratization $\mathcal{E}^1$ is included in the quadratization $\mathcal{E}^2$ if for any $i \in \{1, ..., N_1\}$, there exists $j \in \{1, ..., N_2\}$ such that $\mathcal{E}^1_i = \mathcal{E}^2_j$. We then note $\mathcal{E}^1 \subset \mathcal{E}^2$.*

This definition states that quadratization $\mathcal{E}^1$ is included into quadratization $\mathcal{E}^2$ if every product of variables reformulated in $\mathcal{E}^1$ is also reformulated in $\mathcal{E}^2$. In other words, $\mathcal{E}^2$ contains all the additional variables of $\mathcal{E}^1$. When considering these special case of quadratizations, we can compare the bounds obtained by continuous relaxation as it is shown in the following Lemma.

Proposition 4.2.2. *Let $\mathcal{E}^1$ and $\mathcal{E}^2$ be two quadratizations such that $\mathcal{E}^1 \subset \mathcal{E}^2$. Then $v(\overline{QP}_{f_{\mathcal{E}^1}}) \leq v(\overline{QP}_{f_{\mathcal{E}^2}})$.*

Proof. Again, we prove the inequality on the underlying semi-definite relaxations by equivalence, that is $v(SDP_{f_{\mathcal{E}^1}}) \leq v(SDP_{f_{\mathcal{E}^2}})$.

Let $x^* = (x^*_1, ..., x^*_{N_2})$ be the optimal solution of $(SDP_{f_{\mathcal{E}^2}})$. The truncated solution $y^* = (x^*_1, ..., x^*_{N_1})$ satisfies constraints (4.8 - 4.11) of $(SDP_{f_{\mathcal{E}^1}})$ since these constraints are a subset of the constraints of $(SDP_{f_{\mathcal{E}^2}})$. We now show that y^* satisfies the positive semi-definite constraints. For this, we state and prove the following two well-known results on positive semi-definite matrices.

Property 4.2.1. *The property of positive semi-definiteness is invariant by symmetric permutations of rows and columns.*

Proof. Let A be a symmetric positive semi-definite matrix. We show that, for every permutation matrix P, $PAP^T \succeq 0$.

By application of the spectral theorem for matrix A, there exist an orthogonal matrix O and a positive diagonal matrix D such that $A = ODO^T$. We thus have $PAP^T = PODO^TP^T = (PO)D(PO)^T \succeq 0$. □

Property 4.2.2. *The property of positive semi-definiteness is invariant by symmetric suppression of some rows and the associated corresponding columns.*

Proof. Consider the suppression of the last row and the last column of a semi-definite matrix M. The resulting matrix M' is also positive semi-definite since the determinant of every symmetric minors is positive a fortiori. Using Property 4.2.1, we can generalize the invariance for the suppression of any number of symmetric rows and columns. □

With these two results, we can conclude that the truncated solutions y^* satisfies constraints (4.12).

Finally, as $\mathcal{E}^1 \subset \mathcal{E}^2$, with Theorem 4.2.1 we can deduce that, without loss of generality, objective functions of $(SDP_{f_{\mathcal{E}^1}})$ and $(SDP_{f_{\mathcal{E}^2}})$ are the same and thus have the same value.

We have proven that from an optimal solution of $(SDP_{f_{\mathcal{E}^2}})$ we can deduce a feasible solution of $(SDP_{f_{\mathcal{E}^1}})$ with the same objective value. Thus $v(SDP_{f_{\mathcal{E}^1}}) \leq v(SDP_{f_{\mathcal{E}^2}})$. □

This result confirms an intuitive observation that is, adding more auxiliary variables to a quadratization can sharpen the continuous relaxation bound after convexification. Indeed, the more we introduce additional variables, the more we add valid equalities to the reformulated problem. Thus the bound is expected to be tighter when adding more variables. However, the inequality obtained in the previous Lemma is not always strict. We illustrate this point in the following example.

Example 2 (continued). *We consider 2 quadratizations for (Ex), $\mathcal{E}^1$ and $\mathcal{E}^2$, with $\mathcal{E}^1 \subset \mathcal{E}^2$:*

- *Quadratization $\mathcal{E}^1$ with 2 additional variables, $\mathcal{E}^1_5 = \{2,3\}$ and $\mathcal{E}^1_6 = \{1,4\}$:*

$$\min 2x_1 + 3x_2x_3 - 2\underbrace{x_2x_3}_{x_5}x_4 - 3\underbrace{x_1x_4}_{x_6}\underbrace{x_3x_2}_{x_5}$$

  Continuous relaxation bound value : -0,125

- *Quadratization $\mathcal{E}^2$ with 3 additional variables, $\mathcal{E}^2_5 = \{2,4\}$, $\mathcal{E}^2_6 = \{1,4\}$ and $\mathcal{E}^2_7 = \{2,3\}$:*

$$\min 2x_1 + 3x_2x_3 - 2\underbrace{x_2x_4}_{x_5}x_3 - 3\underbrace{x_1x_4}_{x_6}\underbrace{x_3x_2}_{x_7}$$

Continuous relaxation bound value : -0,125

The bound obtained by the two quadratizations are equal. From a computational point of view, quadratization 1 *should be chosen as it has less variables than quadratization* 2 *while having the same bound.*

We have proven that, in the special case of a quadratization included in another, we can derive an inequality on the value of the continuous relaxation bound. One can now ask if we can derive other relations on bounds for a more general case of quadratization. Unfortunately, we show throughout the next example that the number of additional variables is not always correlated with the quality of continuous relaxation bounds in the general case.

Example 2 (continued). *We introduce 3 different quadratizations of* (Ex) *and we present the continuous relaxation bounds obtained after convexification.*

- *Quadratization* $\mathcal{E}^1$ *with 2 additional variables,* $\mathcal{E}^1_5 = \{2,3\}$ *and* $\mathcal{E}^1_6 = \{1,4\}$*:*

$$\min 2x_1 + 3x_2x_3 - 2\underbrace{x_2x_3}_{x_5}x_4 - 3\underbrace{x_1x_4}_{x_6}\underbrace{x_3x_2}_{x_5}$$

  Continuous relaxation bound value : -0,125

- *Quadratization* $\mathcal{E}^2$ *with 3 additional variables,* $\mathcal{E}^2_5 = \{2,3\}$*,* $\mathcal{E}^2_6 = \{1,2\}$ *and* $\mathcal{E}^2_7 = \{3,4\}$*:*

$$\min 2x_1 + 3x_2x_3 - 2\underbrace{x_2x_3}_{x_5}x_4 - 3\underbrace{x_1x_2}_{x_6}\underbrace{x_3x_4}_{x_7}$$

  Continuous relaxation bound value : -0,625

- *Quadratization* $\mathcal{E}^3$ *with 3 additional variables,* $\mathcal{E}^3_5 = \{2,4\}$*,* $\mathcal{E}^3_6 = \{1,2\}$ *and* $\mathcal{E}^3_7 = \{3,4\}$*:*

$$\min 2x_1 + 3x_2x_3 - 2\underbrace{x_2x_4}_{x_5}x_3 - 3\underbrace{x_1x_2}_{x_6}\underbrace{x_3x_4}_{x_7}$$

  Continuous relaxation bound value : -0,375

We cannot draw any inclusion relationship between these three quadratizations. We remark that the best bound is attained by the quadratization that has the smallest number of variables ($\mathcal{E}^1$)*. This seems counterintuitive since adding less variables means adding less valid equalities. However, the example shows that the valid equalities induced by* $\mathcal{E}^1$ *(and thus the choice of additional variables) may be stronger than the ones of* $\mathcal{E}^2$ *and* $\mathcal{E}^3$*. Moreover, we have an example of two quadratizations that use the same number of additional variables and whose bounds are not equal* ($\mathcal{E}^2$ *and* $\mathcal{E}^3$)*. Again these quadratizations are not comparable in terms of bounds. This is in accordance with the fact that it can be difficult to build a totally ordered set, at least when considering the size criterion only.*

Our study on stability shows that considering the dimension of (QP) is not enough to compare the bounds of different quadratizations in the general case. However, we know that a quadratization $\mathcal{E}^1$ "larger" than other quadratizations $\mathcal{E}^i$ (i.e. when $\mathcal{E}^i \subset \mathcal{E}^1$) has a bound that is at least equal to the bounds of the quadratizations $\mathcal{E}^i$. One solution to have the best possible bound would be to build a quadratization that contains all the other quadratizations. Unfortunately, this is impossible as we can build quadratizations with infinitely many additional variables that recursively reformulate product of variables. Nevertheless it can be interesting to add all the variables "up to some fixed degree" to obtain a satisfying bound. Next, we address this problem and we present two special cases of quadratizations that are interesting both in computational and theoretical point of view. In particular, we present comparisons of our quadratic convex reformulation using these quadratizations with other state of the art methods.

4.2.1 2×2 Full quadratization

In this section we introduce the 2×2 full quadratization (also called *full quadratization*). A full quadratization of a binary polynomial f of degree d consists in the introduction of an additional variable for each possible product of variables up to degree $\lceil \frac{d}{2} \rceil$. In other words, each product of initial variables of degree at most $\lceil \frac{d}{2} \rceil$ is reformulated by an additional variable. We state more formally the definition in Definition 4.2.5.

Definition 4.2.5. *We define by full quadratization, any quadratization $\mathcal{E}$ such that for every product $\prod_j x_j$ of degree at most $\lceil \frac{d}{2} \rceil$, there exist $i \in J$ such that $\mathcal{E}_i = \bigcup_j \mathcal{E}_j$.*

The bound $\lceil \frac{d}{2} \rceil$ is not an arbitrary choice. Indeed, let us notice that in order to compute a quadratic reformulation of a binary monomial of degree d, we need at least two additional variables x_i and x_j and the monomial will be reformulated as $x_i x_j$. As a consequence one of these additional variables reformulates a product of variables of degree at least $\lceil \frac{d}{2} \rceil$. When considering the quadratic reformulation of a degree d polynomial, one need to introduce at least one variable of degree greater than or equal to $\lceil \frac{d}{2} \rceil$.

In the last section, we supposed that "bigger" quadratizations (in the sense of inclusion) could lead to tighter bounds. We thus need to find a quadratization that possesses a property of "completeness", that is a quadratization containing all the additional variables up to a certain degree d'. And, in the case of a degree d polynomial, d' is precisely $\lceil \frac{d}{2} \rceil$. Indeed, we can quadratically reformulate any polynomial of degree d with additional variables of degree at most $\lceil \frac{d}{2} \rceil$. Moreover, we cannot compute a quadratic reformulation of a degree d polynomial with additional variables of degree at most $\lceil \frac{d}{2} \rceil - 1$ as monomials of degree d can not be reformulated with the product of two additional variables. Thus, the full quadratization is the smallest quadratization that can be used to quadratically reformulate any polynomial of degree d and that contains all the other quadratizations using additional variables of degree less than or equal to $\lceil \frac{d}{2} \rceil$.

We can remark that this quadratization is not sensitive to the structure of the initial problem as the number of additional variables only depends on the number of initial variables n. Indeed, this quadratization reformulates every product of variables, even if it does not appear in the objective function. In other words, the full quadratization will introduce the same number of variables for a polynomial program with 10 initial variables and 10 monomials or a program with 10 variables and 1000 monomials.

This full quadratization has been introduced and used in different frameworks. For example it corresponds to the propery of *upward completeness* that was introduced in [21]. The authors use this quadratization in a quadratic reformulation framework. We further study the impact of this quadratization in a quadratic convex reformulation framework. Moreover, this quadratization is also linked with the Lasserre's hierarchy for unconstrained binary polynomial programs [51] as it will be stated further.

We next present an example of a full quadratization and we discuss on the dimension of such a quadratization.

Example 4. *We consider the following new polynomial program which optimal value is -2*

$$(Ex2)\left\{ \min_{x\in\{0,1\}^4} 2x_1x_5 + 3x_2x_3 - 2x_2x_3x_4 - 3x_1x_2x_3x_4 \right.$$

Applying the full quadratization to $(Ex2)$ *amounts to introduce an additional variable for each product in red in the following matrix:*

$$xx^T = \begin{pmatrix} x_1^2 & x_1x_2 & x_1x_3 & x_1x_4 & x_1x_5 \\ x_2x_1 & x_2^2 & x_2x_3 & x_2x_4 & x_2x_5 \\ x_3x_1 & x_3x_2 & x_3^2 & x_3x_4 & x_3x_5 \\ x_4x_1 & x_4x_2 & x_4x_3 & x_4^2 & x_4x_5 \\ x_5x_1 & x_5x_2 & x_5x_3 & x_5x_4 & x_5^2 \end{pmatrix}$$

We add a total of 10 *additional variables which correspond to every possible product of degree* 2*. The bound obtained is* $-0,015$*, which is very tight since this quadratization leads to better (or equal) bounds than all the other considered* 2×2 *quadratizations.*

Given a number of initial variables n *and the degree* d*, the full quadratization will always introduce the same number of additional variables. More precisely, applying full quadratization to any degree* d *polynomial with* 5 *variables will lead to the same set* $\mathcal{E}$ *with these* 10 *additional variables.*

As it is illustrated in the example, the full quadratization does not depend on the number of monomials, it only depends on the number of initial variables n and the degree d. Indeeed, the number of additional variables added is $\mathcal{O}(n^{\frac{d}{2}})$, which can be very large. Although the bound obtained after convexification can be very tight, this quadratization is intractable when considering medium or large instances as it is explained in [21]. In the next paragraph, we

present a quadratization "extracted" from the full quadratization which can be interesting from a computational point of view.

4.2.2 2×2 Partial quadratization

The full quadratization presented in the last paragraph introduced a quadratization with some completeness property and lead to very tight bounds. However, the number of additional variables can be huge for many instances. One idea could be to reduce the number of additional variables according to the structure of each polynomial instance. One way to proceed is to take the structure of the monomials into consideration. To reduce the number of variables of the full quadratization, without overly altering the bound, we can build a quadratization which only reformulates product of variables appearing in a same monomial of degree greater than 2. In the following, we present and define formally such a quadratization that we call *partial quadratization*.

Definition 4.2.6. *We define by partial quadratization, any quadratization $\mathcal{E}$ such that for every product $\prod_j x_j$ of degree $\lceil \frac{d}{2} \rceil$ contained in a same monomial of degree greater than 2, there exist $i \in J$ such that $\mathcal{E}_i = \bigcup_j \mathcal{E}_j$.*

Trivially, the partial quadratization is included in the full quadratization and thus has less additional variables. Moreover the continuous relaxation bounds are expected to be weaker than the ones with full quadratization by Theorem 4.2.2. The question is to know whether the partial quadratization has a better trade-off between dimension and quality of the bound than the full quadratization. Again, the authors of [21] introduce the property of *downward completeness* which corresponds to the partial quadratization. In the case of quadratic reformulation, the authors state that this quadratization is far more tractable than the first one. In the following example we illustrate the application of partial quadratization to our quadratic convex reformulation framework and we compare the results obtained with the ones obtained by full quadratization.

Example 4 (continued). *Applying the partial quadratization to $(Ex2)$ amounts to reformulate all the products in blue in the following matrix:*

$$M = \begin{pmatrix} x_1^2 & x_1x_2 & x_1x_3 & x_1x_4 & x_1x_5 \\ x_2x_1 & x_2^2 & x_2x_3 & x_2x_4 & x_2x_5 \\ x_3x_1 & x_3x_2 & x_3^2 & x_3x_4 & x_3x_5 \\ x_4x_1 & x_4x_2 & x_4x_3 & x_4^2 & x_4x_5 \\ x_5x_1 & x_5x_2 & x_5x_3 & x_5x_4 & x_5^2 \end{pmatrix}$$

We add a total of 6 additional variables to quadratize the last two monomials. Again, the bound obtained is $-0,015$ which is the same as the bound obtained by the full quadratization. With this instance it is shown that the inequality between the two quadratizations can be tight.

Moreover, this partial quadratization takes advantage of the structure of the instance. In other words, an other instance would lead to an other set $\mathcal{E}$ with different additional variables.

In Table 4.1, we present experimental results on small low autocorrelation instances illustrating our study on quadratizations. We apply the `PQCR` algorithm with the two quadratizations introduced in this section, namely the full quadratization (Full) and the partial quadratization (Partial). We compare the total number of variables (N), the number of constraints in the semi-definite relaxation (*Const SDP*), the continuous relaxation bound values (LB) and the total solution time (T_t).

Instance			`PQCR` with Full		`PQCR` with Partial	
Name	n	*Opt*	N	*Const SDP*	N	*Const SDP*
`b.20.5`	20	-416	210	22156	83	1308
`b.20.10`	20	-2936	210	22156	138	6275
`b.20.15`	20	-5960	210	22156	176	13757

Instance			`PQCR` with Full		`PQCR` with Partial	
Name	n	*Opt*	LB	T_t	LB	T_t
`b.20.5`	20	-416	-417	9700	-422	3
`b.20.10`	20	-2936	-3016	7439	-3040	180
`b.20.15`	20	-5960	-6025	10831	-6059	2060

Table 4.1: *Comparison of the full quadratization, the partial quadratization and the quadratization of Algorithm 4.2.1*

The numerical results indicate that the bounds obtained by the full and the partial quadratizations are not always equal, but the partial quadratization seems to have a better trade-off which leads to a faster resolution. Indeed the partial quadratization solves every instance within 30 minutes whereas the full quadratization solves the hardest instance in 3 hours. Moreover, we cannot compare these two quadratizations on other instances as the full quadratization does not scale on bigger instances. The small loss on the bound quality can be explained as some of the additional variables in the full quadratization reformulate products that do not appear in any decomposition of the quadratic objective function. We may suppose that these variables play a minor role in the quality of the bound since they do not appear explicitly in the objective function. Thus, the partial quadratization contains a subset of additional variables that can be qualified as the "most significant" variables in terms of bound quality. Note that the number of variables added by the full quadratization is always the same because all the instances have the same number of initial variables. This is another example of why the full quadratization does not "suit" well. A future research work would be to measure theoretically, for any polynomial instance, the difference of bounds between the two quadratizations.

4.2.3 Replacing PQCR's quadratization scheme with other pairwise covers

In this section, we shortly explore the experimental impact of the chosen quadratization on the tightness of the associated continuous relaxation bound after quadratic convex reformulation. In Table 4.2, we report the continuous relaxation bound values obtained by convexification after applying the quadratization used in PQCR, called PC0, and three quadratizations from [71], namely Pairwise Cover 1, 2 and 3 (PC1, PC2 and PC3). We first describe, in Algorithm 4.2.1, our quadratization algorithm, PC0 used in PQCR.

Algorithm 4.2.1 PC0

Require: A polynomial f of degree $d > 2$

Ensure: A quadratic function f' verifying $\forall x \in \{0,1\}^n,\ f'(x) = f(x)$

```
for each monomial p from 1 to m do
  Sort p by lexicographical order
  deg ← deg(p)
  while deg > 2 do
    s ← ⌊deg/2⌋
    for l from 1 to s do
      Consider the l^th consecutive pair of variables x_j x_k
      Find x_i that represents the product x_j x_k
      if x_i does not exist then
        Create an additional variable x_i and E_i ← E_j ∪ E_k
      end if
      Replace x_j x_k by x_i
    end for
    deg ← ⌈deg/2⌉
  end while
end for
```

Example 5. *Applying the quadratization of Algorithm 4.2.1 to the monomial $x_1x_2x_3x_4x_5$ we obtain the following monomial of degree 3 at the first iteration:*

$$\underbrace{x_1x_2}_{x_6}\underbrace{x_3x_4}_{x_7}x_5$$

we then obtain a quadratic reformulation of the monomial at the second iteration using 3 additional variables:

$$\underbrace{x_6x_7}_{x_8}x_5$$

□

In Pairwise Cover 1, for each monomial of degree $d \geq 3$, the first two variables are linearized to obtain a monomial of degree $d-1$. The process is recursively reproduced until $d = 2$. Pairwise Covers 2 and 3 try to minimize the number of additional variables. In PC2, the authors compute the sub-monomials of any degree that appear the most among all the intersection of pairs of monomials. Then they linearize these sub-monomials using the set $\mathcal{F}_\mathcal{E}$ and they repeat the process until the objective function is quadratic. PC3 linearizes in priority the pair of variables that occurs the most frequently in all the monomials. For instance, if we consider the quadratization of the following monomial of degree 4, $x_1x_2x_3x_4$, we will compute the most frequent pair of variables among the six possible products. If x_1x_2 is the most frequent, then the monomial will be quadratized using two variables, one for the reformulation of x_1x_2 and the other for x_3x_4.

Instance		PQCR with $PC1$			PQCR with $PC2$			PQCR with $PC3$			PQCR with $PC0$		
Name	*Opt*	N	LB	T_t	N	LB	T_t	N	LB	T_t	N	LB	T_t
b.20.05	-416	64	-435	70	56	-439	63	40	-436	59	65	-435	64
b.20.10	-2936	123	-3052	112	135	-3115	132	93	-3068	112	124	-3051	130
b.40.10	-8248	303	-8590	4385	315	-8659	4562	262	-8745	2162	304	-8589	3723

Table 4.2: *Comparison of bounds (LB), number of variables (N) and total time (T_t) of* PQCR *after different quadratizations*

By considering experimental results, we remark that PC1 and quadratization PC0 have similar behavior in terms of dimension and bounds. PC0 has the best bounds among all the other quadratization whereas PC3 has the smallest number of variables on every instance. Thus the chosen quadratization impacts N, the number of variables of (QP). It also impacts the quality of the associated semi-definite bound, LB_i. Indeed, we know from the study in stability that, the more variables are added, the more the size of sets $\mathcal{F}_\mathcal{E}$ and $\mathcal{S}_\mathcal{E}$ increases. As there is not any inclusion relation between the pairwise covers, they cannot be compared. As a consequence some equalities of $\mathcal{S}_\mathcal{E}$ may be stronger than others. Interesting future research directions would be to identify, for a given quadratization, a set of "important" equalities in $\mathcal{S}_\mathcal{E}$, and to determine which quadratization used in PQCR leads to faster solution time and/or sharper initial lower bound.

To conclude this section, after acknowledging that the bound after convexification varies depending on the quadratization, we defined the stability property of quadratizations with respect to the convexification phase. We have presented a first result on stability which proves that different quadratic decompositions of a polynomial function using a single set $\mathcal{E}$ lead to the same bound. Then, we have presented a special case where we are able to order the quadratizations according to their bounds. Finally, we have introduced two special cases of quadratizations that are linked to state of the art methods. The full quadratization is a theoretical quadratization which leads to very sharp bounds, whereas the partial quadratization is a more tractable quadratization which can be used in practice to have good bounds with a relatively small number of additional vari-

ables. In the next section, we explore the connexion between PQCR and the well-known Lasserre's hierarchy. We will see that the full quadratization allows us to link the two methods.

4.3 Link with the Lasserre's hierarchy

In this section we establish the link between PQCR and the Moment/SOS method for unconstrained binary polynomial programs. We show that we can use the powerful Moment/SOS method to compute a quadratic convex reformulation for each order of the Lasserre's hierarchy. Moreover following the same ideas, we can also build a hierarchy of quadratic convex reformulations of (P) whose optimal continuous relaxation values converge towards the optimal value of (P).

4.3.1 A hierarchy of exact quadratic convex reformulations

Our algorithm PQCR is built to compute a quadratic convex reformulation given a quadratization phase. For special cases of quadratization, PQCR gives the same bounds as the hierarchy of Lasserre. We first characterize these quadratizations. We then show that for each order of the Lasserre's hierarchy, we can deduce a quadratic convex reformulation. Using this property we can build our own hierarchy of quadratic convex reformulations based on the Moment/SOS method and taking benefit of all the convergence properties established by Lasserre. Finally we show that, compared to the Lasserre's hierarchy for unconstrained binary programs, PQCR can lead to a better tractability on bigger problems. Indeed, it is possible to compute quadratic convex reformulations that do not correspond to any given order of the hierarchy. This property has a great practical impact. It allows PQCR to solve bigger instances than the Moment/SOS method by using lighter SDP relaxations. In particular this explains why PQCR is able to solve several low autocorrelation instances (see Chapter 7) whereas GloptiPoly cannot.

We start by proving that it is possible to choose a quadratization in PQCR such that the continuous relaxation bound value given by PQCR is the same as the relaxation value of the first order of the Lasserre's hierarchy. Indeed, using the full quadratization in PQCR leads to the same semidefinite relaxation.

Observation 4.3.1. *Let $\mathcal{E}$ be the 2×2 full quadratization. Then $(SDP_{\mathcal{E}})$ is $(M_{\lceil \frac{d}{2} \rceil})$.*

The two semidefinite programs have the same dimension and it suffices to remark that $(M_{\lceil \frac{d}{2} \rceil})$

can be written as

$$
(SDP)\begin{cases}
\min < Q, X > + c^T x & \\
\text{s.t.} & \\
\quad X_{ii} - x_i = 0 & i \in I \cup J \qquad (4.14)\\
\quad -X_{ij} + x_i = 0 & (i,j) \in J \times (I \cup J) : \ \mathcal{E}_j \subset \mathcal{E}_i \qquad (4.15)\\
\quad -X_{jk} + x_i = 0 & (i,j,k) \in J \times (I \cup J)^2 : \ \mathcal{E}_i = \mathcal{E}_j \cup \mathcal{E}_k \qquad (4.16)\\
\quad X_{ij} - X_{kl} = 0 & (i,j,k,l) \in (I \cup J)^4 : \ \mathcal{E}_i \cup \mathcal{E}_j = \mathcal{E}_k \cup \mathcal{E}_l \qquad (4.17)\\
\quad \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 & \qquad (4.18)\\
\quad x \in \mathbb{R}^N, \ X \in S^N & \qquad (4.19)
\end{cases}
$$

since constraints (4.14-4.18) are equivalent to $M_{\lceil \frac{d}{2} \rceil} \succeq 0$. (See for example the moment matrix described in Section 2.2).

The interest of the full quadratization presented in Definition 4.2.5 is rather theoretical. By construction, this quadratization contains all the variables up to degree $\lceil \frac{d}{2} \rceil$ and thus explicits the comparison with the first order of the Lasserre's hierarchy. Moreover, this connection can be done systematically for each order of the hierarchy. More precisely, the following Lemma states that for each order of the Lasserre's hierarchy, it is possible to choose a quadratization, so that PQCR gives the same continuous relaxation bound value as the corresponding Lassere's bound.

Proposition 4.3.1. *For each order r of the Lasserre hierarchy, there exists a quadratization $\mathcal{E}_r$ of PQCR where every product up to degree r is reformulated by a new variable, such that $v(SDP_{\mathcal{E}_r}) = v(M_r)$. Then the sequence of semidefinite programs $(SDP_{\mathcal{E}_r})$, $\lceil \frac{d}{2} \rceil \leq r \leq n$ defines a hierarchy of relaxations.*

Proof. By applying Observation 4.3.1 for each order r, the constraints of $(SDP_{\mathcal{E}_r})$ ensures that the variable matrix is the moment matrix of order r. □

Thus, using a well-chosen sequence of quadratizations, we can make a correspondence between PQCR and the Moment/SOS method for the special case of unconstrained binary polynomial optimization in terms of bounds. However, there is a major difference between the two approaches. On the one hand, given an order and its associated semidefinite relaxation, the Lasserre's hierarchy computes other semidefinite relaxations of higher orders until it reaches the optimal value of (P). On the other hand, given a quadratization and its associated semidefinite relaxation, PQCR builds a single quadratic convex reformulation that is solved by a branch-and-bound algorithm. One of the consequences of Proposition 4.3.1 is that, it is possible to combine the two approaches. More precisely, one can stop the Lasserre's hierarchy at a given order r and derive a quadratic convex reformulation from the associated relaxation that will be solved by branch-and-bound. This is stated formally in the following Corollary.

Corollary 4.3.1. *At each order $\lceil \frac{d}{2} \rceil \leq r \leq n$ one can compute an exact equivalent quadratic convex reformulation to (P) from the associate semidefinite relaxation.*

This Corollary is important in terms of tractability. Indeed, instead of running the hierarchy until optimality, one can choose the order r that has the best trade off between bound quality and tractability. From this order, we then solve the obtained quadratic convex reformulation by branch-and-bound.

In light of these results we can draw a hierarchy of quadratic convex reformulations based on the Moment/SOS hierarchy. The obtained sequence of quadratic convex programs yields increasing continuous relaxation bound values. The convergence results of [51] implies also that there exists a quadratic convex reformulation which continuous relaxation bound value is equal to the optimal value of (P). Indeed, let $\mathcal{E}_n$ be the quadratization that reformulates every product up to degree n. Then the corresponding semidefinite program (and thus the continuous relaxation of the corresponding quadratic convex program) has the same optimal value as (P). However such a quadratization is intractable in general as it requires $\mathcal{O}(2^n)$ variables.

Another important remark is that `PQCR` can improve its tractability for binary polynomial optimization as it allows *non-integer orders* (or *partial orders*) for the relaxation. We precise this claim in the following definition

Definition 4.3.1. *A non-integer order r is such that $d-1 < r < d$ for a given degree d. It corresponds to a quadratization reformulating every product of degree $d-1$ but only a subset of the products of degree d.*

For example, the quadratization we used in Algorithm 4.2.1 leads to a semidefinite relaxation of order $\lceil \frac{d}{2} \rceil - 1 \leq r \leq \lceil \frac{d}{2} \rceil$. Indeed, we only introduced a subset of new variables representing the products of degree $\lceil \frac{d}{2} \rceil$. In addition to the standard integer orders of the Moment/SOS hierarchy, `PQCR` can also compute relaxation of partial orders leading to more tractable quadratic convex reformulations.

Finally, and to justify this study, we show an example where the Lasserre's bound is attained with less variables than required. In other words, all the additional variables do not help improving the bound value.

Example 4 (continued). *We first apply `PQCR` with the partial quadratization defined in Section 4.2.2. This quadratization introduces 6 additional variables $x_6 = x_1x_2$, $x7 = x_1x_3$, $x8 = x_1x_4$, $x_9 = x_2x_3$, $x_{10} = x_2x_4$ and $x_{11} = x_3x_4$. The continuous relaxation bound value after convexification is -2 which is the optimal value of $(Ex2)$.*

We then apply the Lasserre's hierarchy to $(Ex2)$. The first iteration corresponds to the order $\lceil \frac{d}{2} \rceil = 2$. The corresponding relaxation has a 16×16 moment matrix since it amounts to add the variables $x_{12} = x_1x_5$, $x_{13} = x_2x_5$, $x_{14} = x_3x_5$ and $x_{15} = x_4x_5$ to the previous set of additional variables. The optimal value is also reached after the first relaxation of the hierarchy.

This example shows the necessity of choosing a quadratization that suits the studied problem. Indeed, for large problems, it is not necessary to introduce all the additional variables corresponding to a given order r as it can be huge. Instead introducing only the "most useful" variables as it is done in Algorithm 4.2.1 can help with tractability issues.

To conclude, we have shown that, from a given order of the Lasserre's hierarchy, PQCR is able to compute an equivalent quadratic convex reformulation to (P). One of the main advantages of this method is that, unlike the moment/SOS hierarchy, we can avoid solving semidefinite programs of increasing size to compute the optimal value of (P). Indeed, by choosing an arbitrary order r, we can compute the optimal value of (P) by building a quadratic convex reformulation and then running a branch and bound based on this reformulation. Moreover, it is also possible to build a quadratic convex reformulation that does not correspond to any integer order of the hierarchy. Such a reformulation can be useful when the first order of the Lasserre's hierarchy is still not tractable.

4.4 Conclusion

In this Chapter we introduced our main quadratic convex reformulation framework to solve (P). In Section 4.1, we consider the general problem (P) of minimizing a polynomial function where the variables are binary. In this section, we present PQCR, a solution approach for (P). PQCR can be split in 3 phases. We called the first phase *quadratization*, where we rewrite (P) as an equivalent quadratic program (QP). For this, we have to add new variables and linear constraints. We get a linearly constrained quadratic program that still has a non-convex objective function and binary variables. Moreover, even for small instances of (P), the existing convexification methods failed to solve the associate (QP). This is why, we present a family of tailored quadratic convex reformulations of (QP) that exploits its specific structure. For this, we introduce new valid quadratic equalities that vanish on the feasible domain of (QP). We use these equalities to build a family of equivalent quadratic convex formulations to (QP). Then, we focus on finding, within this family, the equivalent convex formulation that maximizes the continuous relaxation bound value. We show that we can compute this "best" convex reformulation using a new semidefinite relaxation of (QP). Finally, we solve our optimal reformulation with a standard solver.

The convexification phase in PQCR gives the tightest bound with respect to the quadratization phase. It means that for a given quadratization, PQCR gives the best possible bound after convexification within this framework, but other quadratizations can give better bounds after convexification. This is why we discuss about quadratization in Section 4.2. We first prove that, given a quadratization $\mathcal{E}$, all possible reformulations of the objective function using $\mathcal{E}$ lead to the same bound after convexification. Then, we prove that adding more additional variables to a given quadratization can improve the bound. We then present two special cases of quadratization. The full quadratization provides a link between PQCR and the Lasserre's hierarchy and the partial quadratization which can be used for computational experiments as it is more tractable than the full quadratization. We end this section by experimentally comparing several pairwise covers within the PQCR framework.

Finally, in Section 4.3 we present the link between PQCR and the Lasserre's hierarchy. More

precisely we prove that applying PQCR with the full quadratization gives the same bound as the first order of the hierarchy. Then, we generalize this result by proving that, for each order of the hierarchy, we can build an equivalent quadratic convex program with the same bound as the corresponding relaxation. The advantage of PQCR lies in the fact that at each order, it is possible to solve a quadratic convex reformulation of (P) with a branch-and-bound instead of continuing the hierarchy and computing bigger relaxations. Finally, we also show that PQCR is more tractable for unconstrained binary polynomial problems than the Lasserre hierarchy. Indeed, we are not forced to add all the auxiliary variables corresponding to a given order. Instead, we can choose a subset of additional variables that will be used in the quadratic reformulation, which amounts to considering a non-integer order relaxation.

Chapter 5

Convexification with other quadratization schemes

Contents

5.1 "Rosenberg-like" quadratizations 94

5.1.1 Applying diagonal convexification after Rosenberg's procedure 94

5.1.2 Using Rosenberg's procedure in a new compact convexification framework 97

5.2 Applying QCR after quadratizations using pairwise covers 101

5.3 Applying QCR after termwise quadratization 103

5.4 Conclusion 105

In the last chapter we have presented our algorithm PQCR, which solves unconstrained binary polynomial programs through quadratic convex relaxation. We have seen that, given a quadratization, we are able to compute a convexification that gives the tightest bound. We consider in this section different state-of-the-art quadratizations that do not add constraints to the reformulation. Further, we study their interactions with a suited convexification phase. For each quadratization, we build an equivalent convex reformulation that keep the "constraintless" property. Moreover, we also present an additional tailored quadratic convex reformulation in the spirit of method PQCR , that losts its "constraintless" property. We finally analyze their performance through computational experiments. We will be interested in two quadratizations using pairwise covers, namely Rosenberg's procedure [72] and the quadratization of Anthony and al. defined in [5], and by the termwise quadratization introduced in [15].

The two first quadratizations depend on a pairwise cover. This means that the quadratizations are applied after the choice of a set $\mathcal{E}$. Once this quadratization scheme is chosen, the two

quadratizations introduce different objective functions in which the link between initial and additional variables is removed. This is the main difference with our quadratization framework, where the Fortet's constraints ensures a relation between the initial and the additional variables. As we make no distinction between these families of variables, we cannot apply our convexification using the valid equalities $S_{\mathcal{E}}$ and we must derive suited convex reformulations instead.

5.1 "Rosenberg-like" quadratizations

We begin by building a first suited convexification phase for Rosenberg's procedure and we discuss about the obtained results. Then we improve our algorithm by introducing a strengthened convex reformulation. We finally compare the results of this new algorithm with the results of the `PQCR` method.

5.1.1 Applying diagonal convexification after Rosenberg's procedure

We apply a diagonal convex reformulation method to "Rosenberg-like" quadratizations. As mentioned in Section 2.3, Rosenberg's procedure [72] is based on a perturbation of the polynomial objective function by adding variables but without adding constraints. This procedure was the first to use penalties to compute equivalent quadratic reformulation to pseudo-Boolean optimization problems. This method was since refined by numerous works including [5]. However, here we only study the results of convexification on the original Rosenberg's quadratization since the study can be easily generalized to other similar quadratizations.

Given a set $\mathcal{E}$ and a quadratic decomposition of the objective function $f^r_{\mathcal{E}}$, we can build the following equivalent problem to (QP)

$$(QP_P)\begin{cases} \min g_P(x,y) = f^r_{\mathcal{E}}(x,y) + \sum_{i,j} P(x_ix_j - 2x_iy_{ij} - 2x_jy_{ij} + 3y_{ij}) \\ \text{s.t.} \\ \quad x \in \{0,1\}^n \\ \quad y \in \{0,1\}^{N_P - n} \end{cases}$$

where P is a large positive number (penalty term) and N_P is the total number of variables (initial and auxiliary) used in the quadratization.

Let us remark that, in this quadratization, y_{ij} represents the product x_ix_j in all the optimal solutions. One idea is to apply the convexification phase of `PQCR` by adding the null functions deduced from the set $S_{\mathcal{E}}$, defined in Lemma 4.1.1, to the objective function and we get the resulting program $(QP_{P_{\alpha,\beta,\delta,\lambda}})$. Unfortunately, the link between variables x and y in this quadratization is lost and only the constraints $x_i^2 = x_i$ of the set $S_{\mathcal{E}}$ are still valid in (QP_P). As a consequence, to enforce the equivalence between $(QP_{P_{\alpha,\beta,\delta,\lambda}})$ and (QP_P), we need to add the Fortet's inequalities

$\mathcal{F}_{\mathcal{E}}$, defined in Section 4.1.1, to both program. We precise the relation between the different programs below

$$(QP) \iff (QP_P) \not\Longleftrightarrow (QP_{P_{\alpha,\beta,\delta,\lambda}})$$

$$(QP) \iff (QP_P) + \mathcal{F}_{\mathcal{E}} \iff (QP_{P_{\alpha,\beta,\delta,\lambda}}) + \mathcal{F}_{\mathcal{E}}$$

However, the whole purpose of Rosenberg's quadratization is to build a quadratic reformulation without any additional constraint. In this context, it is not possible to build an equivalent program based on the set $S_{\mathcal{E}}$ without the linearization constraints. The following example illustrates this remark.

Example 2 (continued). *By introducing two additional variables x_5 and x_6 that represent the products x_1x_2 and x_3x_4 respectively, we can write two parameterized programs: $(QP_{P_{\alpha^*,\beta^*,\delta^*,\lambda^*}})$ which is equivalent to (QP_P) through the linearization constraints*

$$(QP_{P_{\alpha^*,\beta^*,\delta^*,\lambda^*}}) \begin{cases} \min\limits_{x\in\{0,1\}^6} g_{P_{\alpha^*,\beta^*,\delta^*,\lambda^*}}(x) \\ s.t. \\ x_5 - x_1 \leq 0 \\ x_5 - x_2 \leq 0 \\ -x_5 + x_1 + x_2 \leq 1 \\ x_6 - x_3 \leq 0 \\ x_6 - x_4 \leq 0 \\ -x_6 + x_3 + x_4 \leq 1 \\ \quad x \in \{0,1\}^6 \end{cases}$$

and $(RP_{P_{\alpha^*,\beta^*,\delta^*,\lambda^*}})$ which does not contain the linearization constraints.

$$(RP_{P_{\alpha^*,\beta^*,\delta^*,\lambda^*}}) \begin{cases} \min\limits_{x\in\{0,1\}^6} g_{P_{\alpha^*,\beta^*,\delta^*,\lambda^*}}(x) \\ s.t. \\ \quad x \in \{0,1\}^6 \end{cases}$$

The optimal value of $(QP_{P_{\alpha^*,\beta^*,\delta^*,\lambda^*}})$ is 0 whereas the optimal value of $(RP_{P_{\alpha^*,\beta^*,\delta^*,\lambda^*}})$ is -0.125.

In fact, applying our convexification after Rosenberg's procedure decreases the "big M" value and thus the equivalence does not hold anymore. The quadratization structure using penalty terms without constraints is preserved if the convexification uses only diagonal perturbation. This can be explained as the convexification does not modify the Hessian matrix on the same entries as the penalty terms used in Rosenberg's procedure. Indeed, the convexification using $x_i^2 = x_i$ modifies diagonal entries of the Hessian matrix whereas the penalty terms modify only

non-diagonal terms as there are no squared variables inside the penalty terms. Thus, as $x \mapsto x^2 - x$ is a null function over $\{0,1\}^{N_P}$, we can build the following parameterized equivalent program to (QP_P).

$$(QP_{P_\alpha}) \begin{cases} \min g_P(x) + \sum_{i \in I \cup J} \alpha_{P_i}(x_i^2 - x_i) \\ \text{s.t.} \\ \quad x \in \{0,1\}^{N_P} \end{cases}$$

Let us remark that this approach is exactly equivalent to apply the `QCR` [13] method to (QP_P). Following this method, we know that an optimal value of vector α_{P_i} corresponds to the optimal dual variables of constraints (5.1) of the following semi-definite relaxation:

$$(SDP_P) \begin{cases} \min g_P(x, X) \\ \text{s.t.} \\ \quad X_{ii} - x_i = 0 \qquad\qquad i \in I \cup J & (5.1) \\ \quad \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 & (5.2) \\ \quad x \in \mathbb{R}^{N_P},\ X \in S^{N_P} & (5.3) \end{cases}$$

We further provide computational experiments obtained by this approach, called `ROS+QCR`, on small LABS instances (1.2.2) and on Vision instances (1.2.1) in Chapter 7. In particular, we remark that applying the `QCR` method after Rosenberg's procedure is not an efficient method at all, as large positive values on non-diagonal terms lead to large positive values for α. As a result the continuous relaxation bound obtained is very weak.

Example 2 (continued). *We recall the optimization problem we want to solve*

$$(Ex) \left\{ \min_{x \in \{0,1\}^4} 2x_1 + 3x_2x_3 - 2x_2x_3x_4 - 3x_1x_2x_3x_4 \right.$$

For this example, we introduce two additional variables for the quadratization, namely $x_5 = x_1x_2$ *and* $x_6 = x_3x_4$*. The penalty parameter* P *is set to* 5*. Applying the* `QCR` *method, we get* $\alpha_P = (4.3, 3.9, 4, 4, 10, 9)$*. The equivalent program* (QP_{P_α}) *is written as follows*

$$
(QP_{P_\alpha})\begin{cases}
\min g_{P_\alpha}(x) = 2x_1 + 3x_2x_3 - 2x_2x_6 - 3x_5x_6 + 5(x_3x_4 - 2x_3x_6 - 2x_4x_6 + 3x_6) \\
+5(x_1x_2 - 2x_1x_5 - 2x_2x_5 + 3x_5) + 2.1(x_1^2 - x_1) + 1.9(x_2^2 - x_2) + 2(x_3^2 - x_3) \\
+2(x_4^2 - x_4) + 5(x_5^2 - x_5) + 4.5(x_6^2 - x_6) \\
s.t. \\
\quad x \in \{0,1\}^6
\end{cases}
$$

The Hessian matrix of g_{P_α} after the perturbation is

$$
Q_{P_\alpha} = \begin{pmatrix}
4.3 & 2.5 & 0 & 0 & -5 & 0 \\
2.5 & 3.9 & 1.5 & 0 & -5 & -1 \\
0 & 1.5 & 4 & 2.5 & 0 & -5 \\
0 & 0 & 2.5 & 4 & 0 & -5 \\
-5 & -5 & 0 & 0 & 10 & -1.5 \\
0 & -1 & -5 & -5 & -1.5 & 9
\end{pmatrix}
$$

Degree	*Method*	*Continuous relaxation bound value*
1	`Standard linearization`	-1.5
d	`Uniform interval`	-5.3
d	`Non-uniform interval`	-4.3
2	`Smallest eigenvalue`	-1.7
2	`QCR`	-1.6
2	`PQCR`	-0.6
2	`ROS+QCR`	-2

In the next paragraph, we study a smarter way to include Rosenberg's procedure in our convexification step.

5.1.2 Using Rosenberg's procedure in a new compact convexification framework

In this section, we build a new compact convexification phase that comes directly from the Rosenberg's procedure.

Let x_k be the additional variable representing the product x_ix_j. The penalty term $P(x_ix_j - 2x_ix_k - 2x_jx_k + 3x_k)$ induces a valid equality, which was proven in [72]. We formally state this valid equality as the following set of constraints.

$$(R_{\mathcal{E}}) \left\{ \; x_i x_j - 2x_i x_k - 2x_j x_k + 3x_k = 0, \; (i,j,k) \in (I \cup J)^2 \times J : \; \mathcal{E}_k = \mathcal{E}_i \cup \mathcal{E}_j \right. \tag{5.4}$$

Let us observe that, for a triplet $(x_i, x_j, x_k) \in \{0,1\}^3$, $x_i x_j - 2x_i x_k - 2x_j x_k + 3x_k = 0$ if and only if $x_k = x_i x_j$.

Proposition 5.1.1. *Let* $(x_i, x_j, x_k) \in \{0,1\}^3$*, then* $x_i x_j - 2x_i x_k - 2x_j x_k + 3x_k = 0 \iff x_k = x_i x_j$.

Proof.

- ($\Leftarrow$) We have $x_k = x_i x_j$
  - If $x_k = 0$ then $x_i x_j = 0$ and thus the equality $x_k = x_i x_j$ holds.
  - If $x_k = 1$ then $x_i x_j - 2x_i - 2x_j = 3$ which implies $x_i = 1$ and $x_j = 1$ and the equality $x_k = x_i x_j$ holds again.
- ($\Rightarrow$) We have $x_i x_j - 2x_i x_k - 2x_j x_k + 3x_k = 0$
  - if $x_i = 0$ (respectively $x_j = 0$) then $-2x_j x_k + 3x_k = 0$ (respectively $-2x_i x_k + 3x_k = 0$) which leads to $x_k = 0$.
  - if $x_i = 1$ and $x_j = 1$ then $x_k = 1$

□

One of the consequences of this proposition is that, in the initial set of valid equalities $S_{\mathcal{E}}$, we can swap the constraints $y_{ij} = x_i x_j$ (4.10) with the constraints $R_{\mathcal{E}}$ (5.4) without changing the set of feasible solutions. Moreover, it is clear that all the points satisfying $S_{\mathcal{E}}$ also satisfy $R_{\mathcal{E}}$ and we can compute a compact convex reformulation using valid equalities $R_{\mathcal{E}}$ only. More precisely, given a set $\mathcal{E}$ and a quadratic decomposition of the objective function $f^r_{\mathcal{E}}$, we can build a new equivalent quadratic program to (P):

$$(QP_R) \begin{cases} \min f^r_{\mathcal{E}}(x) \\ \text{s.t.} \\ \quad x_i x_j - 2x_i x_k - 2x_j x_k + 3x_k = 0 \qquad (i,j,k) \in (I \cup J)^2 \times J : \; \mathcal{E}_k = \mathcal{E}_i \cup \mathcal{E}_j \\ \quad x \in \{0,1\}^{N_R} \end{cases}$$

where N_R is the total number of variables used in the quadratization.

We now use these quadratic constraints to compute a convexification of the objective function. More precisely, we introduce the following equivalent program to (QP) parameterized by α and λ.

$$(QP_{R_{\alpha,\lambda}}) \begin{cases} \min g_{R_{\alpha,\lambda}}(x) = f^r_{\mathcal{E}}(x) + \sum_{i \in I \cup J} \alpha_{R_i}(x_i^2 - x_i) + \sum_{\substack{(i,j,k) \in (I \cup J)^2 \times J \\ \mathcal{E}_k = \mathcal{E}_i \cup \mathcal{E}_j}} \lambda_{R_{ij}}(x_i x_j - 2x_i x_k - 2x_j x_k + 3x_k) \\ \text{s.t.} \\ \quad x \in \mathcal{F}_{\mathcal{E}} \end{cases}$$

Note here that we need to add the set $\mathcal{F}_{\mathcal{E}}$ to ensure the equivalence between (QP_R) and $(QP_{R_{\alpha,\lambda}})$. An advantage of constraints $x_i x_j - 2x_i x_k - 2x_j x_k + 3x_k = 0$ when compared to constraints $x_i x_j = y_{ij}$, is that their dualization perturbs more entries of the Hessian matrix. This can therefore lead to a more refined convexification in terms of continuous relaxation bound value. An experimental illustration of this impact is presented in Chapter 7. The results confirm that using $x_i x_j - 2x_i x_k - 2x_j x_k + 3x_k = 0$ leads to tighter bounds than when using $x_i x_j = y_{ij}$. Indeed, the resolution method based on $(QP_{R_{\alpha,\lambda}})$ is able to solve 10 more instances.

We now prove that the "best" continuous relaxation bound value of (QP_R), such that $g_{R_{\alpha,\lambda}}$ is convex, is equal to the optimal value of a semi-definite program. More precisely, let (CP_R) be the following program

$$(CP_R) : \max_{\substack{\alpha \in \mathbb{R}^{N_R}, \lambda \in \mathbb{R}^{N_R - n} \\ Q_{R_{\alpha,\lambda}} \succeq 0}} \left\{ \min_{x \in \overline{\mathcal{F}}_{\mathcal{E}}} \; g_{R_{\alpha,\lambda}}(x) \right\}$$

Where $Q_{R_{\alpha,\lambda}}$ is the Hessian matrix of $g_{R_{\alpha,\lambda}}$. We claim that the optimal value of (CP_R) is equal to the optimal value of a semi-definite program (SDP_R). Moreover, we can also deduce optimal values of α and λ from (SDP_R).

Corollary 5.1.1. *The optimal value of (CP_R) is equal to the optimal value of the following semi-definite program (SDP_R):*

$$(SDP_R) \begin{cases} \min < Q, X > + c^T x \\ s.t. \\ \quad X_{ii} - x_i = 0 & i \in I \cup J & (5.5) \\ \quad X_{ij} - 2X_{ik} - 2X_{jk} + 3x_k = 0 & (i,j,k) \in (I \cup J)^2 \times J : \; \mathcal{E}_k = \mathcal{E}_i \cup \mathcal{E}_j & (5.6) \\ \quad \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 & & (5.7) \\ \quad x \in \overline{\mathcal{F}}_{\mathcal{E}} & & (5.8) \\ \quad x \in \mathbb{R}^{N_R}, \; X \in S^{N_R} & & (5.9) \end{cases}$$

The optimal values (α^*, λ^*) of problem (CP_R) are given by the optimal values of the dual variables associated with constraints (5.5) and (5.6) respectively.

Proof. Straightforward from the proof of Theorem 4.1.2 as constraint (5.6) is a linear combination of constraints (4.9) and (4.10). □

This convexification, called `PQCR ROS`, is thus more compact than the one used in `PQCR` as it uses far less constraints. We illustrate the resulting convex reformulation in the following example.

Example 2 (continued). *We recall the optimization problem we want to solve*

$$(Ex)\left\{ \min_{x\in\{0,1\}^4} 2x_1 + 3x_2x_3 - 2x_2x_3x_4 - 3x_1x_2x_3x_4 \right.$$

For this example, we introduce two additional variables for the quadratization, namely $x_5 = x_1x_2$ *and* $x_6 = x_3x_4$*. We get* $\alpha_R = (0.3, 1.6, 2.1, 2.2, 2.1, 4.3)$ *and* $\lambda_R = (0.3, 2.2)$*. The equivalent program* $(QP_{R_{\alpha,\lambda}})$ *is written as follows*

$$(QP_{R_{\alpha,\lambda}})\begin{cases} \min g_{R_{\alpha,\lambda}}(x) = 2x_1 + 3x_2x_3 - 2x_2x_6 - 3x_5x_6 + \sum_{i=1}^{N} \frac{1}{2}\alpha_{R_i}(x_i^2 - x_i) \\ +2.2(x_3x_4 - 2x_3x_6 - 2x_4x_6 + 3x_6) + 0.3(x_1x_2 - 2x_1x_5 - 2x_2x_5 + 3x_5) \\ s.t. \\ \quad x \in \{0,1\}^6 \end{cases}$$

The Hessian matrix of $g_{R_{\alpha,\lambda}}$ *after the perturbation is*

$$Q_{R_{\alpha,\lambda}} = \begin{pmatrix} 0.3 & 0.1 & 0 & 0 & -0.3 & 0 \\ 0.1 & 1.6 & 1.5 & 0 & -0.3 & -1 \\ 0 & 1.5 & 2.1 & 1.1 & 0 & -2.2 \\ 0 & 0 & 1.1 & 2.2 & 0 & -2.2 \\ -0.3 & -0.3 & 0 & 0 & 2.1 & -1.5 \\ 0 & -1 & -2.2 & -2.2 & -1.5 & 4.3 \end{pmatrix}$$

Degree	*Method*	*Continuous relaxation bound value*
1	`Standard linearization`	-1.5
d	`Uniform interval`	-5.3
d	`Non-uniform interval`	-4.3
2	`Smallest eigenvalue`	-1.7
2	`QCR`	-1.6
2	`PQCR`	-0.6
2	`ROS+QCR`	-2
2	`PQCR ROS`	-1

Even if the continuous relaxation bound value of $(QP_{R_{\alpha^*,\lambda^*}})$ is smaller or equal to the one of `PQCR`, `PQCR ROS` can be faster than `PQCR`. We present and comment experimental results in Tables 7.5-7.10.

5.2 Applying QCR after quadratizations using pairwise covers

Here we are interested in quadratizations using pairwise covers. We build a suited convexification phase for a specific quadratization family. The concept of pairwise covers was introduced in [5]. It consists in choosing a decomposition of product of initial variables by a new one. Several choices are possible depending on the criterion we consider as it is shown in Section 4.2. Once a pairwise cover is defined, the authors of [5] build a quadratization. As detailed in Section 2.3, the concept of quadratization is different from our work and we renamed it *quadratization without additional constraint*. Nevertheless, this quadratization, like Rosenberg's procedure, introduces additional monomials to the objective function to obtain a quadratic function that is equal to f on all the optimal points. This quadratization of [5] is described below.

Let $\mathcal{M}$ be the set of all the monomials. Let $\mathcal{H} \subset \mathcal{M}$ be a pairwise cover of $\mathcal{M}$, that is a set of products of variables that will be reformulated by a new one. Thus a monomial M covered by the set of variables $A(M)$ and $B(M)$ (i.e. $m = A(m) \cup B(M)$) will be quadratically reformulated by $y_{A(M)}y_{B(M)}$. By doing so for all the monomials, Anthony et al. compute a quadratization using $|\mathcal{H}|$ auxiliary variables y_i leading to a quadratic reformulation with $N = n + |\mathcal{H}|$ variables. Thus, every pseudo-Boolean function of the form $f(x) = \sum_{M\in\mathcal{M}} a_M \prod_{j\in M} x_j$ is such that

$$f(x) = \min_{y\in\{0,1\}^{|\mathcal{H}|}} f_{PC}(x, y)$$

where

$$f_{PC}(x, y) = \sum_{M\in\mathcal{M}} a_M y_{A(M)} y_{B(M)} + \sum_{H\in\mathcal{H}} b_H \left(y_H \left(|H| - \frac{1}{2} - \sum_{j\in H} x_j \right) + \frac{1}{2} \prod_{j\in H} x_j \right)$$

and $b_H = 0$ for $H \in \mathcal{M}\backslash\mathcal{H}$ and for $H \in \mathcal{H}$ we have

$$\frac{1}{2} b_H = \sum_{\substack{M\in\mathcal{M}: \\ H\in\{A(M),B(M)\}}} \left(|a_M| + \frac{1}{2} b_M \right),$$

Note that the second term of function f_{PC} can be seen as a penalty term as it enforces the equality with function f on all the optimal points. Another remark is that the term $\frac{1}{2}\prod_{j\in H} x_j$ is always quadratic, as the assumption $\mathcal{H} \subset \mathcal{M}$ requires to make recursive substitution until $\mathcal{H}$ is of cardinality 2. The property $\mathcal{H} \subset \mathcal{M}$ is thus equivalent to the *reducibility* property introduced in [21].

For the sake of clarity, we can rewrite $f_{PC}(x, y)$ as $f_{PC}(x)$ where x is a $N-$dimensional vector. Section 5.1 concerning Rosenberg's quadratization made clear that, in the case of quadratization using penalty terms in the objective function, it was not possible to build a convexification without

altering the quadratization itself. The same holds for this quadratization. Moreover, the equality $y^* = \prod_i x_i^*$ does not generally hold on the optimal points (x^*, y^*) of f_{PC}. Consequently, it is difficult to derive new valid equalities to link the variables between them in this framework. However, since f_{PC} do not contain square terms, it is possible to convexify it by using only the equality $x_i^2 = x_i$, since this equality is valid for any $x \in \{0,1\}^n$. More precisely, we consider the following equivalent program to (QP) in $\{0,1\}^{N_{PC}}$ and parameterized by α:

$$(QP_{PC_\alpha}) \begin{cases} \min g_{PC_\alpha}(x) = f_{PC}(x) + \sum_{i \in I \cup J} \alpha_{PC_i}(x_i^2 - x_i) \\ \text{s.t.} \\ \quad x \in \{0,1\}^{N_{PC}} \end{cases}$$

where N_{PC} is the total number of variables used in the quadratization.

Note that this is again equivalent to apply the `QCR` method to an already quadratic program. Similarly to what we have done with Rosenberg's procedure, we can retrieve an optimal value of vector α_{PC} from a semi-definite relaxation of (QP_{PC_α}). We call this method `PC+QCR`.

Example 2 (continued). *We consider the same example as previously with the same pairwise cover. For this example, the resolution of the semi-definite relaxation leads to* $\alpha_1^* = 3.3$, $\alpha_2^* = 2.9$, $\alpha_3^* = 4$, $\alpha_4^* = 4$, $\alpha_5^* = 6.1$ *and* $\alpha_6^* = 9$. *The equivalent program* (QP_{PC_α}) *is written as follows*

$$(QP_{PC_\alpha}) \begin{cases} \min g_{PC_\alpha}(x) = \min 2x_1 + 3x_2x_3 - 2x_6x_2 - 3x_5x_6 \\ +10\left[y_2(2 - \frac{1}{2} - x_3 - x_4) + \frac{1}{2}x_3x_4\right] \\ +6\left[y_1(2 - \frac{1}{2} - x_1 - x_2) + \frac{1}{2}x_1x_2\right] \\ +1.6(x_1^2 - x_1) + 1.4(x_2^2 - x_2) + 2(x_3^2 - x_3) \\ +2(x_4^2 - x_4) + 3(x_5^2 - x_5) + 4.5(x_6^2 - x_6) \\ s.t. \\ \quad x \in \{0,1\}^6 \end{cases}$$

The Hessian matrix of g_{PC_α} *after the diagonal perturbation is*

$$Q_{PC_\alpha} = \begin{pmatrix} 3.3 & 0 & 0 & 0 & -3 & 0 \\ 0 & 2.9 & 1.5 & 0 & -3 & -1 \\ 0 & 1.5 & 4 & 0 & 0 & -5 \\ 0 & 0 & 0 & 4 & 0 & -5 \\ -3 & -3 & 0 & 0 & 6.1 & -1.5 \\ 0 & -1 & -5 & -5 & -1.5 & 9 \end{pmatrix}$$

Degree	*Method*	*Continuous relaxation bound value*
1	`Standard linearization`	-1.5
d	`Uniform interval`	-5.3
d	`Non-uniform interval`	-4.3
2	`Smallest eigenvalue`	-1.7
2	`QCR`	-1.6
2	`PQCR`	-0.6
2	`ROS+QCR`	-2
2	`PQCR ROS`	-1
2	`PC+QCR`	-1.7

We have built a convexification based on pairwise covers and their associated quadratization. Before presenting the experimental results, we introduce a convexification phase for termwise quadratization.

5.3 Applying `QCR` after termwise quadratization

To conclude our comparative study on convexifications using other quadratization schemes, we present a last convexification using termwise quadratization. As mentioned in Section 2.3, an important result was stated in [15] showing that every monomial with positive coefficient can be quadratically reformulated using $\lceil log(n) \rceil - 1$ variables.

We denote by f_T the new quadratic function obtained by applying this quadratization. We recall the construction of f_T explained in Section 2.3. First, f_T contains all the monomials of f of degree less or equal than 2. Let $m(x) = \alpha \prod_{x=1}^{d_m} x_i$ be a monomial of degree at least 3 of f.

If α is negative, the monomial $m(x)$ is replaced in f_T by

$$g_m(x, y) = -\alpha \left[(d_m - 1)y - \sum_{i=1}^{d_m} x_i y \right],$$

where $y \in [0, 1]$ is a single auxiliary variable. If α is positive, then we introduce $\lceil log(d_m) \rceil - 1$ auxiliary variables y_i and we replace $m(x)$ in f_T by

$$g_m(x, y) = \frac{\alpha}{2}(|x| + 2^l - d_m - \sum_{i=1}^{l-1} 2^i y_i)(|x| + 2^l - d_m - \sum_{i=1}^{l-1} 2^i y_i - 1)$$

where $l = \lceil log(d_m) \rceil$.

We thus obtain the following equivalent program to (P)

$$(QP_T)\begin{cases} \min f_T(x,y) = \sum\limits_{\substack{\text{monomials } m \\ \text{of degree } \leq 2}} m(x) + \sum\limits_{\substack{\text{monomials } m \\ \text{of degree } \geq 3}} g_m(x,y) \\ \text{s.t.} \\ \quad x \in \{0,1\}^{N_T} \end{cases}$$

where N_T is the total number of variables used in the quadratization.

Again the difficulty here lies in the fact that it is not possible to derive simple valid equalities involving both initial and additional variables. Thus, the only family of valid equalities from $S_{\mathcal{E}}$ that we can use is $x_i^2 = x_i$ which amounts to apply the regular `QCR` method. We can thus build the following new equivalent quadratic program to (QP_T) parameterized by α.

$$(QP_{T_\alpha})\begin{cases} \min g_{T_\alpha}(x) = g(x) + \sum\limits_{i \in I \cup J} \alpha_{T_i}(x_i^2 - x_i) \\ \text{s.t.} \\ \quad x \in \{0,1\}^{N_T} \end{cases}$$

As we have done previously, we can compute an optimal value of α_T, i.e. the value of α that maximizes the continuous relaxation of (QP_{T_α}) and that makes g_{T_α} convex. We call this approach `T+QCR`.

Example 2 (continued). *We consider the same example as previously and we introduce two additional variables for the termwise quadratization, $y_1 = x_2x_3x_4$ and $y_2 = x_1x_2x_3x_4$. For this example, the resolution of the semi-definite relaxation leads to $\alpha_1^* = 1.8$, $\alpha_2^* = 1.5$, $\alpha_3^* = 1.5$, $\alpha_4^* = 2.9$, $\alpha_5^* = 3.5$ and $\alpha_6^* = 3.5$. The equivalent program (QP_{T_α}) is written as follows*

$$(QP_{T_\alpha})\begin{cases} \min g_{T_\alpha}(x) = \min 2x_1 + 3x_2x_3 \\ +3\left[2y_1 - x_2y_1 - x_3y_1 - x_4y_1\right] \\ +2\left[3y_2 - x_1y_2 - x_2y_2 - x_3y_2 - x_4y_2\right] \\ +0.9(x_1^2 - x_1) + 0.7(x_2^2 - x_2) + 0.7(x_3^2 - x_3) \\ +1.4(x_4^2 - x_4) + 1.7(x_5^2 - x_5) + 1.7(x_6^2 - x_6) \\ s.t. \\ \quad x \in \{0,1\}^4 \\ \quad y \in \{0,1\}^2 \end{cases}$$

The Hessian matrix of g_{T_α} after the diagonal perturbation is

$$Q_{T_\alpha} = \begin{pmatrix} 1.8 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1.5 & 1.5 & 0 & -1.5 & -1 \\ 0 & 1.5 & 1.5 & 0 & -1.5 & -1 \\ 0 & 0 & 0 & 2.9 & -1.5 & -1 \\ 0 & -1.5 & -1.5 & -1.5 & 3.5 & 0 \\ -1 & -1 & -1 & -1 & 0 & 3.5 \end{pmatrix}$$

Degree	*Method*	*Continuous relaxation bound value*
1	*Standard linearization*	-1.5
d	*Uniform interval*	-5.3
d	*Non-uniform interval*	-4.3
2	*Smallest eigenvalue*	-1.7
2	*QCR*	-1.6
2	*PQCR*	-0.6
2	*ROS+QCR*	-2
2	*PQCR ROS*	-1
2	*PC+QCR*	-1.7
2	*T+QCR*	-1.1

A detailed experimental comparisons of this approach is presented in Section 7.4.2.

5.4 Conclusion

This Chapter aimed at showing the behavior of convex reformulation methods when applied to different quadratization frameworks. First, we use Rosenberg's procedure for the first phase of `PQCR`. We show that the quadratization enforced by the penalty term is no longer valid after the convexification. As a result, when applying this kind of quadratizations, the linearization constraints are required, which is not the spirit of Rosenberg's procedure. We thus build a diagonal convexification which amounts to apply the `QCR` method. Then we introduce a new convexification using Rosenberg's penalty term. This leads to a more compact reformulation than `PQCR`. Then, we compute a convexification based on the quadratization defined in [5]. The previous study on Rosenberg's quadratization showed that non-diagonal perturbations can also modify the validity of the quadratization. For this reason, we apply a convexification based on a diagonal perturbation that we can compare with (QP_α). Finally, we derive a diagonal convexification for termwise quadratizations.

Chapter 6

Semi-definite relaxations of box-constrained polynomial programs

Contents

6.1 A quadratic reformulation of $(\bar{P})$ 107

6.2 A compact semidefinite programming relaxation 107

6.3 An improved semidefinite programming relaxation 108

6.4 Conclusion 110

Throughout this dissertation, we discussed binary polynomial optimization. In this chapter, we compute tight bounds for box-constrained polynomial programs using quadratic convex relaxations. One can then use this relaxation at each node of a spatial branch-and-bound to compute an optimal solution to the initial problem.

We consider the box-constrained polynomial optimization problem that can be stated as follows:

$$(\bar{P}) \begin{cases} \min f(x) = \sum_{p=1}^{m} c_p \prod_{i \in \mathcal{M}_p} x_i \\ \text{s.t.} \\ x_i \in [0,1],\ i \in I \end{cases}$$

where $I = \{1, .., n\}$, f is an $n-$variable polynomial of degree d and m is the number of monomials. For a monomial p, $\mathcal{M}_p \subset I$ is a multiset containing the indexes of the variables involved in p. It follows that $d = \max_p |\mathcal{M}_p|$. We suppose that each variable x_i is in the interval $[0,1]$, since variables in any $[\ell_i, u_i]$ interval can be transformed into $[0,1]$ by a simple variable change.

We now introduce new quadratic convex relaxations of $(\bar{P})$. For this, we first reformulate $(\bar{P})$ into an equivalent quadratic program $(\overline{QP}_{\mathcal{E}})$ using the same algorithm as in Chapter 4. We then obtain a quadratically constrained quadratic program. Then, we propose two positive semidefinite relaxations of $(\overline{QP}_{\mathcal{E}})$. We start with the classical compact semidefinite relaxation, and then we strengthen it by addition of valid inequalities that come from the quadratization phase.

6.1 A quadratic reformulation of $(\bar{P})$

In this section, we present how we build equivalent quadratic formulations to $(\bar{P})$. The basic idea is to reduce the degree of f to 2. For this, we use the previously defined notational conventions and Definitions 3.1.1 and 4.1.1.

With these definitions, we reformulate (P) as a non-convex quadratically constrained quadratic program $(\overline{QP}_{\mathcal{E}})$ with N variables:

$$(\overline{QP}_{\mathcal{E}}) \begin{cases} \min g(x) = \sum\limits_{\substack{|\mathcal{M}_p|\geq 3 \\ \mathcal{M}_p=\mathcal{E}_j\cup\mathcal{E}_k}} c_p x_j x_k + \sum\limits_{|\mathcal{M}_p|\leq 2} c_p \prod\limits_{i\in\mathcal{M}_p} x_i \\ \text{s.t.} \\ x_i = x_j x_k,\ (i,j,k)\in J\times(I\cup J)^2:\ \mathcal{E}_i = \mathcal{E}_j\cup\mathcal{E}_k & (6.1) \\ x_i\in[0,1],\ i\in I\cup J & (6.2) \end{cases}$$

By construction, problems $(\bar{P})$ and $(\overline{QP_{\mathcal{E}}})$ are equivalent in the sense that, from any solution of one problem, one can deduce a solution for the other problem with the same objective function value. Let us observe that $(\overline{QP_{\mathcal{E}}})$ is parameterized by the quadratization defined by sets $\mathcal{E}$. Indeed, here again, several valid quadratizations can be applied to $(\bar{P})$, each of them leading to different sets $\mathcal{E}_i$.

For the sake of simplicity, $g(x)$ can be rewritten as $g(x) = \langle Q, xx^T\rangle + c^T x$, where $Q\in S_N$ and $c\in\mathbb{R}^N$.

6.2 A compact semidefinite programming relaxation

Here we build the standard semidefinite relaxation of $(\overline{QP_{\mathcal{E}}})$. Classically, we linearize the products xx^T using a matrix variable $X\in S_N$ and we relax the equality $X = xx^T$ as $X - xx^T \succeq 0$. We obtain the following semidefinite program:

$$
(\overline{SDP}^0_{\mathcal{E}}) \begin{cases} \min < Q, X > + c^T x \\ \text{s.t.} \\ x_i = X_{jk},\ (i,j,k) \in J \times (I \cup J)^2 :\ \mathcal{E}_i = \mathcal{E}_j \cup \mathcal{E}_k & (6.3) \\ X_{ii} \leq x_i,\ i \in I \cup J & (6.4) \\ \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 & (6.5) \\ x \in \mathbb{R}^N,\ X \in S^N & (6.6) \end{cases}
$$

where Constraints (6.3) and (6.4) correspond to the linearization of Constraints (6.1) and (6.2), respectively. By Schur's Lemma, Constraint (6.3) is equivalent to the relaxed constraint $X - xx^T \succeq 0$. It is important to note that the number $N - n$ of Constraints (6.3) depends on the quadratization $\mathcal{E}$ used. $(\overline{SDP}^0_{\mathcal{E}})$ has a reasonable size of $\mathcal{O}(N^2)$ variables and $\mathcal{O}(N)$ constraints.

6.3 An improved semidefinite programming relaxation

We now focus on building a tighter semidefinite relaxation of $(\overline{QP_{\mathcal{E}}})$ by strengthening $(\overline{SDP}^0_{\mathcal{E}})$. We start by adapting the valid quadratic equalities of the set $S_{\mathcal{E}}$ defined in Lemma 4.1.1. We recall that these equalities come from the quadratization $\mathcal{E}$.

More formally, for a quadratization characterized by $\mathcal{E}$, we introduce the following set of constraints that is valid when Constraints (6.1)-(6.2) are satisfied.

Lemma 6.3.1. *The following quadratic equalities and inequalities are valid:*

$$
\begin{cases} x_i x_j \leq x_i,\ (i,j) \in J \times (I \cup J) :\ \mathcal{E}_j \subset \mathcal{E}_i & (6.7) \\ x_i x_j = x_k x_l,\ (i,j,k,l) \in (I \cup J)^4 :\ \mathcal{E}_i \sqcup \mathcal{E}_j = \mathcal{E}_k \sqcup \mathcal{E}_l & (6.8) \end{cases}
$$

Proof. Constraints (6.7) trivially hold since $x_i \in [0,1]$. We then prove the validity of the Constraints (6.8). By definition we have:

$$
\begin{aligned} x_i x_j &= \prod_{i' \in \mathcal{E}_i} x_{i'} \prod_{j' \in \mathcal{E}_j} x_{j'} \\ &= \prod_{i' \in \mathcal{E}_i \sqcup \mathcal{E}_j} x_{i'} \\ &= \prod_{k' \in \mathcal{E}_k \sqcup \mathcal{E}_l} x_{k'} \text{ since } \mathcal{E}_i \sqcup \mathcal{E}_j = \mathcal{E}_k \sqcup \mathcal{E}_l \\ &= x_k x_l \end{aligned}
$$

□

In a sense, Constraints (6.8) can be viewed as constraints that break symmetries. Constraints (6.7) and (6.8) are not convex, but since they are quadratic, we can easily linearize them using the matrix variable X.

Adding these constraints to $(\overline{SDP}^0_{\mathcal{E}})$, we obtain the following semidefinite relaxation:

$$(\overline{SDP}^1_{\mathcal{E}}) \begin{cases} \min < Q, X > + c^T x \\ \text{s.t.} \\ \quad (6.3) - (6.6) \\ \quad X_{ij} \leq x_i, \ (i,j) \in J \times (I \cup J) : \ \mathcal{E}_j \subset \mathcal{E}_i & (6.9) \\ \quad X_{ij} = X_{kl}, \ (i,j,k,l) \in (I \cup J)^4 : \ \mathcal{E}_i \sqcup \mathcal{E}_j = \mathcal{E}_k \sqcup \mathcal{E}_l & (6.10) \end{cases}$$

$(\overline{SDP}^1_{\mathcal{E}})$ is larger than $(\overline{SDP}^0_{\mathcal{E}})$. Indeed, it still has $\mathcal{O}(N^2)$ variables, but there are $\mathcal{O}(N^4)$ constraints. Here again, the number $N(N-n)$ of Constraints (6.9) depends on the quadratization $\mathcal{E}$ used. As for the number of Constraints (6.10), it depends also on the structure of the instance and can be up to N^4.

We further evaluate and compare these two relaxations on random instances. The results, as well as the comments, can be found in Chapter 7. Both of these relaxations can be used in a spatial branch-and-bound based on semi-definite programming. Indeed, at each node we solve $(SDP^0_{\mathcal{E}})$ or $(SDP^1_{\mathcal{E}})$ and we branch on the violation of constraints (6.1), i.e. $x_i = x_j x_k$, to get back the equivalence with the original problem $(\overline{QP}_{\mathcal{E}})$.

Finally, we end this section with an illustration of our two relaxations on a small instance.

Example 6. *Let us consider the following polynomial optimization problem with $n = 4$ variables and $m = 7$ monomials.*

$$(Ex) \begin{cases} \min f(x) = -0.21x_1 - 0.08x_2 - 0.58x_3 - 0.49x_1x_4 \\ +0.78x_1x_3x_4 - 0.54x_1x_2x_4 + 0.88x_2x_3x_4 \\ s.t. \\ \quad x \in [0,1]^4 \end{cases}$$

The optimal value of (Ex) is $-1,32$.

Applying the quadratization PC0 (Algorithm 4.2.1), we introduce 3 additionnal variables and 7 constraints. We obtain the following equivalent quadratic problem $(Ex_{\mathcal{E}})$ to (Ex) with 7 variables.

$$(SDPEx_{\mathcal{E}}) \begin{cases} \min g(x) = -0.21x_1 - 0.08x_2 - 0.58x_3 - 0.49x_1x_4 \\ +0.78x_5x_4 - 0.54x_6x_4 + 0.88x_7x_4 \\ s.t. \\ \quad x_1x_3 = x_5 \\ \quad x_1x_2 = x_6 \\ \quad x_2x_3 = x_7 \\ \quad x \in [0,1]^7 \end{cases}$$

We now build the semidefinite relaxation $(Ex^0_{\mathcal{E}})$ *which contains* 35 *variables and* 11 *constraints. The optimal value of* $(Ex^0_{\mathcal{E}})$ *is* $-1,37$.

$$(SDPEx^0_{\mathcal{E}})\begin{cases} \min g(x,X) = -0.21x_1 - 0.08x_2 - 0.58x_3 - 0.49X_{1,4} \\ +0.78X_{5,4} - 0.54X_{6,4} + 0.88X_{7,4} \\ s.t. \\ \quad X_{ii} \leq x_i, \quad 1 \leq i \leq 7 \\ \quad X_{1,3} = x_5 \\ \quad X_{1,2} = x_6 \\ \quad X_{2,3} = x_7 \\ \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 \\ \quad x \in \mathbb{R}^7,\ X \in S^7 \end{cases}$$

We strengthen this first relaxation by adding the two families of valid constraints (6.9) and (6.10) that we expand in the following. The resulting semidefinite relaxation $(Ex^1_{\mathcal{E}})$ *has* 35 *variables and* 23 *constraints. The optimal value of* $(Ex^1_{\mathcal{E}})$ *is* $-1,32$ *which is also the optimal value of* (Ex).

$$(Ex^1_{\mathcal{E}})\begin{cases} \min g(x,X) \\ s.t. \\ \quad X_{ii} \leq x_i, \quad 1 \leq i \leq 7 \\ \quad X_{1,3} = x_5 \\ \quad X_{1,2} = x_6 \\ \quad X_{2,3} = x_7 \\ \quad X_{1,5} \leq x_5 \quad X_{3,5} \leq x_5 \\ \quad X_{1,6} \leq x_6 \quad X_{2,6} \leq x_6 \\ \quad X_{2,7} \leq x_7 \quad X_{3,7} \leq x_7 \\ \quad X_{2,5} = X_{6,3} \quad X_{3,6} = X_{7,1} \\ \quad X_{2,5} = X_{7,1} \quad X_{6,5} = X_{6,3} \\ \quad X_{7,6} = X_{7,1} \quad X_{7,5} = X_{7,1} \\ \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0 \\ \quad x \in \mathbb{R}^7,\ X \in S^7 \end{cases}$$

6.4 Conclusion

We introduced two SDP relaxations based on a quadratization and we prove their practical interest. Future work consists in embedding these relaxations within a spatial branch-and-bound framework in order to compute an exact solution. More precisely, we can apply a spatial

branch-and-bound algorithm on such relaxations by enforcing the equality between a new variable and its associated product through the branching phase.

Chapter 7

Computational experiments and implementations

Contents

7.1 Hardware and used software 113

7.2 Details on the instances 113

7.3 Performance of `PQCR` (presented in Chapter 4) 114

7.3.1 The compared methods (`PQCR`, `Q+QCR`, `Q+MIQCR`, `Q+cplex` and `Baron`) . . 114

7.3.2 Results on Vision instances 115

7.3.3 Results on LABS instances 116

7.4 Convex reformulation with other quadratization schemes (presented in Chapter 5) 123

7.4.1 Results related to Rosenberg's reformulation 123

7.4.2 Experimental comparison between termwise and pairwise convexification 131

7.5 Tightness of semi-definite relaxations for box-constrained polynomial programs (presented in Chapter 6) 136

7.5.1 Instance generation 136

7.5.2 Experimental results 136

We present the computational results obtained by the different methods presented in this thesis.

7.1 Hardware and used software

Experimental setting All our experiments were carried out on a server with 2 CPU Intel Xeon each of them having 12 cores and 2 threads of 2.5 GHz and $4*16$ GB of RAM using a Linux operating system. For all algorithms, we used the multi-threading version of `Cplex 12.7` with up to 48 threads. We have used the modeling language `AMPL` [29] to write our generic models. The different equivalent quadratic convex reformulations are solved by `Cplex` 12.7 [42]. The semi-definite relaxations are solved by the free software `csdp` [14] alongside with the `Conic Bundle` algorithm [40] that will be explained further.

The `Conic Bundle` algorithm [40] The semi-definite relaxations presented in this thesis can be hard to solve when considering dense instances. Standard semi-definite programming solvers that implement interior point algorithms [14, 77] fail to solve them. Thus, following the ideas of [12], we develop a sub-gradient algorithm within a Lagrangian duality framework to solve harder instances.

The algorithm is an iterative method that computes at each iteration a "best" approximation of dual variables. More precisely, it consists in dualizing a subset of the constraints in a given semi-definite relaxation. For our experiments, we choose to dualize every constraints, except $x_i^2 = x_i$. We then obtain a parameterized program. For a given parameter, we evaluate the obtained program which amounts to solve a semi-definite program with much less constraints than the original one. Then, a subgradient is computed from the solution of the previous program. With this subgradient, we can compute a new parameter that will be used at the next iteration. Our implementation includes an additional parameter p that is the proportion of considered constraints in the dualization. In that context, if $p < 1$, the algorithm returns only a feasible solution of the original semi-definite relaxation. This solution can also be optimal if only a proportion p of constraints are active at the optimum.

Parameters of the solvers To ensure that the experimental study is accurate, we set all the parameters of the solvers to the same values for each method.

- `Cplex` : we let the default parameters, except the parameter `qptolin` that is set to `0`.
- `csdp` : Parameters `axtol, aytol` of `Csdp` are set to 10^{-3}.
- `Conic Bundle` : the precision is set to 10^{-3}. Parameter p (see [12]) is set to 0.2.

7.2 Details on the instances

Although the presented methods can be applied to instances of any given degree, we focus on two real-life instances of degree 4 in this chapter.

- Vision instances (described in 1.2.1): The size of the considered instances are $l \times h = 10 \times 10$, 10×15, and 15×15, or in the polynomial formulation $n = 100, 150$ and 225, with a number of monomials of $m = 668$, 1033, and 1598 respectively. In our experiments, 15 instances of each size are considered obtaining a total of 45 instances. The ith instance of $l \times h$ pixels is labeled `v.l.h i`. Observe that the 15 instances of the same size have identical monomials with different coefficients, because they represent different images with the same number of pixels.

- LABS instances (described in 1.2.2): We consider the instances available on the MINLPLIB website [63]. We exploit a specific symmetry of the objective function by fixing to 0 the variable that appears the most in the polynomial. Each instance is labeled $\texttt{b.n.n}_0$.

7.3 Performance of PQCR (presented in Chapter 4)

7.3.1 The compared methods (PQCR, Q+QCR, Q+MIQCR, Q+cplex and Baron)

In our experiments, we refer to different algorithms:

- `PQCR`: This is our main algorithm that we have presented in Algorithm 4.1.1. A first phase of quadratic reformulation is performed on polynomial program (P) using algorithm PC0, described in Algorithm 4.2.1. Then an equivalent parameterized convex program is built and the optimal parameters are deduced from a semi-definite program. The resulting "optimal" convex program is solved by `Cplex 12.7` using a branch-and-bound algorithm. The quadratization is implemented in `C`, and we used the solver `csdp` to solve the semi-definite program. For denser instances, we used the solver `csdp` [14] together with the `Conic Bundle algorithm` [40] to solve the semi-definite program of `PQCR`, as described in [12].

- `Q+QCR` (described in Section 4.1.2.2): After a quadratization phase using algorithm PC0, we compute a diagonal convexification by considering the valid equality $x_i^2 = x_i$ only and the Fortet's constraints $\mathcal{F}_{\mathcal{E}}$ defined in Section 4.1.1. The resulting convex continuous relaxation is then solved at each node by `Cplex`.

- `Q+MIQCR` After the quadratization phase using PC0, we compute an equivalent quadratic program by perturbing every entries of the Hessian matrix. The resulting convex continuous relaxation is then solved at each node by `Cplex`.

- `Q+Cplex`: It consists in a quadratization followed by the direct submission to `Cplex 12.7`. Here again, the quadratization is implemented in `C`, and we used the `ampl` interface of the solver `Cplex`.

- `Baron`: Finally we evaluate the direct submission of polynomial program (P) to the general mixed-integer non-linear solver `Baron 17.4.1` [73]. Here, we used the `gams` interface of the solver `Baron`.

An important remark is that all the methods relying on a prior quadratization phase introduce additional variables through the pairwise cover PC0. We relax the integrality on these additional variables in our experiments, since the integrality is ensured by the linearization constraints $\mathcal{F}_{\mathcal{E}}$.

Here we evaluate `PQCR` on two applications. The first one is the *image restoration* problem (Vision) presented in Section 1.2.1, which results are presented in Table 7.1. The instances of this application are quite sparse with an average ratio $\frac{m}{n}$ of about 7. We choose to use these instances in order to compare `PQCR` with existing convexifications and in particular with methods `Q+QCR` and `Q+MIQCR` that are not able to handle larger and/or denser instances. Then, in Tables 7.2 and 7.3, we present the results of the second application, the LABS problem presented in Section 1.2.2 which instances are much denser (average ratio $\frac{m}{n}$ of about 212). These instances are available on the `minlplib` website [63], and are very hard to solve. For most of them, the optimal solution value is not known. The legend of the corresponding tables can be found in Figure 7.1.

7.3.2 Results on Vision instances

We focus on the comparison of several convexification methods after quadratization. Indeed, several ways are possible to solve the quadratic non-convex program (QP) obtained after the quadratization phase. For instance, the standard solver `Cplex` can directly handle it (`Q+Cplex`), or one can apply the `QCR` [13] (`Q+QCR`) or `MIQCR` [11] (`Q+MIQCR`) methods. We compare `PQCR` with these three approaches. We do not report the results for method `Q+MIQCR` since it was not able to start the computation due to the size of the considered instances. We also give the computational results coming from the direct submission of the initial polynomial (P) to the solver `Baron 17.4.1`. The results for these instances are summed up in Table 7.1, where each line corresponds to one instance.

We start by comparing the convexification phase of our `PQCR` algorithm with the original `Q+QCR` and `Q+MIQCR` methods. We observe that none of these convexifications are able to handle any considered instances: `Q+QCR` because of the weakness of its initial bound, and `Q+MIQCR` because of the size of the semi-definite problem considered for computing the best reformulation. Note that `Q+QCR` has a better gap than `Q+Cplex` and `Baron`, which shows that a linearization method performs better than a weak quadratic convex reformulation. These experiments confirm the interest of designing `PQCR`, an algorithm devoted to polynomial optimization. Then, we can see that `Q+Cplex` dominates `PQCR`. However, one has to note that these instances are very sparse (average ratio $\frac{m}{n}$ of about 7). It is well known that the standard linearization performs very well on sparse instances, and `Cplex` linearizes non-diagonal quadratic terms. Clearly, for these instances, the time spent on solving a large semi-definite program, even once, is not profitable in

comparison to the efficiency of LP heuristic or cut methods implemented in `cplex 12.7`. Indeed, `Q+Cplex` solves all the considered instances at the root node of its branch-and-bound. Moreover, it is interesting to remark that 99% of the CPU time of `PQCR` is spent for solving the semi-definite relaxation (SDP), while the CPU time for solving the optimal quadratic convex reformulation (QP^*) is always smaller than 14 seconds. Finally, we compare `PQCR` with the direct submission to the solver `Baron`. We observe that `Baron` is faster than `PQCR` on the medium size instances ($n = 100$ or 150), but is not able to solve all the larger instances within the time limit. Indeed, for $n = 225$, it solves only 3 instances out of 15. On the contrary, `PQCR` seems quite stable to the increase of the size of the instances. Indeed, the initial gap remains stable and very weak (0.42% on average) while the total CPU time increases reasonably.

7.3.3 Results on LABS instances

For these instances, we do not report the results for methods `Q+QCR` and `Q+MIQCR` since they have failed to solve all the considered instances. Two instances that are already quadratic (`b.20.03` and `b.25.03`) are solved by the method `Q+Cplex` in 7 and 75 seconds respectively. However, this method was not able to solve the other instances within the time limit.

We present in Table 7.2 a detailed comparison of `PQCR` with the direct submission to `baron 17.1.4`. For these experiences the total time limit was set to 5 hours, and we limit the CPU time for solving (SDP) to 3 hours. Indeed, any feasible solution to the dual of (SDP) can be used to get a convex objective function in the equivalent formulation. Thus, if the CPU time in column $tSdp$ is smaller than three hours it means that (SDP) was solved to optimality. In the other case, we get a feasible dual solution and we can suppose that the initial gap of `PQCR` could be improved. For these instances `PQCR` is faster than baron since it solves 17 instances out of 45 within the time limit while `baron` solves only 13 instances. Here, `baron 17.1.4` solves 2 instances that were stated as unsolved on `minlplib`. As expected, `PQCR` has an initial gap much smaller that `baron` (reduced by a factor 22 on average). We also observe that the number of nodes visited by `Cplex` during the branch-and-bound is significantly larger than the the number of nodes of `baron` (increased by a factor of about 40000 on average).

We present in Table 7.3 the values of the best solutions and of the final lower bounds obtained by `PQCR` within 5 hours of CPU time, and those available on the `minlplib` website. More precisely, we report in the column `minlplib` the best solution/final lower bound value obtained among the results of the solvers `Antigone`, `Baron`, `Couenne`, `Lindo`, and `Scip`. `PQCR` solves to optimality 6 unsolved instances (labeled as **). It also improves the best known solution values of 9 instances (labeled as $^{\#}$), and improves the final lower bound of all the unsolved instances (labeled as *). In this table, each line corresponds to one instance, and we only present results for instances that were stated as unsolved on `minlplib`.

To illustrate these results, we plot in Figure 7.2, for each instance reported in Table 7.3, the *final gap* of `PQCR` and `minlplib`. Clearly, the final gap of `PQCR` is much smaller than the final gap

of `minlplib` (reduced by a factor 3 on average).

A last remark concerns the CPU time necessary to solve (SDP). Indeed, this time represents on average 75% of the total CPU time. A natural improvement would be to identify the set of "important equalities" in a preprocessing step in order to improve the behavior of the solution of (SDP).

- *Name*: Name of the considered instance.
- n: number of variables in the initial polynomial formulation.
- m: number of monomials.
- BKN: is the optimal solution value or the best known solution value of the instance.
- N: number of variables after quadratization.
- *gap*: is the initial gap, i.e. the gap at the root node of the branch-and-bound, $gap = \left|\frac{BKN - LB_i}{BKN}\right| * 100$, where LB_i is the initial lower bound.
- *Solution*: best solution value found within the time limit.
- *Const SDP*: number of constraints in the semi-definite relaxation.
- *Opt*: optimal value of the objective function.
- *LB*: Optimal value of the associated semi-definite relaxation.
- T_t: total CPU time in seconds of the associated method. It takes into account the SDP time and the branch-and-bound time.
- *tSdp*: CPU time in seconds for solving semi-definite programs in `PQCR` and `Q+QCR`. The time limit is set to 2400 seconds for the *vision* problem and 3 hours for the *LABS* problem. If the solver reaches the time limit, *tSdp* is labeled as "-".
- T_t: total CPU time in seconds of the associated method. The time limit is set to 1 hour for the *vision* problem and 5 hours for the *LABS* problem. If an instance remains unsolved within the time limit, we put the *final gap*= $\left|\frac{BKN - LB_f}{BKN}\right| * 100$, where LB_f is the final lower bound.
- *Nodes*: number of nodes visited by the branch-and-bound algorithm.
- `-` : means that the time limit of 3h on the SDP phase is reached.

Figure 7.1: *Legend of Tables 7.1-7.3.*

Instance				PQCR			Q+QCR			Q+Cplex		Baron	
Name	n	m	N	*Gap (%)*	*tSdp*	T_t	*Gap (%)*	*tSdp*	T_t	*Gap (%)*	T_t	*Gap (%)*	T_t
v.10.10 1	100	668	352	**0,59**	66	68	396	7	(250 %)	1113	**2**	1098	15
v.10.10 2	100	668	352	**0,28**	64	66	536	8	(343 %)	1549	**2**	1529	10
v.10.10 3	100	668	352	**0,05**	65	67	973	8	(573 %)	3375	**1**	3332	6
v.10.10 4	100	668	352	**0,12**	63	65	957	8	(561 %)	3377	**1**	3334	6
v.10.10 5	100	668	352	**0,13**	65	66	1006	8	(585 %)	3568	**1**	3523	5
v.10.10 6	100	668	352	**0,11**	73	74	359	9	(229 %)	984	**2**	972	11
v.10.10 7	100	668	352	**0,02**	64	65	305	8	(194 %)	829	**2**	817	14
v.10.10 8	100	668	352	**1,37**	64	66	1376	8	(804 %)	4765	**1**	4705	7
v.10.10 9	100	668	352	**3,02**	65	67	1749	8	(1026 %)	6187	**1**	6110	4
v.10.10 10	100	668	352	**3,64**	66	68	1879	8	(1075 %)	6843	**1**	6757	4
v.10.10 11	100	668	352	**0,36**	70	72	489	8	(316 %)	1388	**2**	1370	35
v.10.10 12	100	668	352	**0,20**	70	72	361	9	(232 %)	997	**2**	984	23
v.10.10 13	100	668	352	**0,00**	60	61	709	8	(392 %)	2654	**1**	2620	2
v.10.10 14	100	668	352	**0,00**	60	61	546	8	(297 %)	2027	**1**	2001	2
v.10.10 15	100	668	352	**0,00**	118	119	541	8	(285 %)	2048	**1**	2022	1
v.10.15 1	150	1033	542	**0,31**	290	294	447	24	(351 %)	1245	**5**	1234	80
v.10.15 2	150	1033	542	**0,00**	285	287	367	24	(287 %)	999	**5**	990	36
v.10.15 3	150	1033	542	**0,05**	280	283	1027	27	(772 %)	3549	**3**	3520	6
v.10.15 4	150	1033	542	**0,33**	276	280	845	27	(640 %)	2840	**3**	2817	7
v.10.15 5	150	1033	542	**0,07**	269	271	799	27	(595 %)	2808	**3**	2785	4
v.10.15 6	150	1033	542	**0,55**	297	302	462	25	(366 %)	1277	**5**	1266	47
v.10.15 7	150	1033	542	**0,08**	288	291	360	26	(283 %)	981	**5**	972	38
v.10.15 8	150	1033	542	**0,79**	281	284	1792	26	(1356 %)	6202	**3**	6152	19
v.10.15 9	150	1033	542	**1,80**	283	286	1525	26	(1160 %)	5209	**3**	5167	10
v.10.15 10	150	1033	542	**1,38**	275	279	1510	25	(1124 %)	5500	**3**	5456	7
v.10.15 11	150	1033	542	**0,10**	283	286	391	25	(305 %)	1102	**5**	1092	41
v.10.15 12	150	1033	542	**0,60**	275	279	453	25	(355 %)	1269	**5**	1258	125
v.10.15 13	150	1033	542	**0,00**	254	256	634	27	(469 %)	2254	**3**	2236	4
v.10.15 14	150	1033	542	**0,04**	269	273	731	27	(547 %)	2590	**3**	2569	2
v.10.15 15	150	1033	542	**0,00**	258	259	576	28	(423 %)	2183	**2**	2165	2
v.15.15 1	225	1598	827	**0,12**	1234	1244	365	70	(320 %)	998	**9**	993	(95 %)
v.15.15 2	225	1598	827	**0,43**	1251	1265	482	83	(421 %)	1350	**9**	1343	(138 %)
v.15.15 3	225	1598	827	**0,10**	1167	1175	678	65	(582 %)	2326	**5**	2313	(83 %)
v.15.15 4	225	1598	827	**0,04**	1251	1256	877	65	(753 %)	2996	**5**	2980	(127 %)
v.15.15 5	225	1598	827	**0,03**	1167	1174	641	67	(546 %)	2252	**5**	2240	(76 %)
v.15.15 6	225	1598	827	**0,28**	1238	1249	403	64	(353 %)	1104	**10**	1098	(107 %)
v.15.15 7	225	1598	827	**0,36**	1237	1246	525	67	(463 %)	1455	**10**	1447	(144 %)
v.15.15 8	225	1598	827	**0,29**	1197	1205	1148	73	(979 %)	4104	**5**	4082	(137 %)
v.15.15 9	225	1598	827	**0,27**	1170	1176	1542	66	(1315 %)	5570	**5**	5541	(171 %)
v.15.15 10	225	1598	827	**0,31**	1173	1179	1194	67	(1020 %)	4380	**5**	4357	1154
v.15.15 11	225	1598	827	**0,27**	1224	1230	529	69	(462 %)	1528	**8**	1520	(144 %)
v.15.15 12	225	1598	827	**0,25**	1225	1235	461	68	(414 %)	1273	**12**	1266	(133 %)
v.15.15 13	225	1598	827	**0,00**	1124	1128	651	63	(551 %)	2398	**4**	2385	1239
v.15.15 14	225	1598	827	**0,02**	1171	1177	651	63	(553 %)	2359	**5**	2346	(79 %)
v.15.15 15	225	1598	827	**0,00**	1100	1103	609	65	(513 %)	2320	**4**	2308	263

Table 7.1: *Comparison of* ***PQCR*** *to* ***Q+QCR***, ***Q+Cplex*** *and* ***Baron*** *for the vision instances - time limit one hour. Legend on Figure 7.1.*

Instance			PQCR					Baron 17.4.1		
Name	*n*	*m*	*N*	*Gap (%)*	*tSdp*	T_t	*Nodes*	*Gap (%)*	T_t	*Nodes*
b.20.03	20	38	20	**0**	1	2	0	100	**1**	1
b.20.05	20	207	65	**23**	22	23	5886	1838	**2**	1
b.20.10	20	833	124	**8**	837	846	24183	2918	**125**	7
b.20.15	20	1494	164	**5**	1228	1242	9130	3202	**728**	9
b.25.03	25	48	25	**0**	1	2	0	100	**0**	1
b.25.06	25	407	105	**17**	461	469	163903	2307	**65**	27
b.25.13	25	1782	206	**4**	**1552**	1603	76828	3109	3750	75
b.25.19	25	3040	265	**4**	–	**13433**	224550	3356	14399	129
b.25.25	25	3677	289	**5**	–	**13395**	167423	3405	(12 %)	100
b.30.04	30	223	82	**23**	58	78	134635	1347	**7**	7
b.30.08	30	926	174	**10**	1940	**2040**	752765	2696	2778	237
b.30.15	30	2944	296	**5**	–	**13525**	438278	3221	(21 %)	103
b.30.23	30	5376	390	**11**	5953	**6865**	9337391	3450	(135 %)	8
b.30.30	30	6412	422	**4**	8500	**15352**	452460	3470	(161 %)	5
b.35.04	35	263	97	**19**	135	167	156085	1350	**32**	13
b.35.09	35	1381	234	**10**	2245	**4630**	8163651	2826	(29 %)	354
b.35.18	35	5002	419	**644**	–	**(12 %)**	4899872	3356	(133 %)	10
b.35.26	35	8347	530	**30**	–	**(5 %)**	5006407	3508	(229 %)	3
b.35.35	35	10252	579	**12**	–	**(11 %)**	134426	3499	(214 %)	3
b.40.05	40	447	145	**25**	430	**1630**	23459121	1856	3674	1021
b.40.10	40	2053	304	**9**	–	**(4 %)**	25480163	2953	(54 %)	147
b.40.20	40	7243	544	**9**	–	**(4 %)**	9783350	3405	(203 %)	3
b.40.30	40	12690	702	**360**	–	**(25 %)**	281134	3561	(274 %)	1
b.40.40	40	15384	762	**62**	–	**(44 %)**	57534	3536	(464 %)	1
b.45.05	45	507	165	**24**	1384	**(4 %)**	84159279	1854	**16609**	4727
b.45.11	45	2813	382	**9**	–	**(2 %)**	25114985	3018	(132 %)	33
b.45.23	45	10776	706	**21**	–	**(16 %)**	1225234	3470	(242 %)	2
b.45.34	45	18348	898	**137**	–	**(105 %)**	38513	3604	(375 %)	1
b.45.45	45	21993	969	**187**	–	**(153 %)**	25964	3559	(624 %)	1
b.50.06	50	882	230	**19**	1230	**(9 %)**	49490829	2321	(35 %)	1225
b.50.13	50	4457	506	**8**	–	**(5 %)**	12039566	3131	(192 %)	7
b.50.25	50	14412	866	**676**	–	**(247 %)**	684010	3511	(280 %)	1
b.50.38	50	25446	1118	**242**	–	**(163 %)**	309289	3646	(505 %)	1
b.50.50	50	30271	1202	**360**	–	**(305 %)**	49507	3541	(729 %)	1
b.55.06	55	977	255	**21**	–	**(11 %)**	23603952	2323	(54 %)	6
b.55.14	55	5790	607	**11**	–	**(7 %)**	7829649	3186	(373 %)	6
b.55.28	55	19897	1069	**174**	–	**(106 %)**	580827	3553	(646 %)	2
b.55.41	55	33318	1347	**330**	–	**(244 %)**	117912	3654	(639 %)	1
b.55.55	55	40402	1459	**547**	–	**(493 %)**	117027	3575	(705 %)	1
b.60.08	60	2036	384	**12**	–	**(9 %)**	24800852	2712	(175 %)	1
b.60.15	60	7294	716	**16**	–	**(14 %)**	4044387	3236	(404 %)	1
b.60.30	60	25230	1264	**256**	–	**(165 %)**	295197	3578	(471 %)	1
b.60.45	60	43689	1614	**547**	–	**(439 %)**	26955	704	(671 %)	1
b.60.60	60	52575	1742	**784**	–	**(704 %)**	23716	3604	(762 %)	1

Table 7.2: *Results of* **PQCR** *and* **Baron** *for the 45 instances of the LABS problem. Time limit 5 hours (3h for SDP, 2h for* **Cplex***). Legend on Figure 7.1.*

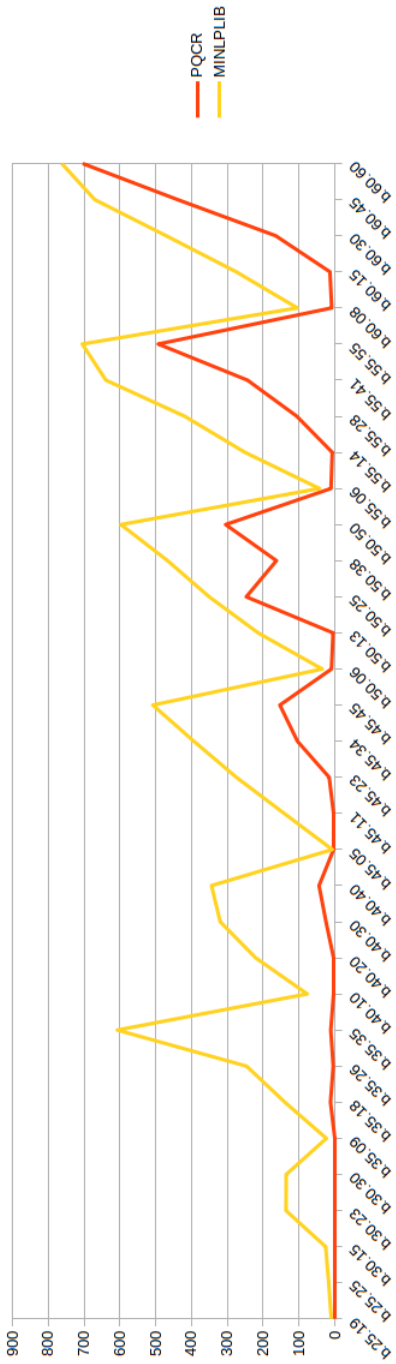


Figure 7.2: *Comparison between the final gap of* `PQCR` *and the final gap computed with the best known solution and the best bound from* `minlplib` *for the unsolved auto-correlation instances*

Instance	PQCR (5h)		minlplib [63]	
Name	*Solution*	LB_f	*Solution*	LB_f
b.25.19**	-14644	-14644	-14644	-16108
b.25.25**	-10664	-10664	-10664	-12494
b.30.15**	-15744	-15744	-15744	-19780
b.30.23**	-30460	-30460	-30420	-72030
b.30.30**	-22888	-22888	-22888	-54014
b.35.09**	-5108	-5108	-5108	-6312
b.35.18*	-31144	-34964	-31160	-74586
b.35.26#*	-55288	-57789	-55184	-191466
b.35.35*	-41052	-45787	-41068	-290424
b.40.10#*	-8248	-8551	-8240	-14618
b.40.20#*	-50576	-52465	-50516	-162365
b.40.30#*	-94872	-118324	-94768	-398617
b.40.40*	-67528	-98031	-67964	-302028
b.45.05*	-1068	-1112	-1068	-1145
b.45.11#*	-12748	-13035	-12740	-30771
b.45.23#*	-85423	-98984	-85248	-320397
b.45.34*	-151352	-311627	-152368	-752427
b.45.45*	-111292	-285811	-112764	-685911
b.50.06*	-2160	-2363	-2160	-2921
b.50.13#*	-23791	-24975	-23772	-74768
b.50.25*	-124572	-433247	-124748	-562446
b.50.38*	-232344	-611906	-232496	-1318325
b.50.50*	-162640	-681105	-168216	-1173058
b.55.06*	-2400	-2659	-2400	-3439
b.55.14#*	-33272	-35698	-33168	-116748
b.55.28*	-189896	-392929	-190472	-989145
b.55.41*	-335388	-1160180	-337388	-2494477
b.55.55*	-233648	-1434663	-241912	-1947633
b.60.08*	-6792	-7388	-6792	-13915
b.60.15#*	-45232	-51467	-44896	-169767
b.60.30*	-259271	-692721	-261048	-1491016
b.60.45*	-475504	-2579935	-478528	-3687344
b.60.60*	-343400	-2816441	-350312	-3021077

Table 7.3: *Comparison of the best known solution and best lower bound values of* ***PQCR*** *and of the* ***minlplib*** *for the unsolved LABS instances.* $**$*: solved for the first time,* #*: best known solution improved, and* $*$*: best known lower bound improved. Legend on Figure 7.1.*

7.4 Convex reformulation with other quadratization schemes (presented in Chapter 5)

7.4.1 Results related to Rosenberg's reformulation

We presented theoretical results on the tightness of the different bounds presented in Chapter 5, but we want to measure experimentally this difference on real polynomial instances. We thus compare several convexifications on the instances presented in Sections 1.2.1 and 1.2.2. The results are presented in Tables 7.4-7.11, and the corresponding legend can be found in Figure 7.3. We first describe the methods we refer to in this section:

- `PQCR` detailed in Section 7.3.1.
- `Q+QCR` detailed in Section 7.3.1.
- `ROS+QCR` (Described in Section 5.1.1): We apply Rosenberg's procedure to (P) by using the pairwise cover PC0. We then apply the `QCR` method to the obtained equivalent unconstrained quadratic program. The resulting convex continuous relaxation is then solved by `Cplex`. For the penalty term in Rosenberg's procedure, we choose the sum of absolute values of negative coefficients of f.
- `PQCR+ROS` (Described in Section 5.1.2): After the quadratization phase using PC0, we apply a convexification using the valid equalities $x_i^2 = x_i$ and $x_ix_j - 2x_ix_k - 2x_jx_k + 3x_k = 0$ and the Fortet's constraints $\mathcal{F}_\mathcal{E}$. The resulting convex continuous relaxation is then solved at each node by `Cplex`.
- `PQCR 2`: After a quadratization phase using algorithm PC0, we compute a compact convexification by considering the valid equalities $x_i^2 = x_i$ and $y_{ij} = x_ix_j$ of $S_\mathcal{E}$ defined in Lemma 4.1.1, and the Fortet's constraints $\mathcal{F}_\mathcal{E}$. The resulting convex continuous relaxation is then solved at each node by `Cplex`.

We start by noticing that `ROS+QCR` is not a viable method to solve problem (P), since only the instances that are already quadratic can be solved by this method. Indeed, the average gap is huge with more than $10^5\%$ for both the Vision instances (Table 7.8) and the LABS instances (Table 7.4). This can be explained by the fact that we apply a diagonal convexification on a quadratic function whose Hessian matrix contains large positive values (up to 10^6) in off-diagonal terms. In order to get a positive semi-definite Hessian matrix, the `QCR` algorithm introduce diagonal parameters of the same magnitude to balance the Hessian matrix. We know that large positive values for diagonal convexification significantly deteriorate the quality of the bound and that's the most likely reason to explain such a gap.

Next we focus on `Q+QCR` which results are presented in Tables 7.6 and 7.10, and `PQCR 2` which results are presented in Tables 7.7 and 7.11. Obviously, `PQCR 2` has a better continuous relaxation bound value than `Q+QCR`, but it is interesting to experimentally measure the differences between the two methods. We can notice for example that adding only N additional constraints reduces the gap by half. As a matter of fact, `Q+QCR` has an average gap of 776% on Vision instances and 404% on LABS instances, whereas `PQCR 2` obtains 441% and 269% respectively. Again, this can be explained theoretically, as unlike the $x_i^2 = x_i$, the valid equality $y_{ij} = x_i x_j$ involves both initial and additional variables and as a result, the continuous relaxation bound is much stronger.

One interesting result to analyze is the performance of `PQCR ROS` defined in Section 5.1.2 (Tables 7.5 and Table 7.9). In our theoretical development, we proved that this equality is equivalent to $y_{ij} = x_i x_j$ when considering the $\{0,1\}^N$ space. The main difference between these constraints is that the equality $x_i x_j - 2x_i y_{ij} - 2x_j y_{ij} + 3y_{ij} = 0$ modifies more entries of the Hessian matrix and thus, could lead to sharper bounds. Experimentally, we indeed measure better performances for `PQCR ROS` than `PQCR 2`. Recall that both methods have the same number of constraints in their respective semi-definite relaxations. As for the LABS instances, both methods solve 8 out of 9. However, the bound obtained by `PQCR ROS` is twice as better (166% on average for Vision and 146% for LABS) and its number of nodes is also reduced. For these reasons, `PQCR ROS` is able to solve every instance within 7 minutes, whereas it takes more than 12 minutes for `PQCR 2` to solve the hardest instance. For the vision instances, the difference is more remarkable as `PQCR 2` cannot solve any instance within one hour whereas `PQCR ROS` solves 10 out of 45. The quality of the gap of `PQCR ROS` seems to play a major role here as the number of nodes is reduced by a factor 1.5 and leads to the resolution of several instances. In terms of computational results, this reformulation is very interesting as it perturbs the Hessian matrix both on diagonal and non-diagonal entries at the same cost as `PQCR 2`. Indeed, the computation of the semi-definite relaxation is still immediate but the gain on the bound is significant and allows to solve additional instances. Theoretically, it means that, although both equalities are equivalent in the quadratic reformulation, their linearized counterparts do not describe the same set of points in the semi-definite relaxation. Experimentally, it seems that the equality $X_{ij} - 2X_{ik} - 2X_{jk} + 3x_k = 0$ is a tighter cut than $X_{ij} = x_k$.

Finally note that the method `PQCR` dominates all the previously mentioned methods concerning continuous relaxation bound values, as it has an average bound of 0.42% on Vision instances Table (7.1) and 11% on LABS instances (Table 7.2, note that the small LABS instances are solved within one hour even if the time limit is set to 5 hours). However, on small LABS instances, we remark that `Q+QCR`, `PQCR 2` and `PQCR ROS` are faster than `PQCR`. Indeed, `PQCR` spends a lot of time on solving a heavy semi-definite relaxation leading to tight bounds. On small instances, this difference between the bounds is not significant enough as the branch-and-bound of `Cplex` computes the optimal solution within a few seconds. As a conclusion, on small instances,

it is not worth computing a heavy semi-definite relaxation and one would prefer running the alternative methods we presented in this paragraph instead. Finally, to support this conclusion, these alternative methods should be viewed as complementary methods to `PQCR` on easy instances, rather than competitive methods.

- *Name*: name of the considered instance.
- N: total number of variables after each quadratization phase.
- *Const*: number of constraints in the quadratic convex reformulation.
- *Const SDP*: number of constraints in the semi-definite relaxation.
- *Opt*: optimal value of the objective function.
- *BKN*: is the best known solution value of the instance.
- *LB*: Optimal value of the associated semi-definite relaxation.
- *gap*: is the initial gap, i.e. the gap at the root node of the branch-and-bound, $gap = \left|\frac{BKN - LB_i}{BKN}\right| * 100$, where LB_i is the initial lower bound.
- *Nodes*: number of nodes visited by the branch-and-bound algorithm.
- T_t: total CPU time in seconds of the associated method. It takes into account the SDP time and the branch-and-bound time.
- `-` : means that the time limit is reached.

Figure 7.3: *Legend of Tables 7.4-7.15*

Name	*N*	*Const*	*Const SDP*	*Opt*	*Gap (%)*	*Nodes*	T_t
`b.20.3`	20	0	21	-72	0	0	0
`b.20.5`	65	0	66	-416	804612	73437542	-
`b.20.10`	124	0	125	-2936	31206	38511235	-
`b.20.15`	164	0	165	-5960	48643	35727150	-
`b.25.3`	25	0	26	-92	0	0	0
`b.25.6`	105	0	106	-960	1843723	36766956	-
`b.25.13`	206	0	207	-8144	59614	20336389	-
`b.30.4`	82	0	83	-324	829465	38587389	-
`b.35.4`	97	0	98	-384	1038548	39040424	-

Table 7.4: *Performance of* `ROS+QCR` *on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

Name	*N*	*Const*	*Const SDP*	*Opt*	*Gap (%)*	*Nodes*	T_t
b.20.3	20	0	21	-72	0	0	1
b.20.5	65	180	291	-416	160	87279	10
b.20.10	124	416	645	-2936	217	279796	25
b.20.15	164	576	885	-5960	276	401423	27
b.25.3	25	0	26	-92	0	0	1
b.25.6	105	320	506	-960	191	4734342	108
b.25.13	206	724	1112	-8148	230	7609987	344
b.30.4	82	208	343	-324	118	20439127	438
b.35.4	97	248	408	-384	119	63351897	-

Table 7.5: *Performance of* **PQCR ROS** *on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

Name	*N*	*Const*	*Const SDP*	*Opt*	*Gap (%)*	*Nodes*	T_t
b.20.3	20	0	21	-72	0	0	1
b.20.5	65	180	246	-416	489	153530	7
b.20.10	124	416	541	-2936	552	420776	32
b.20.15	164	576	741	-5960	598	475440	41
b.25.3	25	0	26	-92	0	0	1
b.25.6	105	320	426	-960	579	7712985	154
b.25.13	206	724	931	-8148	566	10528468	422
b.30.4	82	208	291	-324	423	60649294	1255
b.35.4	97	248	346	-384	428	54495492	-

Table 7.6: *Performance of* **Q+QCR** *on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

Name	*N*	*Const*	*Const SDP*	*Opt*	*Gap (%)*	*Nodes*	T_t
b.20.3	20	0	21	-72	0	0	0
b.20.5	65	180	291	-416	342	109396	4
b.20.10	124	416	645	-2936	350	323266	20
b.20.15	164	576	885	-5960	364	455265	32
b.25.3	25	0	26	-92	0	0	0
b.25.6	105	320	506	-960	403	5946968	132
b.25.13	206	724	1112	-8148	354	7959187	279
b.30.4	82	208	343	-324	303	36426676	709
b.35.4	97	248	408	-384	307	59173465	-

Table 7.7: *Performance of* **PQCR 2** *on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

Name	*N*	*Const*	*Const SDP*	*Opt*	*Gap (%)*	*Nodes*	T_t
v.10.10 1	352	0	353	-810	38129	26146170	-
v.10.10 2	352	0	353	-595	51819	26264035	-
v.10.10 3	352	0	353	-280	109073	28656310	-
v.10.10 4	352	0	353	-280	109106	28560442	-
v.10.10 5	352	0	353	-265	115099	28136129	-
v.10.10 6	352	0	353	-905	34075	29567303	-
v.10.10 7	352	0	353	-1060	29191	27511353	-
v.10.10 8	352	0	353	-200	152704	28935338	-
v.10.10 9	352	0	353	-155	197337	29465221	-
v.10.10 10	352	0	353	-140	217844	28322295	-
v.10.10 11	352	0	353	-660	46770	29361189	-
v.10.10 12	352	0	353	-900	34488	28307833	-
v.10.10 13	352	0	353	-355	86361	28494142	-
v.10.10 14	352	0	353	-460	66709	28780161	-
v.10.10 15	352	0	353	-455	67366	28561339	-
v.10.15 1	542	0	543	-1135	65813	21605315	-
v.10.15 2	542	0	543	-1390	53776	22112650	-
v.10.15 3	542	0	543	-415	178399	21699346	-
v.10.15 4	542	0	543	-515	143729	23412975	-
v.10.15 5	542	0	543	-520	142118	23935673	-
v.10.15 6	542	0	543	-1110	67364	23778466	-
v.10.15 7	542	0	543	-1415	52875	24187953	-
v.10.15 8	542	0	543	-240	308101	23512819	-
v.10.15 9	542	0	543	-285	259550	20895722	-
v.10.15 10	542	0	543	-270	273685	24518261	-
v.10.15 11	542	0	543	-1270	58788	22255609	-
v.10.15 12	542	0	543	-1115	66970	24327878	-
v.10.15 13	542	0	543	-645	115060	24477737	-
v.10.15 14	542	0	543	-565	131491	24204581	-
v.10.15 15	542	0	543	-665	111554	24058813	-
v.15.15 1	827	0	828	-2155	82560	14821039	-
v.15.15 2	827	0	828	-1630	109039	15502130	-
v.15.15 3	827	0	828	-970	182251	14235378	-
v.15.15 4	827	0	828	-760	232624	14329857	-
v.15.15 5	827	0	828	-1000	176691	15031426	-
v.15.15 6	827	0	828	-1970	90510	15582782	-
v.15.15 7	827	0	828	-1520	116939	15595444	-
v.15.15 8	827	0	828	-560	315795	14430103	-
v.15.15 9	827	0	828	-415	425951	14718402	-
v.15.15 10	827	0	828	-525	336530	13512351	-
v.15.15 11	827	0	828	-1455	122402	21603879	-
v.15.15 12	827	0	828	-1730	103262	21591569	-
v.15.15 13	827	0	828	-945	187622	15160372	-
v.15.15 14	827	0	828	-960	184703	15150889	-
v.15.15 15	827	0	828	-975	181747	18182499	-

Table 7.8: *Performance of* **ROS+QCR** *on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

Name	*N*	*Const*	*Const SDP*	*Opt*	*Gap (%)*	*Nodes*	T_t
v.10.10 1	352	1008	1613	-810	129	26694693	–
v.10.10 2	352	1008	1613	-595	170	24887588	–
v.10.10 3	352	1008	1613	-280	198	7377592	673
v.10.10 4	352	1008	1613	-280	184	2929643	280
v.10.10 5	352	1008	1613	-265	193	2468343	191
v.10.10 6	352	1008	1613	-905	120	25073715	–
v.10.10 7	352	1008	1613	-1060	102	24731654	–
v.10.10 8	352	1008	1613	-200	289	23552060	1001
v.10.10 9	352	1008	1613	-155	341	7030061	714
v.10.10 10	352	1008	1613	-140	340	1598095	185
v.10.10 11	352	1008	1613	-660	163	27881823	–
v.10.10 12	352	1008	1613	-900	120	25390765	–
v.10.10 13	352	1008	1613	-355	100	62103	38
v.10.10 14	352	1008	1613	-460	71	44271	23
v.10.10 15	352	1008	1613	-455	66	19716	15
v.10.15 1	542	1568	2503	-1135	149	19661206	–
v.10.15 2	542	1568	2503	-1390	126	18481035	–
v.10.15 3	542	1568	2503	-415	208	16367035	–
v.10.15 4	542	1568	2503	-515	185	17147424	–
v.10.15 5	542	1568	2503	-520	150	17245322	–
v.10.15 6	542	1568	2503	-1110	159	19672116	–
v.10.15 7	542	1568	2503	-1415	122	18126880	–
v.10.15 8	542	1568	2503	-240	372	16300894	–
v.10.15 9	542	1568	2503	-285	330	16402882	–
v.10.15 10	542	1568	2503	-270	260	16186793	–
v.10.15 11	542	1568	2503	-1270	126	17744761	–
v.10.15 12	542	1568	2503	-1095	155	17290458	–
v.10.15 13	542	1568	2503	-645	102	15786500	–
v.10.15 14	542	1568	2503	-565	120	16598840	–
v.10.15 15	542	1568	2503	-665	67	319546	166
v.15.15 1	827	2408	3838	-2155	121	11538754	–
v.15.15 2	827	2408	3838	-1630	157	10079701	–
v.15.15 3	827	2408	3838	-970	132	7032342	–
v.15.15 4	827	2408	3838	-760	179	8278511	–
v.15.15 5	827	2408	3838	-1000	114	4203730	–
v.15.15 6	827	2408	3838	-1970	136	10898839	–
v.15.15 7	827	2408	3838	-1520	180	10820196	–
v.15.15 8	827	2408	3838	-560	201	7000064	–
v.15.15 9	827	2408	3838	-415	259	7090928	–
v.15.15 10	827	2408	3838	-525	188	3846681	–
v.15.15 11	827	2408	3838	-1395	176	9187650	–
v.15.15 12	827	2408	3838	-1690	163	10018801	–
v.15.15 13	827	2408	3838	-945	83	5881992	–
v.15.15 14	827	2408	3838	-960	93	560000	–
v.15.15 15	827	2408	3838	-975	64	393056	–

Table 7.9: *Performance of* `PQCR ROS` *on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

Name	*N*	*Const*	*Const SDP*	*Opt*	*Gap (%)*	*Nodes*	T_t
v.10.10 1	352	1008	1361	-810	396	29093538	-
v.10.10 2	352	1008	1361	-595	536	26308369	-
v.10.10 3	352	1008	1361	-280	973	25377770	-
v.10.10 4	352	1008	1361	-280	957	25177676	-
v.10.10 5	352	1008	1361	-265	1006	24068178	-
v.10.10 6	352	1008	1361	-905	359	26207273	-
v.10.10 7	352	1008	1361	-1060	305	26429950	-
v.10.10 8	352	1008	1361	-200	1376	24684882	-
v.10.10 9	352	1008	1361	-155	1749	24187338	-
v.10.10 10	352	1008	1361	-140	1879	25307018	-
v.10.10 11	352	1008	1361	-660	489	26623135	-
v.10.10 12	352	1008	1361	-900	361	25468762	-
v.10.10 13	352	1008	1361	-355	709	23737082	-
v.10.10 14	352	1008	1361	-460	546	22692423	-
v.10.10 15	352	1008	1361	-455	541	23620814	-
v.10.15 1	542	1568	2111	-1135	447	17890981	-
v.10.15 2	542	1568	2111	-1390	367	17939444	-
v.10.15 3	542	1568	2111	-415	1027	15984793	-
v.10.15 4	542	1568	2111	-515	845	16369673	-
v.10.15 5	542	1568	2111	-520	799	15767968	-
v.10.15 6	542	1568	2111	-1110	462	17083565	-
v.10.15 7	542	1568	2111	-1415	360	16765612	-
v.10.15 8	542	1568	2111	-240	1792	14425697	-
v.10.15 9	542	1568	2111	-285	1525	14172456	-
v.10.15 10	542	1568	2111	-270	1510	14082116	-
v.10.15 11	542	1568	2111	-1270	391	16575441	-
v.10.15 12	542	1568	2111	-1115	453	14605369	-
v.10.15 13	542	1568	2111	-645	634	13722945	-
v.10.15 14	542	1568	2111	-565	731	13766844	-
v.10.15 15	542	1568	2111	-665	576	12604173	-
v.15.15 1	827	2408	3236	-2155	365	9341315	-
v.15.15 2	827	2408	3236	-1630	482	8592313	-
v.15.15 3	827	2408	3236	-970	678	8701600	-
v.15.15 4	827	2408	3236	-760	877	8851069	-
v.15.15 5	827	2408	3236	-1000	641	7540246	-
v.15.15 6	827	2408	3236	-1970	403	7579696	-
v.15.15 7	827	2408	3236	-1520	525	9868868	-
v.15.15 8	827	2408	3236	-560	1148	11576520	-
v.15.15 9	827	2408	3236	-415	1542	11830253	-
v.15.15 10	827	2408	3236	-525	1194	11576160	-
v.15.15 11	827	2408	3236	-1395	557	13103706	-
v.15.15 12	827	2408	3236	-1690	474	13837716	-
v.15.15 13	827	2408	3236	-945	651	12103779	-
v.15.15 14	827	2408	3236	-960	651	11965495	-
v.15.15 15	827	2408	3236	-975	609	11447030	-

Table 7.10: *Performance of* `Q+QCR` *on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

Name	*N*	*Const*	*Const SDP*	*Opt*	*Gap (%)*	*Nodes*	T_t
v.10.10 1	352	1008	1613	-810	259	24264617	-
v.10.10 2	352	1008	1613	-595	347	24358116	-
v.10.10 3	352	1008	1613	-280	551	23995725	-
v.10.10 4	352	1008	1613	-280	533	24118791	-
v.10.10 5	352	1008	1613	-265	557	24331395	-
v.10.10 6	352	1008	1613	-905	237	24777297	-
v.10.10 7	352	1008	1613	-1060	200	24609050	-
v.10.10 8	352	1008	1613	-200	783	23589203	-
v.10.10 9	352	1008	1613	-155	987	23934929	-
v.10.10 10	352	1008	1613	-140	1020	24462952	-
v.10.10 11	352	1008	1613	-660	320	24829708	-
v.10.10 12	352	1008	1613	-900	237	24342792	-
v.10.10 13	352	1008	1613	-355	367	25112545	-
v.10.10 14	352	1008	1613	-460	283	24806402	-
v.10.10 15	352	1008	1613	-455	273	27022983	-
v.10.15 1	542	1568	2503	-1135	292	17351911	-
v.10.15 2	542	1568	2503	-1390	241	18101388	-
v.10.15 3	542	1568	2503	-415	574	16702074	-
v.10.15 4	542	1568	2503	-515	484	16872548	-
v.10.15 5	542	1568	2503	-520	436	16694187	-
v.10.15 6	542	1568	2503	-1110	303	18004770	-
v.10.15 7	542	1568	2503	-1415	236	17801167	-
v.10.15 8	542	1568	2503	-240	1013	16618705	-
v.10.15 9	542	1568	2503	-285	872	16460247	-
v.10.15 10	542	1568	2503	-270	803	16525592	-
v.10.15 11	542	1568	2503	-1270	253	17507848	-
v.10.15 12	542	1568	2503	-1115	295	17563707	-
v.10.15 13	542	1568	2503	-645	343	16360888	-
v.10.15 14	542	1568	2503	-565	399	16288976	-
v.10.15 15	542	1568	2503	-665	285	16797284	-
v.15.15 1	827	2408	3838	-2155	237	11739426	-
v.15.15 2	827	2408	3838	-1630	312	11871919	-
v.15.15 3	827	2408	3838	-970	375	10796046	-
v.15.15 4	827	2408	3838	-760	491	10928145	-
v.15.15 5	827	2408	3838	-1000	345	10966658	-
v.15.15 6	827	2408	3838	-1970	265	11797979	-
v.15.15 7	827	2408	3838	-1520	345	11789411	-
v.15.15 8	827	2408	3838	-560	617	10741699	-
v.15.15 9	827	2408	3838	-415	822	10675188	-
v.15.15 10	827	2408	3838	-525	620	10912085	-
v.15.15 11	827	2408	3838	-1455	338	11466148	-
v.15.15 12	827	2408	3838	-1730	302	11554104	-
v.15.15 13	827	2408	3838	-945	333	10860798	-
v.15.15 14	827	2408	3838	-960	340	10952590	-
v.15.15 15	827	2408	3838	-975	297	11056701	-

Table 7.11: *Performance of* `PQCR 2` *on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

7.4.2 Experimental comparison between termwise and pairwise convexification

This section aims at commenting and analyzing the computational results obtained by termwise and pairwise quadratizations within the convexification framework defined for each method in Sections 5.2 and 5.3. We recall these methods:

- `PC+QCR` (Described in Section 5.2): We apply the `QCR` method after the quadratic reformulation of Anthony et al. [5] using the pairwise cover PC0.
- `T+QCR` (Described in Section 5.3): We apply the `QCR` method after the termwise quadratization of Boros et al. [15].

The comparison is based on small LABS instances and 10×10 and 15×15 Vision instances.

We start with the Vision instances which results are detailed in Tables 7.12 and 7.13. These instances are sparse but they contain more variables than LABS instances. This is why, none of the tested methods is able to solve any instance within one hour. The best bounds are obtained by `T+QCR` with an average gap of 306% against 518% for `PC+QCR`. However, the number of variables is bigger with an average of 855 variables against 590. The number of nodes for both methods is of the same order of magnitude, that is $\mathcal{O}(10^7)$. This number reflects the poor quality of the bounds as the branch-and-bound of `cplex` cannot prune a sufficient number of nodes.

In Tables 7.14 and 7.15, we present the results for small LABS instances. Unlike Vision instances, the LABS instances are smaller in terms of number of variables but they are much more denser with a huge number of monomials. As a consequence, `T+QCR` has much more variables as it introduces one additional variable for each monomial. Thus, although the bounds obtained are the best with an average gap of (402%), the number of variables is huge with an average of 1926 against 283 only for `PC+QCR`. Logically, this method failed to solve a single LABS instance. As for `PC+QCR`, it is able to solve 1 instance within the time limit. The gaps are similar to the Vision instances with an average of 517%.

For the convexification `PC+QCR`, note that the bounds are worse than `Q+QCR`. However, both methods use the valid equality $x_i^2 = x_i$ only. The difference of tractability can be explained on the one hand by noticing that the quadratization of [15] introduce a "hidden" penalty term by introducing additional monomials in the objective function. On the other hand, by applying the `QCR` method to this quadratization, the idea was to keep the constraint-less property. As for `Q+QCR`, not only does it not contain penalty terms but it also has the linearization constraints $\mathcal{F}_{\mathcal{E}}$ to enforce $y_{ij} = x_i x_j$. This seems to make the difference here. It is also interesting to note the performance of `T+QCR`. The bounds obtained are better than `PC+QCR` and `Q+QCR`. However, one should prefer this method for very sparse instances as termwise quadratizations are very sensitive to the number of monomials.

These results also highlights the choice of 2×2 quadratizations for the PQCR method. Indeed, thanks to this family of quadratizations, we are able to derive numerous valid equalities which highly improve the continuous relaxation bound values. As a result, within the considered quadratic convex reformulation framework, PQCR obtains the best experimental results. For the quadratizations considered here, there is no explicit links between initial and additional variables which makes it difficult to derive valid equalities other than $x^2 = x$. Future research directions would be to derive other kind of valid equalities for these state-of-the-art quadratizations. To conclude, these experimentations were aimed at illustrating the application of quadratic convex reformulation frameworks to state-of-the-art quadratization. The obtained results prove that these methods are not tractable in general. It also suggests that quadratizations using constraints can be better as they do not contain penalty terms that can alter the bound. Moreover they also lead to additional valid equalities and thus better bounds. As a result, one would rather choose the original PQCR algorithm, as expanding slightly the number of constraints highly improves the tractability.

Name	*N*	*BKN*	*Gap (%)*	*Nodes*	T_t
v.10.10 1	352	-810	286	28948477	–
v.10.10 2	352	-595	387	27330545	–
v.10.10 3	352	-280	669	29661973	–
v.10.10 4	352	-280	648	29897428	–
v.10.10 5	352	-265	681	28964929	–
v.10.10 6	352	-905	259	29661662	–
v.10.10 7	352	-1060	219	29735079	–
v.10.10 8	352	-200	945	29304165	–
v.10.10 9	352	-155	1207	28768728	–
v.10.10 10	352	-140	1267	29444380	–
v.10.10 11	352	-660	355	28813894	–
v.10.10 12	352	-900	258	29917453	–
v.10.10 13	352	-355	462	29466236	–
v.10.10 14	352	-460	351	29858728	–
v.10.10 15	352	-455	346	29401641	–
v.15.15 1	827	-2155	257	19188044	–
v.15.15 2	827	-1630	344	19379560	–
v.15.15 3	827	-970	450	19401033	–
v.15.15 4	827	-760	589	19349873	–
v.15.15 5	827	-1000	419	19466247	–
v.15.15 6	827	-1970	287	18619118	–
v.15.15 7	827	-1520	380	18796389	–
v.15.15 8	827	-560	757	19524808	–
v.15.15 9	827	-415	1021	18933569	–
v.15.15 10	827	-525	776	19410174	–
v.15.15 11	827	-1455	375	18220156	–
v.15.15 12	827	-1730	330	19037547	–
v.15.15 13	827	-945	416	19679357	–
v.15.15 14	827	-960	421	17420934	–
v.15.15 15	827	-975	379	19022822	–

Table 7.12: *Performance of* `PC+QCR` *using Pairwise Cover 0 on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

Name	*N*	*BKN*	*Gap (%)*	*Nodes*	T_t
v.10.10 1	505	-810	192	26101236	-
v.10.10 2	505	-595	259	25511303	-
v.10.10 3	505	-280	398	25652122	-
v.10.10 4	505	-280	373	25936300	-
v.10.10 5	505	-265	390	24326513	-
v.10.10 6	505	-905	174	22995208	-
v.10.10 7	505	-1060	143	25936617	-
v.10.10 8	505	-200	568	27531478	-
v.10.10 9	505	-155	732	26975910	-
v.10.10 10	505	-140	718	27236854	-
v.10.10 11	505	-660	246	26524840	-
v.10.10 12	505	-900	174	24896938	-
v.10.10 13	505	-355	254	26799355	-
v.10.10 14	505	-460	186	27401515	-
v.10.10 15	505	-455	175	27309292	-
v.15.15 1	1205	-2155	170	15797347	-
v.15.15 2	1205	-1630	226	15602283	-
v.15.15 3	1205	-970	253	15795763	-
v.15.15 4	1205	-760	350	15516077	-
v.15.15 5	1205	-1000	225	15991191	-
v.15.15 6	1205	-1970	193	15748388	-
v.15.15 7	1205	-1520	260	15315590	-
v.15.15 8	1205	-560	428	16082729	-
v.15.15 9	1205	-415	580	16085086	-
v.15.15 10	1205	-525	419	15731195	-
v.15.15 11	1205	-1455	251	15512693	-
v.15.15 12	1205	-1730	227	15268273	-
v.15.15 13	1205	-945	215	16441029	-
v.15.15 14	1205	-960	224	15784924	-
v.15.15 15	1205	-975	180	16686176	-

Table 7.13: *Performance of* `T+QCR` *on Vision instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

Name	*N*	*BKN*	*Gap (%)*	*Nodes*	T_t
b.20.5	65	-416	310	20255336	452
b.20.10	124	-2936	441	47323811	-
b.25.6	105	-960	404	48685587	-
b.25.13	206	-8148	488	36563196	-
b.25.19	265	-14644	597	33516200	-
b.25.25	289	-10664	679	33441895	-
b.30.4	82	-324	249	63623531	-
b.30.8	173	-2952	484	41274044	-
b.30.15	295	-15744	553	31382764	-
b.30.23	390	-30460	641	26603808	-
b.30.30	422	-22888	719	25066030	-
b.35.4	97	-384	255	39199517	-
b.35.9	233	-5108	520	33395749	-
b.35.18	417	-31160	586	26226133	-
b.40.5	144	-936	359	22764283	-
b.40.10	303	-8248	553	32192359	-
b.40.20	543	-50576	623	23499347	-

Table 7.14: *Performance of* **PC+QCR** *using Pairwise Cover 0 on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

Name	*N*	*BKN*	*Gap (%)*	*Nodes*	T_t
b.20.5	114	-416	239	32889233	-
b.20.10	529	-2936	361	24897180	-
b.25.6	254	-960	328	30810993	-
b.25.13	1248	-8148	403	19983753	-
b.25.19	2335	-14644	472	15494328	-
b.25.25	2912	-10664	512	11478226	-
b.30.4	126	-324	191	19931268	-
b.30.8	648	-2952	396	24152445	-
b.30.15	2197	-15744	463	11285716	-
b.30.23	4333	-30460	527	3526886	-
b.30.30	5289	-22888	544	3333009	-
b.35.4	151	-384	196	20091757	-
b.35.9	1011	-5108	439	20747980	-
b.35.18	3905	-31160	483	9016580	-
b.40.5	274	-936	283	29217808	-
b.40.10	1569	-8248	466	17305500	-
b.40.20	5845	-50576	527	1876860	-

Table 7.15: *Performance of* **T+QCR** *on small LABS instances. Time limit 1 hour for the branch-and-bound. Legend on Figure 7.3.*

7.5 Tightness of semi-definite relaxations for box-constrained polynomial programs (presented in Chapter 6)

More precisely, we compare the following methods:

- $(SDP^0_{\mathcal{E}})$: After a quadratization phase using PC0, we compute the compact relaxation described in Section 6.2. For this we use the `Conic Bundle algorithm` [40].
- $(SDP^1_{\mathcal{E}})$: After a quadratization phase using PC0, we compute the improved relaxation described in Section 6.3. For this we use the `Conic Bundle algorithm` [40].
- `SCIP 5.0.0` (described in Section 2.1): We submit the polynomial program (P) to this MINLP solver. The problem is then solved by successive linear relaxations and cutting planes methods. We use the default parameters.

7.5.1 Instance generation

In this section, we evaluate the bound obtained by our relaxations $(SDP^0_{\mathcal{E}})$ and $(SDP^1_{\mathcal{E}})$ on randomly generated instances of degree 4. The number of variables varies between 10 and 40 and the number of monomials from 10 to 400. These instances are quite sparse as their density, i.e. the ratio $\frac{m}{n}$, is between 1 and 10. The generation process is similar to [20], that is:

- the number of initial variables n is contained in $\{10, 15, 20, 25, 30, 35, 40\}$.
- the number of monomials m is a multiple of n.
- the coefficient of each monomial is uniformly generated in the interval $[-1, 1]$.
- the variables composing each monomial are generated by randomly choosing an index within $\{0, 1, ..., n\}$. Each time the value 0 is generated, the degree of the monomial decreases by one. An index can only be chosen once in a monomial.

7.5.2 Experimental results

We present in Table 7.16 a comparison of the lower bounds obtained by $(SDP^0_{\mathcal{E}})$ and $(SDP^1_{\mathcal{E}})$ after 20 minutes of CPU time, with the final lower bound obtained by the solver `SCIP` after one hour of CPU time. Each line represents one instance and the legend of this table is presented in Figure 7.4. As expected, we observe that the gap obtained with the relaxation $(SDP^1_{\mathcal{E}})$ is significantly smaller than the gap obtained with $(SDP^0_{\mathcal{E}})$. Indeed, out of the 70 considered instances, $(SDP^0_{\mathcal{E}})$ has an average gap of 64% whereas $(SDP^1_{\mathcal{E}})$ has an average gap of 33%. This shows the importance of the valid constraints added to $(SDP^1_{\mathcal{E}})$ that exploit both the quadratization used, and the structure of the instances. Comparing the gaps obtained by $(SDP^1_{\mathcal{E}})$ to those obtained by the

solver SCIP, we observe that SCIP is not able to provide better lower bounds than our relaxation on 21 instances even after one hour of branch-and-bound. The average gap for SCIP is about 27% over all the instances. In fact, SCIP is able to close the gap within one hour of CPU time for the smaller instances, but its gap significantly increases on medium sized instances and dense instances. On the contrary, the gap obtained with $(SDP^1_{\mathcal{E}})$ is more stable to the increase of n.

- n: number of variables in the polynomial formulation.
- N: number of variables after the quadratization.
- m: number of monomials.
- BKN: is the best known solution value at the end of the branch-and-cut of SCIP (1 hour).
- LB_{end}: is the lower bound obtained by the branch-and-cut of SCIP, after 1 hour.
- Gap_{end}: is the final optimality gap at the end of the branch-and-cut of SCIP, i.e. $Gap_{end} = \left|\frac{BKN - LB_{end}}{BKN}\right| * 100$.
- LB: is the lower bound obtained within 20 minutes of CPU time when solving $(SDP^0_{\mathcal{E}})$ and $(SDP^1_{\mathcal{E}})$.
- Gap: is the relative gap between BKN and LB for $(SDP^0_{\mathcal{E}})$ and $(SDP^1_{\mathcal{E}})$, i.e. $Gap = \left|\frac{BKN - LB}{BKN}\right| * 100$.

Figure 7.4: *Legend of Table 7.16*

Instance				SCIP		$SDP^0_{\mathcal{E}}$		$SDP^1_{\mathcal{E}}$	
n	N	m	BKN	LB_{end}	Gap_{end} (%)	LB	Gap (%)	LB	Gap (%)
10	17	10	-2,07	-2,07	0	-2,14	3,34	-2,13	3,19
10	26	20	-2,28	-2,28	0	-2,79	22,01	-2,68	17,20
10	25	30	-3,38	-3,38	0	-5,37	58,94	-4,74	40,38
10	34	40	-5,25	-5,25	0	-6,49	23,63	-5,76	9,70
10	36	50	-5,43	-5,43	0	-7,16	31,76	-6,31	16,03
10	38	60	-3,54	-3,54	0	-6,34	78,84	-4,65	31,29
10	46	70	-3,75	-3,75	0	-7,38	97,08	-4,88	30,20
10	39	80	-5,51	-5,51	0	-7,99	45,09	-5,97	8,46
10	44	90	-4,39	-4,39	0	-8,26	88,25	-5,33	21,53
10	50	100	-6,36	-6,36	0	-9,84	54,85	-6,73	5,82
15	29	15	-1,54	-1,54	0	-2,06	33,84	-1,98	28,90
15	33	30	-3,57	-3,57	0	-4,40	23,28	-3,87	8,43
15	59	45	-4,89	-4,89	0	-6,18	26,43	-5,58	14,04
15	64	60	-5,43	-5,43	0	-8,36	53,94	-7,09	30,64
15	75	75	-12,20	-12,20	0	-14,43	18,29	-12,80	4,88
15	77	90	-9,33	-9,33	0	-12,67	35,82	-10,44	11,93
15	85	105	-9,34	-9,34	0	-14,58	56,07	-11,17	19,53
15	85	120	-10,65	-10,65	0	-16,47	54,61	-12,05	13,13
15	87	135	-7,85	-7,85	0	-14,98	90,92	-10,56	34,57
15	98	150	-7,53	-9,31	23,65	-18,03	139,50	-11,13	47,78
20	27	20	-3,92	-3,92	0	-3,95	0,86	-3,98	1,46
20	73	40	-5,18	-5,18	0	-7,64	47,42	-6,80	31,23
20	95	60	-7,82	-7,82	0	-12,26	56,71	-10,57	35,15
20	101	80	-8,48	-8,48	0	-13,54	59,73	-10,73	26,63
20	116	100	-12,77	-12,77	0	-16,85	32,01	-14,47	13,35
20	115	120	-9,41	-10,64	13,11	-17,33	84,14	-13,43	42,76
20	138	140	-11,57	-12,54	8,44	-20,63	78,34	-15,27	32,04
20	139	160	-10,38	-13,28	27,97	-20,39	96,42	-13,91	34,04
20	143	180	-14,22	-19,82	39,35	-26,66	87,47	-19,09	34,27
20	157	200	-12,53	-16,37	30,64	-24,24	93,47	-16,44	31,19
25	51	25	-4,37	-4,37	0	-5,11	17,00	-4,99	14,35
25	79	50	-8,81	-8,81	0	-10,50	19,14	-9,77	10,92
25	118	75	-8,73	-8,73	0	-13,18	51,03	-11,50	31,82
25	147	100	-12,02	-12,02	0	-17,22	43,20	-14,15	17,73
25	165	125	-15,28	-17,92	17,28	-24,17	58,18	-19,62	28,42
25	165	150	-12,20	-14,16	16,04	-20,86	70,99	-16,01	31,22
25	197	175	-16,24	-22,98	41,50	-28,78	77,22	-22,24	36,96
25	188	200	-18,28	-28,22	54,36	-32,02	75,18	-24,62	34,67
25	217	225	-19,03	-28,81	51,38	-34,06	78,95	-25,45	33,72
25	213	250	-12,54	-25,50	103,33	-30,82	145,76	-21,13	68,49
30	43	30	-6,65	-6,65	0	-6,82	2,53	-6,73	1,09
30	103	60	-9,38	-9,38	0	-13,33	42,09	-12,19	29,93
30	150	90	-12,27	-12,27	0	-17,88	45,75	-16,09	31,14
30	181	120	-9,25	-13,04	40,96	-19,46	110,41	-15,86	71,47
30	216	150	-12,98	-16,35	25,99	-22,58	73,96	-18,35	41,40
30	237	180	-12,74	-22,31	75,11	-26,14	105,14	-20,69	62,37
30	260	210	-20,39	-31,54	54,68	-36,75	80,22	-28,47	39,62
30	266	240	-20,32	-35,89	76,60	-40,24	98,01	-30,04	47,81
30	274	270	-17,44	-35,58	104,04	-39,40	125,92	-28,23	61,87
30	292	300	-15,38	-38,68	151,52	-42,09	173,69	-28,30	84,00
35	59	35	-6,12	-6,12	0	-6,51	6,28	-6,49	6,05
35	130	70	-10,29	-10,29	0	-13,93	35,38	-13,33	29,52
35	182	105	-15,97	-15,97	0	-21,13	32,28	-19,28	20,73
35	220	140	-15,36	-20,68	34,64	-25,41	65,42	-22,21	44,62
35	252	175	-15,76	-22,67	43,83	-28,59	81,39	-23,83	51,21
35	275	210	-20,81	-27,07	30,09	-31,29	50,35	-26,71	28,38
35	304	245	-22,83	-36,02	57,80	-39,64	73,63	-32,09	40,55
35	344	280	-29,01	-47,36	63,26	-48,13	65,90	-39,39	35,77
35	347	315	-27,09	-46,81	72,80	-48,66	79,61	-38,42	41,84
35	369	350	-34,13	-55,09	61,41	-57,00	67,01	-45,27	32,63
40	85	40	-4,22	-4,22	0	-5,00	18,32	-4,86	15,04
40	139	80	-9,50	-9,50	0	-11,98	26,05	-11,61	22,15
40	215	120	-11,19	-13,28	18,70	-18,23	62,93	-16,38	46,34
40	272	160	-19,26	-22,46	16,59	-27,07	40,54	-23,98	24,50
40	300	200	-23,04	-34,49	49,69	-38,22	65,89	-33,09	43,61
40	327	240	-22,25	-37,66	69,25	-40,37	81,42	-34,44	54,80
40	375	280	-22,29	-38,30	71,81	-41,20	84,85	-33,80	51,62
40	397	320	-26,01	-51,03	96,21	-51,72	98,85	-44,38	70,63
40	436	360	-25,29	-54,27	114,60	-53,90	113,14	-47,40	87,43
40	454	400	-24,68	-62,84	154,61	-61,88	150,72	-53,97	118,68

Table 7.16: *Comparison of the lower bounds of $SDP^0_{\mathcal{E}}$ and $SDP^1_{\mathcal{E}}$ with the lower bounds obtained by the branch-and-cut of* `SCIP` *in one hour of CPU time. Legend on Figure 7.4*

Chapter 8

Conclusion

Contents

8.1 Main contributions **139**

8.2 Perspectives and research directions **142**

This thesis was prepared in Ecole Nationale Supérieure des Techniques Avancées (ENSTA ParisTech) in Palaiseau and Conservatoire National des Arts et Métiers (CNAM) in Paris thanks to a scholarship granted by École Doctorale Mathématiques Hadamard (EDMH). It has also widely benefited from the research stays in HEC Liège and RWTH Aachen, thanks to mobility grants obtained through the GDR RO foundation.

In the following we recall the main contributions and we outline some future improvements of the work presented.

8.1 Main contributions

This dissertation is concerned with polynomial optimization with mixed-integer variables. The major part of our work concerns unconstrained binary and box-constrained polynomial optimization problems. The recent advances in this area have led to a renewed interest and various thesis, articles, tutorials and solvers have been published in the last decade. These advances concerned theoretical as well as practical materials. In this thesis we focus on both. The main practical part corresponds to implementing `PQCR`, our main solver for unconstrained binary polynomial programs. This solver can be efficient on very hard instances such as the LABS instances (1.2.2). The computational experiments in Section 7 confirms that `PQCR` performs better on these instances than the other available solvers. We also implemented linearization and direct convexification methods. Finally, we implemented a relaxation computation framework for box-constrained programs. On the other hand, the theoretical part of this thesis concerns the designing of all the

previous methods. Moreover, we also provide detailed theoretical comparisons between our methods and those of the literature.

One-step reformulation methods Chapter 3 presents one-step reformulation algorithms for exact solution of binay polynomial programs. It begins with the introduction of a special framework called $q-$linearization. This framework regroups all the linearizations that introduce additional variables and use reformulation constraints. The main result here concerns the continuous relaxation bound values of such linearizations. We prove that when linearizing each monomial separately, for all q, any $q-$linearization lead to the same bound. We further show the connection with the $2-$links cuts proposed in [24]. Indeed, reusing additional variables in several monomials amounts to adding some of these inequalities.

We then discuss about direct convexification. Starting from problem (P), we compute an interval matrix that approximates the non-constant Hessian matrix of f. Then we add the null function $x \mapsto \lambda \sum_i (x_i^2 - x_i)$ to f. Using the interval matrix, we provide value for real parameter λ such that the new objective function is convex. Next we consider a vector parameter α and we add to f the null function $x \mapsto \sum_i \alpha_i (x_i^2 - x_i)$. Similarly, we provide values for α in order to attain convexity.

Quadratic convex reformulation methods Chapter 4 addresses our main resolution algorithm, that is quadratic convex reformulation. The `PQCR` method is a $3-$phases algorithm with a two-steps reformulation. We first compute a quadratic reformulation (QP) of (P) by using additional variables and linearization constraints. Then, after presenting standard convexification methods, we introduce the set $S_{\mathcal{E}}$ of null functions. These functions are used to build a new parameterized objective function $g_{\alpha,\beta,\delta,\lambda}$. We then characterize optimal values of $\alpha, \beta, \delta, \lambda$ such that, the new objective function is convex and the continuous relaxation bound value is maximized. Moreover, we show that one can extract these optimal values from a semi-definite relaxation of (QP). Finally, the resulting quadratic convex continuous relaxation is then solved at each node of a branch-and-bound algorithm.

However, our "best" convexification is dependent on the quadratization phase and from one quadratization to another, the bounds may vary. We thus study the quadratization phase more thoroughly. We prove that, given a quadratization defined by $\mathcal{E}$, all the possible quadratic objective functions lead to the same bound after convexification. Then we show that adding auxiliary variables to a quadratization cannot worsen the bound. Finally we present the 2×2 full quadratization, which consists in reformulating every product of two variables. This allows us to relate `PQCR` with the Moment/SOS method. We also introduce the partial quadratization, which consists in reformulating products of variables appearing in the same monomial. We show that we obtain nice numerical results with this quadratization.

Finally, we establish the link between the PQCR framework and the Lasserre's hierarchy. More precisely, we state that the semi-definite relaxation obtained after applying the full quadratization is the same as the first order hierarchy of the Moment/SOS method. Then we show that at each order of the hierarchy, we can derive a quadratic convex reformulation to (P) with the same bound as the semi-definite relaxation of the considered order. We also show that the PQCR method considers partial orders of the hierarchy for unconstrained binary polynomial programs, i.e. not every product of degree less or equal to a given degree needs to be reformulated. This allows to reduce the number of variables and to solve larger instances.

Convex reformulation with other quadratization schemes Chapter 5 discusses the impact of other quadratization families within the PQCR framework. We begin by studying the quadratizations using penalty terms such as Rosenberg's procedure. We prove that applying this quadratization within our framework amounts to apply the QCR algorithm after the quadratization phase. We then show that we can derive a new convexification by using the penalty term as a null function.

We then study different quadratization schemes (called pairwise covers) and measure their impact on the number of variables and the continuous relaxation bound value. We then derive a convexification phase for the types of quadratizations described in [5]. Similarly to Rosenberg's procedure, these quadratizations use penalty terms without any constraints. We show that it is possible to preserve this "constraintless" structure by only perturbing diagonal terms by adding the null function $x \mapsto \sum_i \alpha_i(x_i^2 - x_i)$. We finally apply this diagonal perturbation on termwise quadratizations. We further compare the results obtained by these methods.

Semi-definite relaxations for box-constrained polynomial programs Chapter 6 is concerned with more general classes of polynomial programs. We compute semi-definite relaxations for box-constrained polynomial programs. We build a first compact relaxation $(SDP^0_{\mathcal{E}})$ with $\mathcal{O}(N^2)$ variables and $\mathcal{O}(N)$ constraints. Then we build an improved relaxation $(SDP^1_{\mathcal{E}})$ by adding more constraints corresponding to valid inequalities. It has $\mathcal{O}(N^4)$ constraints.

Summary and comparison of the methods presented in this thesis Finally, we end this thesis by illustrating the links between the different methods introduced in this manuscript. We propose a theoretical comparison in Figure 8.1, as well as an experimental comparison in Figure 8.2. We remark that PQCR dominates every other method both theoretically and experimentally. Indeed, we know by construction that PQCR has the best continuous relaxation bounds and the computational results confirms that PQCR is able to solve the highest number of instances among all the considered methods. Let us also remark that, even though PQCR performs better than the

linearization methods, there is no proof that the bound obtained by PQCR is always better than the one obtained by linearization. For the same reasons, the $q-$linearizations cannot be compared with other families of quadratic convex reformulation.

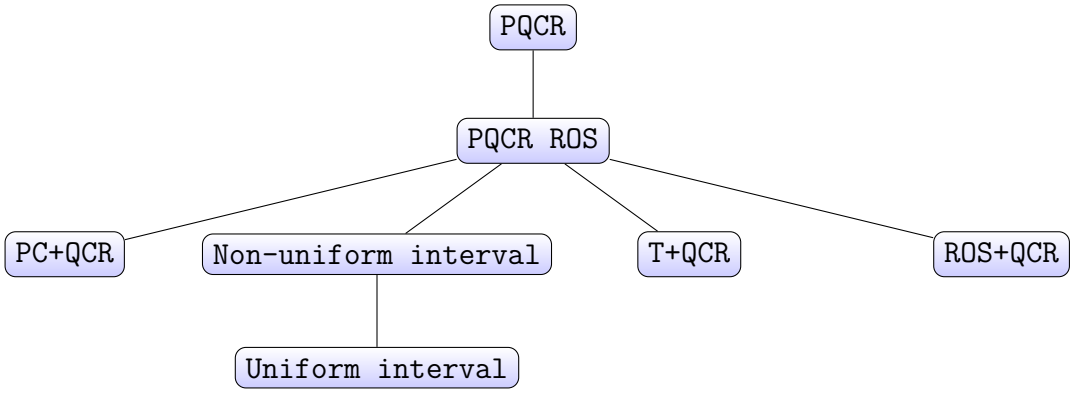


Figure 8.1: *Theoretical comparison of the different methods. A vertical link means that the method placed at the top is proven to have a better bound than the one at the bottom.*

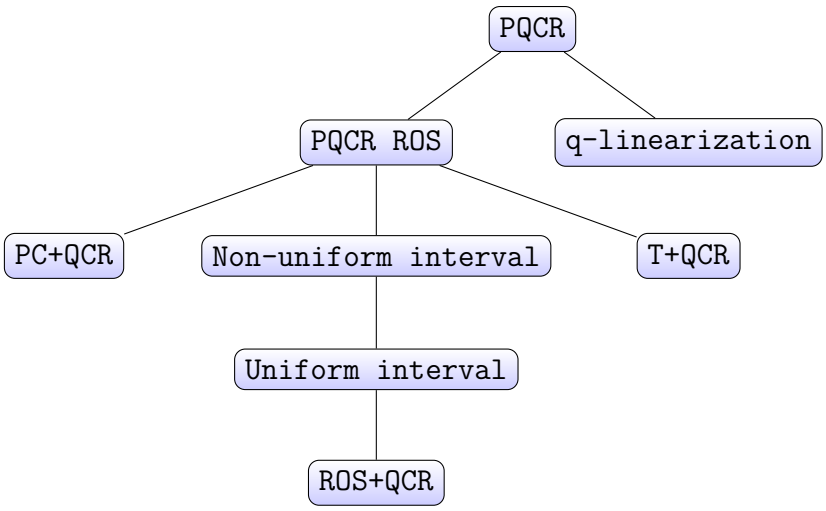


Figure 8.2: *Experimental comparison of the different methods. A vertical link means that the method placed at the top obtains shorter CPU times than the one at the bottom on the instances considered in this thesis.*

8.2 Perspectives and research directions

In this thesis we have addressed unconstrained binary polynomial problems and relaxation for the continuous case. Other classes of problems could be handled by our quadratic convex reformulation framework. In the next paragraphs, we give some suggestions to extend our approach to more general class of problems.

Linearly constrained binary polynomial optimization Throughout our studies we did not consider constrained polynomial programs. We only focused on unconstrained programs. The work we have done within the PQCR framework allows us to compute exact reformulation for the linearly constrained case. In this case, one can directly apply the PQCR method. Indeed, linear constraints are already taken into account in the same way as constraints $\mathcal{F}_{\mathcal{E}}$. They are added to the equivalent quadratic convex program as well as the semidefinite relaxation without any change.

Polynomially constrained binary polynomial optimization We now consider the case of polynomially constrained binary polynomial program (P_P). We can perform a first phase of quadratic reformulation on the objective function by applying PQCR on the relaxation of (P_P) obtained by removing its constraints. We thus get a polynomial program with a quadratic objective function. Then, one can apply any $q-$linearization to the polynomial constraints. For example, we can apply the $q-$linearization that reuses the maximum number of additional variables introduced in the convexification phase.

For quadratic programs, the authors of [11] derive a convex reformulation method. They also prove that linearizing quadratic constraints suffices to get the best continuous relaxation bound values within their framework. It would be interesting to study whether similar properties can be applied to constrained binary polynomial programs.

General integer polynomial optimization In this thesis, we discussed polynomial programs with both binary and continuous variables. One can extend this approach to general integer variables. We follow the ideas presented in [49]. Let $0 \leq x \leq u$ be an integer variable. By introducing $\lfloor \log(u) \rfloor + 1$ binary variables t_i, observe that $x = \sum_{i=0}^{\lfloor \log(u) \rfloor} 2^i t_i$. Thus by replacing each integer variable by its binary decomposition, each product of integer variables $x_i x_j$ is reduced to several products of binary variables. Thus our quadratic convex reformulation framework presented in Section 4.1 can be applied. However, this may not be tractable as the number of additional binary variables is huge. Indeed, for n integer variables such that $0 \leq x_i \leq u_i$, the number of additional binary variables is $\mathcal{O}(\sum_{i=0}^{\lfloor \log(u_i) \rfloor^2})$. This method was tractable for the case of quadratic programs but since we introduce other additional variables for the quadratization phase, the number of variables becomes too restrictive.

Semi-definite relaxations for mixed-integer polynomial programs In this paragraph, we give some clues to compute tight semi-definite relaxations for general mixed-integer polynomial programs. Here we focus on unconstrained programs, as we have established that the constraints can be handled by an additional linearization phase. Moreover, we can also consider mixed-binary variables without loss of generality, as integer variables can be reduced to binary ones thanks to the binary decomposition. Let (P_{MB}) be an unconstrained mixed-binary program. We can derive a first quadratic reformulation of (P_{MB}) by successively reformulating products

of variables with a new one by using a pairwise cover. We then obtain a constrained quadratic program. The constraints establish the equality between an additional variable t_{ij} and the product of two variables that it represents. In addition to these reformulation constraints, we can add other constraints to tighten the further semi-definite relaxation. We make the distinction between three cases:

- t_{ij} is a product of 2 binary variables (i.e. $t_{ij} = x_i x_j$): We add the set of constraints $S_{\mathcal{E}}$ defined in Lemma 4.1.1.
- t_{ij} is a product of 2 continuous variables (i.e. $t_{ij} = y_i y_j$): We add the set of constraints defined in Chapter 6 for the continuous case.
- t_{ij} is a product of a continuous variable by a binary one (i.e. $t_{ij} = y_i x_j$): We add the following set of linear constraints

$$\begin{cases} t_{ij} \leq x_j, \\ t_{ij} \geq 0, \\ t_{ij} \leq y_i, \\ t_{ij} \geq y_i - (1 - x_j), \\ 0 \leq t_{ij} \leq 1 \end{cases}$$

From this quadratic program, we can compute a semi-definite relaxation by linearizing these constraints in the space of semi-definite matrices, as it was done in Chapter 6.

Appendix A

The PQCR solver

The PQCR solver (Polynomial optimization in binary variables through Quadratic Convex Reformulation) solves unconstrained binary polynomial optimization problems to optimality and is able to derive strong convex relaxations on continuous problems. The solver is not available as an open-source software yet since it is still in development phase. Various methods are implemented to solve (P). All the generic models are written in AMPL [29] and all the preprocessing is coded in C. All the solvers presented hereunder admits a boolean parameter relative to the symmetry of the problem: 1 if the problems admits a symmetry that allows to remove a variable, 0 otherwise. Moreover, a first phase of presolve is performed commonly for all the algorithms in order to remove the "trivial" variables, i.e. variables that appear only in negative or positive monomials.

A.1 Implemented algorithms

- $q-$linearizations: the standard linearization presented in Section 3.1 as well as other benchmark linearizations are implemented. The resulting linear program is solved with Cplex
- Direct convexification: both the convexification using uniform and non-uniform diagonal perturbation presented in Section 3.2 are implemented. The resulting polynomial program is solved by Ipopt [80] or SCIP [79].
- Smallest eigenvalue convexification. After the quadratization of (P), the smallest eigenvalue of the new Hessian is computed in Matlab. The resulting quadratic convex reformulation is solved by Cplex.
- PQCR the original algorithm presented in this thesis. For tractability issues, a second parameter is required by PQCR , which is the number of families of valid equalities added from $S_{\mathcal{E}}$ between 0 and 4. A quadratization is first chosen among the available ones, this point is discussed hereunder. Then the semidefinite relaxation (SDP) is solved with CSDP [14]. The

dual variables are then recovered. Finally the equivalent quadratic convex reformulation (QP^*) is solved by `Cplex`.

- `PQCR` using the Conic Bundle algorithm: this version of `PQCR` relies on the Conic Bundle algorithm [40]. It consists in solving iteratively a relaxed version of the quadratic convex reformulation by dynamically adding constraints to it. The resulting program is then solved by `Cplex`.

- Semidefinite relaxation for continuous programs. This algorithm presented in Chapter 6 computes tight relaxations for unconstrained continuous polynomial programs. After the quadratization phase, the semidefinite relaxation is solved by `Csdp`.

A.2 Quadratizations

`PQCR` computes a quadratic convex reformulation given any quadratization. We have coded a sample of quadratizations that we used in this thesis.

- Random 2×2 quadratization. This quadratization reformulates random product of two variables with a new one until the function is quadratic.

- Partial and Full quadratization described in Section 4.2. Additional variables are iteratively added.

- "Rosenberg-like" quadratizations. Any quadratization using penalty terms can be used for `PQCR`. The parameter is the value of the penalty term P.

- "Lasserre quadratization". In section 4.3.1, we established that for any order r of the Lasserre's hierarchy, there exist a quadratization such that we can compute an equivalent quadratic convex reformulation with the same bound. These quadratizations satisfy this property. The code requires an integer r representing the order of the hierarchy and the quadratization in question will be computed.

A.3 Instances

`PQCR` reads a certain format of instance and can also randomly generate instances.

Format For a binary polynomial program of the form

$$(P)\begin{cases} \min \sum_{m=1}^{M} c_m \prod_{j=1}^{d_m} x_j \\ \text{s.t.} \\ \quad x \in \{0,1\}^n \end{cases}$$

The corresponding instance file for `PQCR` is:
n
b l_i u_i (where l_i and u_i are 0 and 1 for the binary case)
M
c_m d_m j (where j are the indices of the variables in the monomial m)

Random instances `PQCR` generates random instances of (P). These instances are rather purely continuous or purely binary. The program takes 8 parameters, namely n, m, c_l, c_u, d, t, l, u where:

- n is the number of variables.
- m is the number of monomials.
- c_l and c_u are the lower and upper bounds of the coefficients.
- d is the degree of the polynomial. For each monomial, the coefficient and the degree are randomly chosen, respectively in $[c_l, c_u]$ and between 1 and d. The variables composing each monomial are generated by randomly choosing an index within $\{1, ..., n\}$. A same variable cannot appear more than once in a monomial for the special case of binary variables.
- t is the type of the variables, "c" for continuous and "i" for integer.
- l and u are the upper and lower bounds for the variables. Once the bounds are fixed, a random number is chosen in $[l, u]$ for each variable. If the variables are integers then the random numbers are rounded.

Both the format of instance and the generation process are similar to [20]. A random generator for mixed-integer programs is yet to develop.

Bibliography

[1] T. Achterberg. Scip : solving constraint integer programs. *Mathematical Programming Computation*, (1):1–41, 2009.

[2] C.S. Adjiman, S. Dallwig, C.A. Floudas, and A. Neumaier. A global optimization method, αbb, for general twice-differentiable constrained nlps—i. theoretical advances. *Computers and Chemical Engineering*, 22(9):1137–1158, 1998.

[3] A. A. Ahmadi and A. Majumdar. Dsos and sdsos optimization: Lp and socp-based alternatives to sum of squares optimization. In *2014 48th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–5, March 2014.

[4] M.F. Anjos and J.B. Lasserre. Handbook of semidefinite, conic and polynomial optimization: Theory, algorithms, software and applications. *International Series in Operational Research and Management Science*, 166, 2012.

[5] M. Anthony, E. Boros, Y. Crama, and A. Gruber. Quadratic reformulations of nonlinear binary optimization problems. *Mathematical Programming*, 162:115–144, 2017.

[6] B. Balasundaram and A.0. Prokopyev. On characterization of maximal independent sets via quadratic optimization. 19, 06 2011.

[7] M. Baratto. *Enveloppe convexe de la linéarisation d'une fonction pseudo-booléenne*. Master thesis, Université de Liège, Liège, Belgique, 2018.

[8] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex minlp. *Optimization Methods and Software*, 4–5(24):597–634, 2009.

[9] J. Bernasconi. Low autocorrelation binary sequences: statistical mechanics and configuration space analysis. *J. Physique*, 141(48):559–567, 1987.

[10] A. Billionnet and S. Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. *Mathematical Programming*, 109(1):55–68, 2007.

[11] A. Billionnet, S. Elloumi, and A. Lambert. Exact quadratic convex reformulations of mixed-integer quadratically constrained problems. *Mathematical Programming*, 158(1):235–266, 2016.

[12] A. Billionnet, S. Elloumi, A. Lambert, and A. Wiegele. Using a Conic Bundle method to accelerate both phases of a Quadratic Convex Reformulation. *INFORMS Journal on Computing*, 29(2):318–331, 2017.

[13] A. Billionnet, S. Elloumi, and M. C. Plateau. Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation: The QCR method. *Discrete Applied Mathematics*, 157(6):1185 – 1197, 2009. Reformulation Techniques and Mathematical Programming.

[14] B. Borchers. CSDP, A C Library for Semidefinite Programming. *Optimization Methods and Software*, 11(1):613–623, 1999.

[15] E. Boros, Y. Crama, and E. Rodríguez-Heck. Quadratizations of symmetric pseudo-Boolean functions: sub-linear bounds on the number of auxiliary variables. In *ISAIM*, Fort Lauderdale, 2018. ISAIM, International Symposium on Artificial Intelligence and Mathematics. Available at http://isaim2018.cs.virginia.edu/.

[16] E. Boros, A. Fix, A. Gruber, and R. Zabih. A hypergraph-based reduction for higher-order binary Markov random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(7):1387–1395, 2015.

[17] E. Boros and A. Gruber. On quadratization of pseudo-Boolean functions. In *ISAIM*, Fort Lauderdale, 2012. ISAIM, International Symposium on Artificial Intelligence and Mathematics. Open Source: arXiv:1404.6538v1.

[18] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1):155 – 225, 2002.

[19] E. Boros, P.L. Hammer, and X. Sun. Network flows and minimization of quadratic pseudo-boolean functions. Technical Report TR: 1991-17, RUTCOR, 1991.

[20] C. Buchheim and C. D'Ambrosio. Monomial-wise optimal separable underestimators for mixed-integer polynomial optimization. *Journal of Global Optimization*, pages 1–28, 2016.

[21] C. Buchheim and G. Rinaldi. Efficient reduction of polynomial zero-one optimization to the quadratic case. *SIAM Journal on Optimization*, 18(4):1398–1413, 2007.

[22] S. Cafieri, J. Lee, and L. Liberti. On convex relaxations of quadrilinear terms. *Journal of Global Optimization*, 47(4):661–685, Aug 2010.

[23] M.W. Carter. The indefinite zero-one quadratic problem. *Discrete Applied Mathematics*, pages 23–44, 1984.

[24] Y. Crama and E. Rodríguez-Heck. A class of valid inequalities for multilinear 0-1 optimization problems. *Discrete Optimization*, pages 28–47, 2017.

[25] E. Dalkiran and L. Ghalami. On linear programming relaxations for solving polynomial programming problems. *Computers and Operations Research*, 99:67 – 77, 2018.

[26] E. Dalkiran and H.D. Sherali. Rlt-pos: Reformulation-linearization technique-based optimization software for solving polynomial programming problems. *Mathematical Programming Computation*, 8(3):337–375, Sep 2016.

[27] A. Del Pia and A. Khajavirad. The multilinear polytope for acyclic hypergraphs. *SIAM Journal on Optimization*, 28(2):1049–1076, 2018.

[28] R. Fortet. L'algèbre de Boole et ses Applications en Recherche Opérationnelle. *Cahiers du Centre d'Etudes de Recherche Opérationnelle*, 4:5–36, 1959.

[29] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* The Scientific Press (now an imprint of Boyd & Fraser Publishing Co.), Danvers, MA, USA, 1993.

[30] D. Freedman and P. Drineas. Energy minimization via graph cuts: settling what is possible. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2:939–946 vol. 2, 2005.

[31] M.R. Garey and D.S. Johnson. Computers and Intractability: A guide to the theory of NP-Completness. *W.H. Freeman, San Francisco, CA*, 1979.

[32] S. Gershgorin. Uber die abgrenzung der eigenwerte einer matrix. *Bulletin de l'Académie des Sciences de l'URSS.*, 6:749–754, 1931.

[33] B. Ghaddar, J. C. Vera, and M. F. Anjos. A dynamic inequality generation scheme for polynomial programming. *Mathematical Programming*, 156(1):21–57, Mar 2016.

[34] F. Glover and E. Woolsey. Further reduction of zero-one polynomial programs to zero-one linear programming, 1973.

[35] F. Glover and E. Woolsey. Converting the 0-1 polynomial programming problem to a 0-1 linear program, 1974.

[36] F. W. Glover and G. A. Kochenberger. A tutorial on formulating QUBO models. *CoRR*, abs/1811.11538, 2018.

[37] C. E. Gounaris and C. A. Floudas. Tight convex underestimators for $\mathcal{C}^2$-continuous problems: II. multivariate functions. *Journal of Global Optimization*, 42(1):69–89, Sep 2008.

[38] P.L. Hammer, I. Rosenberg, and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas*. Springer, New York, 1968.

[39] P.L. Hammer and A.A. Rubin. Some remarks on quadratic programming with 0-1 variables. *Revue Française d'Informatique et de Recherche Opérationnelle*, 4:67–79, 1970.

[40] C. Helmberg. *Conic Bundle v0.3.10*, 2011.

[41] D. Henrion and J. B. Lasserre. Gloptipoly: Global optimization over polynomials with matlab and sedumi. *ACM Transactions on Mathematical Software*, 29(2):165–194, 2003.

[42] IBM-ILOG. IBM ILOG CPLEX 12.7 Reference Manual. "`http://www-01.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html`", 2017.

[43] H. Ishikawa. Transformation of general binary MRF minimization to the first-order case. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(6):1234–1249, June 2011.

[44] N. Ito, S. Kim, and M. Kojima nad A.Takeda K.C. Toh. BBCPOP: A Sparse Doubly Nonnegative Relaxation of Polynomial Optimization Problems with Binary, Box and Complementarity Constraints. *ArXiv e-prints*, April 2018.

[45] R.M. Karp. Reducibility among combinatorial problems. pages 85–103, 1972.

[46] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, Feb 2004.

[47] J. Krarup and P.M. Pruzan. Computer-aided layout design. pages 75–94, 1978.

[48] X. Kuang, , B. Ghaddar, J . Naoum-Sawaya, and L.F. Zuluaga. Alternative SDP and SOCP Approximations for Polynomial Optimization. *ArXiv e-prints*, October 2015.

[49] A. Lambert. *Résolution de programmes quadratiques en nombres entiers*. Thèse de doctorat en informatique, Conservatoire National des Arts et Métiers, Paris, 2009.

[50] J.B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.

[51] J.B. Lasserre. An explicit equivalent positive semidefinite program for nonlinear 0-1 programs. *SIAM Journal on Optimization*, 12(3):756–769, 2002.

[52] J.B. Lasserre. *An Introduction to Polynomial and Semi-Algebraic Optimization*. Cambridge University Press, Cambridge, 2015.

[53] J.B. Lasserre and T.P. Thanh. Convex underestimators of polynomials. *Journal of Global Optimization*, pages 1–25, 2013.

[54] J.D. Laughhunn. Quadratic binary programming with applications to capital budgeting problems. 18:454–461, 06 1970.

[55] M. Laurent. A comparison of the sherali-adams, lovász-schrijver, and lasserre relaxations for 0–1 programming. *Mathematics of Operations Research*, 28(3):470–496, 2003.

[56] C. Lemarechal and F. Oustry. Semidefinite relaxations and lagrangian duality with application to combinatorial optimization. Technical report, RR-3710, INRIA Rhones-Alpes, 1999.

[57] A. Lundell, J. Westerlund, and T. Westerlund. Some transformation techniques with applications in global optimization. *Journal of Global Optimization*, 43(2):391–405, Mar 2009.

[58] C. D. Maranas and C. A. Floudas. Global optimization in generalized geometric programming. *Computers & Chemical Engineering*, 21(4):351 – 369, 1997.

[59] R.D. McBride and J.S. Yormark. An implicit enumeration algorithm for quadratic integer programming. *Management Science*, 1980.

[60] G.P. McCormick. Computability of global solutions to factorable non-convex programs: Part i - convex underestimating problems. *Mathematical Programming*, 10(1):147–175, 1976.

[61] C. A. Meyer and C. A. Floudas. Trilinear monomials with mixed sign domains: Facets of the convex and concave envelopes. *Journal of Global Optimization*, 29(2):125–155, Jun 2004.

[62] C. A. Meyer and C. A. Floudas. Convex underestimation of twice continuously differentiable functions by piecewise quadratic perturbation: Spline αbb underestimators. *Journal of Global Optimization*, 32(2):221–258, Jun 2005.

[63] MINLPLIB. Library of mixed integer non linear programs. "`http://www.gamsworld.org/minlp/minlplib.htm`", 2012.

[64] R. Misener and C.A. Floudas. Antigone: algorithms for continuous/integer global optimization of nonlinear equations. *Journal of Global Optimization*, 59(2-3):503–526, 2014.

[65] K. Murty and S. Kabadi. Some np-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39:117–129, 1987.

[66] M. Padberg. The boolean quadric polytope: some characteristics, facets and relatives. *Mathematical programming*, 45(1):139–172, 1989.

[67] M. Padberg and M.J. Wilczak. Boolean polynomials and set functions. *Mathematical and computer modelling*, 17(9):3–6, 1989.

[68] P.M Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328, Apr 1994.

[69] M.R. Rao. Cluster analysis and mathematical programming. *Journal of the American Statistical Association*, 66(335):622–626, 1971.

[70] J.M.W. Rhys. A selection problem of shared fixed costs and network flows. *Management Science*, 17(3):200–207, 1970.

[71] E. Rodríguez-Heck. *Linear and quadratic reformulations of nonlinear optimization problems in binary variables*. PhD thesis, Université de Liège, 2018.

[72] I. G. Rosenberg. Reduction of bivalent maximization to the quadratic case. *Cahiers du Centre d'Études de Recherche Opérationnelle*, 17:71–74, 1975.

[73] N.V. Sahinidis and M. Tawarmalani. Baron 9.0.4: Global optimization of mixed-integer nonlinear programs. *User's Manual*, 2010.

[74] H.D. Sherali and W.P. Adams. A hierarchy of relaxation between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal Discrete Mathematics*, 3:411–430, 1990.

[75] H.D. Sherali and C.H. Tuncbilek. A global optimization algorithm for polynomial programming using a reformulation-linearization technique. *Journal of Global Optimization*, 2:101–112, 1992.

[76] C. De Simone. The cut polytope and the boolean quadric polytope. *Discrete Mathematics*, 79(1):71 – 75, 1990.

[77] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *OPTMS*, 11-12:625–653, 1999.

[78] M. Tawarmalani and N. V. Sahinidis. Convex extensions and convex envelopes of l.s.c. functions. math. program. 93, 247-263. *Mathematical Programming*, 93:247–263, 12 2002.

[79] S. Vigerske and Ambros G. Scip: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. *Optimization Methods and Software*, pages 1–31, 2017.

[80] A. Wächter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, Mar 2006.

[81] L. J. Watters. Letter to the editor—reduction of integer polynomial programming problems to zero-one linear programming problems. *Operations Research*, 15(6):1171–1174, 1967.

[82] W. I. Zangwill. *Media Selection by Decision Programming*, pages 132–133. Springer Berlin Heidelberg, Berlin, Heidelberg, 1976.

université PARIS-SACLAY
École doctorale de mathématiques Hadamard (EDMH)

Titre : Optimisation globale de programmes polynomiaux en variables mixtes-entières

Mots Clefs : Optimisation non convexe, Optimisation discrète, Optimisation polynomiale, Résolution exacte, Reformulation quadratique convexe

Résumé : Dans cette thèse, nous nous intéressons à l'étude des programmes polynomiaux. Ces problèmes ont de nombreuses applications pratiques et constituent actuellement un champ de recherche très actif, mais restent très difficiles et on ne sait résoudre en toute généralité que des instances de petite taille. Dans la majeure partie de ce manuscrit, nous étudions les problèmes d'optimisation en variables binaires. Nous proposons plusieurs reformulations convexes pour ces problèmes. Nous nous intéressons tout d'abord aux linéarisations en introduisant le concept de $q-$*linéarisation*. Ensuite, nous appliquons une reformulation convexe au problème polynomial. Nous étendons ensuite la reformulation quadratique convexe au cas polynomial. Nous proposons plusieurs nouvelles reformulations que nous comparons aux méthodes existantes sur des instances de la littérature. En particulier nous présentons la méthode `PQCR` pour les problèmes polynomiaux binaires sans contrainte qui permet de résoudre des instances jusqu'ici non résolues. Nous proposons aussi une étude théorique visant à comparer les différentes reformulations quadratiques de la littérature puis à leur appliquer une reformulation convexe. Enfin nous considérons des cas plus généraux et nous proposons une méthode permettant de calculer des relaxations convexes pour des problèmes en variables continues.

Title : Global optimization of polynomial programs with mixed-integer variables

Keys words : Non convex optimization, Discrete optimization, Polynomial optimization, Exact resolution, Convex quadratic reformulation

Abstract : In this thesis, we are interested in the study of polynomial programs. These problems have many practical applications and are currently actively studied, but they remain very difficult and only small instances are addressed. In most of this manuscript, we study optimization problems with binary variables. We propose several convex reformulations for these problems. We first focus on linearizations by introducing the concept of $q-$*linearization*. Then, we apply convex reformulation to the polynomial problem. We then extend quadratic convex reformulation to the polynomial case. We propose several new reformulations that we compare to existing methods on instances of the literature. In particular we present method `PQCR` for unconstrained binary polynomial problems, which is able to solve several unsolved instances. We also propose a theoretical study to compare the different quadratic reformulations of the literature and then apply a convex reformulation to them. Finally, we consider more general problems and we propose a method to compute convex relaxations for problems with continuous variables.