# Column generation methods for quadratic mixed binary programming

Enrico Bettiol

Université Paris 13 - Université Paris Nord
Laboratoire d'Informatique de Paris Nord - LIPN
Équipe: Algorithmes et Optimisation Combinatoire - AOC

Thèse de doctorat

# Column generation methods for quadratic mixed binary programming

présentée par

Enrico Bettiol

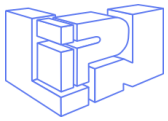
à l'École Doctorale Galilée

pour obtenir le grade de
Docteur d'Université
Spécialité: Informatique

soutenue publiquement devant le jury composé de:

| | | |
|---|---|---|
| Lucas Létocart | Université Paris 13 | Directeur de thèse |
| Emiliano Traversi | Université Paris 13 | Co-encadrant de thèse |
| Frédéric Roupin | Université Paris 13 | Président du jury |
| Samuel Burer | University of Iowa | Rapporteur |
| Antonio Frangioni | University of Pisa | Rapporteur |
| Immanuel Bomze | University of Vienna | Examinateur |
| Francesco Rinaldi | University of Padova | Examinateur |

à Villetaneuse, le 6 novembre 2019




# Contents

# Abstract

A significant number of real-world problems can be modeled as (mixed integer) nonlinear programming problems. There are several solution methods in literature for these problems, which are, however, not always efficient in general, in particular for large scale problems. Decomposition strategies such as Column Generation have been developed in order to substitute the original problem with a sequence of more tractable ones. One of the most known of these techniques is Dantzig-Wolfe Decomposition: it has been developed for linear problems and it consists in solving a sequence of subproblems, called respectively master and pricing programs, which leads to the optimum. This method can be extended to convex non linear problems and a classic example of this, which can be seen also as a generalization of the Frank-Wolfe algorithm, is Simplicial Decomposition (SD).

In this thesis we discuss decomposition algorithms for solving quadratic optimization problems. In particular, we start with quadratic convex problems, both continuous and mixed binary. Then we tackle the more general class of binary quadratically constrained, quadratic problems.

In the first part, we concentrate on SD based-methods for continuous, convex quadratic programming. We introduce new features in the algorithms, for both the master and the pricing problems of the decomposition, and provide results for a wide set of instances, showing that our algorithm is really efficient if compared to the state-of-the-art solver *Cplex*. This first work is accepted for publication in the journal *Computational Optimization and Applications*.

We then extend the SD-based algorithm to mixed binary convex quadratic problems; we embed the continuous algorithm in a branch and bound scheme that makes us able to exploit some properties of our framework. In this context again we obtain results which show that in some sets of instances this algorithm is still more efficient than *Cplex*, even with a very simple branch and bound algorithm. This work is in preparation for submission to a journal.

In the second part of the thesis, we deal with a more general class of problems, that is quadratically constrained, quadratic problems, where the constraints can be quadratic and both the objective function and the constraints can be non convex. For this class of problems we extend the formulation to the matrix space of the products of variables; we study an algorithm based on Dantzig-Wolfe Decomposition that exploits a relaxation on the Boolean Quadric Polytope (BQP), which is strictly contained in the

Completely Positive cone and hence in the cone of positive semidefinite (PSD) matrices. This is a constructive algorithm to solve the BQP relaxation of a binary problem and we obtain promising results for the root node bound for some quadratic problems. We compare our results with those obtained by the Semidefinite relaxation of the ad-hoc solver *BiqCrunch*. We also show that, for linearly constrained quadratic problems, our relaxation can provide the integer optimum, under certain assumptions. We further study block decomposed matrices and provide results on the so-called BQP-completion problem; these results are connected to those of PSD and CPP matrices. We show that, given a BQP matrix with some unspecified elements, it can be *completed* to a full BQP matrix under some assumptions on the positions of the specified elements. This result is related to optimization problems. We propose a BQP-relaxation based on the block structure of the problem. We prove that it provides a lower bound for the previously introduced relaxation, and that in some cases the two formulations are equivalent. We also conjecture that the equivalence result holds if and only if its so-called *specification graph* is chordal. We provide computational results which show the improvement in the performance of the block-based relaxation, with respect to the unstructured relaxation, and which support our conjecture. This work is in preparation for submission to a journal.

**Keywords** : *Quadratic Optimization, Column Generation, Dantzig-Wolfe-Decomposition, Simplicial Decomposition, Correlation polytope, Boolean Quadric Polytope.*

# Résumé

La programmation non linéaire mixte peut modéliser un grand nombre de problèmes réels. Cependant, ces problèmes peuvent contenir de nombreuses variables ou contraintes, il convient donc de proposer des méthodes de décomposition afin de les résoudre efficacement. Parmi ces techniques on peut citer la génération de colonnes et notamment la décomposition de Dantzig-Wolfe. Il s'agit d'une reformulation du problème original, qui permet de générer une séquence de sous-problèmes plus simples, appelés maître et pricing, pour obtenir la valeur optimale. Développée d'abord pour les problèmes linéaires, la décomposition de Dantzig-Wolfe peut être généralisée à des problèmes convexes: dans ce contexte, elle est notamment connue sous le nom de décomposition simpliciale.

Cette thèse présente des algorithmes de décomposition pour des problèmes quadratiques. La première partie de ce manuscrit est dédiée aux problèmes quadratiques convexes, continus et mixtes binaires. Dans la deuxième partie, des algorithmes pour résoudre des problèmes binaires avec contraintes quadratiques sont présentés.

La première partie est consacrée à la résolution de problèmes convexes, quadratiques et continus. Un algorithme basé sur la décomposition simpliciale est proposé: des nouveaux éléments sont ajoutés à la fois au problème maître et au pricing; nous avons testé notre algorithme sur une grande quantité d'instances avec une structure déterminée, et nos résultats montrent que l'algorithme que nous proposons est très efficace par rapport à *Cplex*, un solveur générique pour ces problèmes. Ce premier travail a été soumis à un journal pour publication. Ensuite, nous étendons cet algorithme aux problèmes convexes mixtes binaires. Nous incorporons la méthode pour le cas continu dans un algorithme de *branch and bound* qui nous permet d'exploiter des propriétés de notre formulation. Dans ce contexte aussi, des résultats numériques sont fournis: ils montrent que, dans certains cas, les performances de notre algorithme sont efficaces par rapport à *Cplex*. Ce travail est en préparation pour soumission à un journal.

La deuxième partie de cette thèse est dédiée à l'étude d'algorithmes pour des problèmes quadratiques avec contraintes quadratiques. On se concentre sur les problèmes binaires, dont la relaxation continue peut être non convexe. Nous considérons en premier lieu la formulation étendue avec une matrice qui représente les produits des variables. Nous proposons ensuite un algorithme basé sur la décomposition de Dantzig-Wolfe pour obtenir une relaxation dans le *Boolean Quadric Polytope* (BQP). Ce polytope est connu aussi comme *Correlation polytope* et il est strictement contenu dans le cône des matrices *complètement positives* et des matrices *semidéfinies positives*. Notre algorithme permet de

résoudre cette relaxation, les bornes obtenues sont plus fortes que les bornes SDP et, dans certains cas, les temps de calcul sont comparables ou meilleurs que ceux de *BiqCrunch*, un solveur ad-hoc. On montre aussi que la relaxation BQP est une reformulation du problème binaire original, en exploitant un résultat sur les matrices complètement positives, pour les problèmes à contraintes linéaires en égalité.

Ensuite, nous considérons des problèmes où les matrices sont décomposables par blocs. Une relaxation basée sur les blocs est proposée et nous prouvons que cette relaxation est valide pour la relaxation BQP. De plus, prouver l'équivalence entre les deux relaxations est un problème de *complétion BQP*. La relaxation décomposée par blocs est *BQP-complétable* dans certains cas, mais n'est pas possible dans d'autres cas. À partir de résultats expérimentaux, nous conjecturons que la classe de problèmes qui sont BQP-complétables est la classe de problèmes dont le graphe *de specification* des matrices est chordal. Des résultats computationnels montrent que la formulation par blocs, dans certaines instances où elle est équivalente à la relaxation originale, peut être beaucoup plus efficace. Ce travail est en préparation pour soumission à un journal.

**Mots-clés** : *Optimisation quadratique, Géneration de colonnes, Décomposition de Dantzig-Wolfe, Décomposition simpliciale, Boolean Quadric polytope, Correlation polytope.*

# Acknowledgments

I would like to thank Prof. Antonio Frangioni and Prof. Samuel Burer for having accepted to be referees of my thesis.

Many thanks to Lucas and Emiliano, for having accepted to supervise me for my PhD after having helped me during my Master's Thesis. In particular I thank Lucas for his support, his scientific advice and help. Emiliano devoted an incredible amount of time to me and was always available: I thank him for the long discussions, his commitment, suggestions, and patience.

I would like to express my gratitude to Francesco for his patience, from the supervision of my Master's thesis to now: he never stopped helping me. And it is mostly thanks to him I decided to start a PhD.

Thanks to Prof. Manuel Bomze for having offered me the possibility to spend two intense months in Vienna and to work with him, for his scientific advice and suggestions, and his kindness.

Thanks to the other members of the AOC team, in particular to Frédéric Roupin, who accepted to be president of the jury, and to Roberto, available to listen and help whenever I needed. Thanks to all the members of the LIPN, who make this lab a pleasant environment.

Thanks to my office mates and to the other PhD colleagues: Ugo, Sarah, Mehdi, Davide, Jawher, Juan José, Massinissa, and all others, with whom I shared this experience and enjoyed several hilarious moments.

A special thank goes to Emiliano jr, for all the moments we shared in these years, the endless discussions about everything, for his advice and where necessary psychological support.

Infinite thanks to Stefania, exceptionally generous friend, who helped me on everything, included picking up the mail for me and offering me to stay at her place (and having my name engraved on her mailbox for that).

Many thanks to all my friends for their support, in particular to Marcos, Desi, Giulia, to Michelone and all the Sicilian guys, to many other people who I met in these years.

Grazie in modo speciale ai miei genitori e a mio fratello, che mi hanno sostenuto sempre.

# Abbreviations and Notations

| | |
|---|---|
| **B&B** | **B**ranch-and-**B**ound algorithm |
| **IP** | **I**nteger **P**rogramming |
| **LP** | **L**inear **P**rogramming |
| **MILP** | **M**ixed **I**nteger **L**inear **P**rogramming |
| **QP** | **Q**uadratic **P**rogramming |
| **MIQP** | **M**ixed **I**nteger **Q**uadratic **P**rogramming |
| **QCQP** | **Q**uadratically **C**onstrained **Q**uadratic **P**rogramming |
| **CP** | **C**onic **P**rogramming |
| **SDP** | **S**emi**D**efinite **P**rogramming |
| **COP** | **C**opositive **P**rogramming |
| **BQP** | **B**oolean **Q**uadric **P**olytope |
| **PSD** | **P**ositive **S**emi**D**efinite matrices |
| **CPP** | **C**om**P**letely **P**ositive matrices |

**Notations:** We use the following standard notation:

$\mathbb{N}$: set of natural numbers;

$\mathbb{R}$: set of real numbers;

$\mathbb{Z}$: set of integer numbers;

$\mathcal{S}^n$: set of symmetric matrices in $\mathbb{R}^n$;

$\mathcal{S}^n_+$: set of positive semidefinite matrices in $\mathbb{R}^n$;

$\mathcal{S}^n_{++}$: set of positive definite matrices in $\mathbb{R}^n$;

$\mathcal{N}^n$: set of nonnegative matrices in $\mathbb{R}^n$;

$\mathcal{DNN}^n$: set of doubly nonnegative matrices in $\mathbb{R}^n$;

$\mathcal{C}^n$: set of copositive matrices in $\mathbb{R}^n$;

$\mathcal{C}^{*n}$: set of completely positive matrices in $\mathbb{R}^n$.

Given a vector $v \in \mathbb{R}^n$ and a matrix $A \in \mathbb{R}^{m\times n}$:

$v_i$: i-th element of $v$;

$A_{ij}$: element of $A$ in row $i$ and column $j$;

$\mathrm{Tr}(\cdot)$: trace operator;

$v^\top \in \mathbb{R}^n$, $A^\top \in \mathbb{R}^{n\times m}$: transpose of $v$ and of $A$, respectively;

$e$: vector of all ones;

$E$: matrix of all ones;

$E_{ij}$: matrix with 1 on $(i,j)$-th and $(j,i)$-th element, 0 elsewhere;

$\mathbb{I}_n$: the identity matrix of size $n$;

$0_n$: null vector of dimension n;

$\mathbb{O}_n$: null square matrix of dimension $n \times n$;

$v \geq 0 \iff \forall i = 1,\dots,n,\ v_i \geq 0$;

$A \geq 0$ $(A \leq 0) \iff \forall i = 1,\dots,m,\ \forall j = 1,\dots,n,\ A_{i,j} \geq 0$ $(A_{i,j} \leq 0)$;

$A \succeq 0$ $(A \succ 0) \iff$ $A$ positive semidefinite (positive definite);

$\mathbb{R}_+ := \{x \in \mathbb{R} |\, x \geq 0\}$, $\mathbb{R}^n_+ := \{v \in \mathbb{R}^n \,|\, v \geq 0\}$;

Given a function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ and a point $x \in X$:

$\nabla f(x) \in \mathbb{R}^n$: gradient of $f$ in $x$;

Given two matrices $A, B \in \mathbb{R}^{m\times n}$:

$\langle A, B\rangle := \sum_{i=1}^m \sum_{j=1}^n A_{ij} \cdot B_{ij} = \mathrm{Tr}(A^\top B)$ (*Hilbert product*);

Given a set $S \in \mathbb{R}^n$:

$int(S)$: the interior of S;

$cl(S)$: the closure of S;

$ri(S)$: the relative interior of S;

$\partial S$: the boundary of S;

# Introduction

In diverse fields, like Business, Industry, Finance, Logistics among many others, but also in everyday life, making decision is a fundamental activity. In many cases decisions are made informally, mostly based on experience. However, a formal approach is often preferable, because some problems are complex and making the best decision is not always straightforward. In addition, in some cases the best choice is counter-intuitive; however, it could make us save a lot of money, time, or other resources.

Operations research is the discipline that studies methods which help to make decisions. A formal approach is done with the introduction of a mathematical model to describe the problem. The need for a correct and efficient way to find the best solution in several domains gave rise to the discipline of Mathematical Optimization, which has the objective to develop methods to solve optimization problems. The origin and the main goal of this discipline is to help solving decision problems and a relevant part of it is related to computational aspects. However, theoretical research is fundamental, and makes this one of the clearest examples of fruitful applications of Mathematics, which comprises both theory and applications.

The attention to efficiency and algorithms is another fundamental element. Different formulations for the same problem can be solved in very different computational times. Moreover, for the same formulation different algorithms can be used and the performances can vary a lot. Some problems (those which belong to the so-called Polynomial class) can be solved with algorithms which have a complexity which is bounded by a polynomial on the size of the data. For other problems, typically the so-called NP-hard problems, no polynomial algorithm is known and hence they are generally much more difficult to solve.

An optimization problem is formally defined as the minimization (or maximization) of an objective, expressed as a function of several variables, and some constraints that the variables must satisfy. The most common optimization problems are *linear problems* and have the following form:

$$\min\ c^\top x \tag{1a}$$

$$\text{s. t. } Ax = b, \tag{1b}$$

$$x \geq 0. \tag{1c}$$

Here $n, m \in \mathbb{N}$, $x \in \mathbb{R}^n$ are the variables of the problem, $A$ is a matrix in $\mathbb{R}^{m \times n}$ which defines the constraints, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ is the vector of *costs*, which defines the objective function. The solution of this problem is given by a vector $x^* \in \mathbb{R}^n$, the optimal

vector, which has objective function value $v^* = c^\top x^*$, the minimal value among all the feasible vectors. Linear problems can model several different problems and efficient algorithms exist to solve them. The most used one is the *simplex* algorithm: it is the most efficient in applications, but it is known not to be polynomial. However, polynomial algorithms have been introduced: firstly the so-called *ellipsoid* algorithm, and more recently *interior point methods*.

In some cases, continuous variables are not sufficient to describe the model and we need to employ integer, or binary variables. This is the case, for instance, of the so-called *knapsack* problem: we are given $n$ objects, each with a value $v_i$ and a weight $w_i$ $(i = 1, \ldots, n)$; we want to pick the subset of these objects with maximal value, whose total weight does not exceed a capacity $W$. The most natural choice of variables is a binary variable for every object, which is 1 if the object is picked, 0 otherwise.

An optimization problem can have more general forms than (1): for instance, the objective function or the constraints can be nonlinear. If they are quadratic functions, the problem is denoted as a quadratic problem. Depending on more specific features of the model, several efficient ad-hoc algorithms have been developed for this class of problems as well. This will be treated in more detail in Chapter 2. Moreover, optimization problems can have a large number of variables or of constraints, and have specific structures. Some solution methods specifically tailored for problems with such particular characteristics has been developed. For instance, typically for large-scale problems, decomposition methods are used, which find the optimum of a problem by solving sequences of subproblems with simplified structure. Two important classes of decomposition methods are the following: those based on an *outer* approximation of the feasible region, (for instance cutting plane methods) and those which compute an *inner* approximation of it. Several decomposition methods exist, but we concentrate on a specific inner approximation decomposition: the so-called *column generation* technique, which will be better described in Chapter 2. In this thesis we will present some algorithms to solve quadratic problems which are based on a specific column generation method. We will treat different types of problems and we will present both theoretical investigations and extensive computational results. In the next section we will describe the structure in more detail.

## Structure of the thesis

This thesis is devoted to the study and the development of algorithms based on column generation to solve quadratic programs. Mainly, we deal with general linearly constrained convex quadratic programs and quadratically constrained, quadratic programs. The application of specific column generation techniques for these two classes of problems are treated in the two main parts of the thesis. The first chapters are instead dedicated to the introduction of the background theory, from basic concepts to more specific ones: they are intended to summarize and present in a unified notation the notions on which this research has been built.

We will start with an introduction in Chapters 1 and 2. In the first one we will recall some standard mathematical concepts which are the background for this study. They

include convex sets and linear algebra, polyhedral and graph theory; then, we further introduce the definitions of some matrices, cones and polytopes which are used later.

In Chapter 2 we will introduce some aspects of Quadratic Programming, along with the most important first order solution methods and in particular column generation methods. We also introduce basics of conic programming, with particular interest on Semidefinite and Copositive Optimization.

This chapter concludes the introductory overview on the basic tools that are treated and used in this thesis. The rest of the thesis is divided into two main parts: both of them deal with column generations based algorithms applied to quadratic programming, but with a focus on different types of problems.

In the first part, which consists of Chapters 3 and 4, we will treat algorithms for linearly constrained convex quadratic problems; in the second part (Chapters 5 and 6) we will consider nonconvex, quadratically constrained ones. The other main difference is that the first part is characterized by a specific column generation algorithm, namely *Simplicial Decomposition*, and its connections with continuous optimization methods; the second part is instead based on a more combinatoric approach, linked also to conic programming.

In Chapter 3 we present a Simplicial decomposition based algorithm for a class of continuous, convex quadratic problems. We develop an ad-hoc algorithm, specifically for the master problem; we integrate one other method for the master and three algorithms for the pricing. We provide extensive computational results which show that under some assumptions our framework is efficient with respect to a state-of-the-art solver (*Cplex*).

In Chapter 4 we extend the framework to the case of mixed binary convex quadratic problems. We embed SD into a Branch and bound scheme, which allows us to take efficiently advantage of some properties of our framework.

The second part starts with Chapter 5, where we describe the algorithm that we propose for solving quadratically constrained, binary quadratic problems, which consists of a relaxation based on the Dantzig-Wolfe reformulation on the extended space. It turns out to be a relaxation in the BQP polytope, strictly contained in the so-called *Completely Positive* cone. We provide computational results, with a particular attention to the bounds that we obtain in some cases.

Then, in Chapter 6 we consider block-decomposable problems and we show how to take advantage of the structure in order to reduce the size of our subproblems. We show that, in order to prove results on the equivalence of our formulations, we have to deal with a *completion* problem. We present an interesting theoretical property of the completion problem for the so-called *Boolean Quadric Polytope*. We also show the effectiveness of our algorithm with results on the bound and on the computing time needed to obtain them.

Finally, in the last Chapter 7, we draw some conclusions and some perspectives of future research directions.

# Chapter 1

# Preliminaries and basic definitions

Chapters 1 and 2 are devoted to the introduction of the background theory which is strongly used in this thesis. In this chapter we recall some necessary basic concepts, such as definitions of convexity, Linear Algebra and Polyhedral theory. Then, we will recall some basic results of some important classes of matrices which will be object of study in the following chapters. Successively, we will describe elements of graph theory. We will conclude the chapter with a description of some important polytopes and cones of matrices which we will deal with in the rest of the thesis and we also concentrate on a specific problem: the so-called matrix completion problem, which involves graphs and cones of matrices.

## 1.1 Linear Algebra and Geometry

We start with recalling some basic concepts of convex sets and Linear Algebra, then some classic definitions and key results in polyhedral and conic theory. The results are standard and for this section we mainly consider the references [23] and [39].

**Definition 1.1.** *A set $C$ is* convex *if for any couple of points $x, y \in C$, any convex combination $z$ of $x$ and $y$ is contained in $C$: $z = \alpha x + (1 - \alpha y)$, $z \in C \, \forall \, 0 \leq \alpha \leq 1$.*

**Definition 1.2.** *Given a function $f : \mathbb{R}^n \mapsto R$:*

*$f$ is said to be* convex *if $\forall x, y \in \mathbb{R}^n$, $\forall \theta \in [0, 1]$, $f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta) f(y)$.*

*For every $t \in \mathbb{R}$, its sublevel sets are: $\{x \in \mathbb{R}^n \mid f(x) \leq t\}$. If $f$ is convex, all its sublevel sets are convex.*

*$f$ is* strictly convex *if $\forall x, y \in \mathbb{R}^n$, $\forall 0 < \theta < 1$, $f(\theta x + (1 - \theta)y) < \theta f(x) + (1 - \theta) f(y)$.*

*$f$ is* (strictly) concave *if the function $-f$, defined as the opposite of $f$, is (strictly) convex.*

**Definition 1.3.** *A* linear combination *of vectors $v_1, \ldots, v_n$ is a vector $v = \alpha_1 v_1 + \cdots + \alpha_n v_n$, where $\alpha_i \in \mathbb{R} \, \forall i = 1, \ldots, n$.*

*Given some vectors $v_1, \ldots, v_n$, they are* linearly independent *if the only coefficients $\alpha_i$, $i = 1, \ldots, n$ such that $\alpha_1 v_1 + \cdots + \alpha_n v_n = 0$ are $\alpha_i = 0 \, \forall i = 1, \ldots, n$.*

*A set $V$ is a* (real) linear vector space *if it is closed under linear combinations, i.e. every linear combination of elements of $V$ belongs to $V$.*

*A set of vectors $v_1, \ldots, v_n$ is a* basis *of a vector space $V$ if they are linearly independent and any vector $v \in V$ is a linear combination of them. (As a consequence, this linear combination is unique).*

*The* dimension *of a linear vector space is the maximum number of independent vectors in it, so it is the number of vectors in any basis.*

*A subset of a vector space which is a vector space is called* vector subspace.

**Definition 1.4.** *An* affine combination *of points $x_1, \ldots x_n$ is a point $x = \alpha_1 x_1 + \cdots + \alpha_n x_n$, where $\alpha_i \in \mathbb{R} \, \forall i = 1, \ldots, n$ and $\sum_{i=1}^{n} \alpha_i = 1$.*

*Given some points $x_1, \ldots x_n$, they are* affinely independent *if the only coefficients $\alpha_i$, $i = 1, \ldots, n$ such that $\alpha_1 x_1 + \cdots + \alpha_n x_n = 0$ and $\alpha_1 + \cdots + \alpha_n = 0$ are $\alpha_i = 0 \, \forall i = 1, \ldots, n$. Equivalently, the vectors $x_2 - x_1, \ldots, x_n - x_1$ are linearly independent, irrespective of the choice of $x_1$.*

*A set $S$ is an* affine space *if it is closed under affine combinations.*

*The* dimension *of an affine space $S$ is the maximum number of affinely independent points in $S$ minus 1.*

*Given a set of points $x_1, \ldots x_n$, their* affine hull *is the set of all their affine combinations. It is an affine space of dimension at most $n - 1$.*

*Given a set of affinely independent points $x_1, \ldots x_n$, and the point $x = \alpha_1 x_1 + \cdots + \alpha_n x_n$, the coefficients $\alpha_i$, $i = 1, \ldots, n$ are the* barycentric coordinates *of $x$.*

**Definition 1.5.** *Given a set $S \subseteq \mathbb{R}^n$, its* dimension *is the maximum number of affinely independent points in $S$ minus 1. $S$ is* fully dimensional *if its dimension is $n$.*

**Definition 1.6.** *A* convex combination *of points $x_1, \ldots x_n$ is a point $x = \alpha_1 x_1 + \cdots + \alpha_n x_n$, where $\alpha_i \in \mathbb{R}_+ \, \forall i = 1, \ldots, n$ and $\sum_{i=1}^{n} \alpha_i = 1$.*

*Given a set of points $x_1, \ldots x_n$, their* convex hull *is the set of all their convex combinations. If the points $x_1, \ldots x_n$ are affinely independent, their convex hull is called* simplex.

**Definition 1.7.** *A* conic combination *of vectors $v_1, \ldots v_n$ is a vector $v = \alpha_1 v_1 + \cdots + \alpha_n v_n$, where $\alpha_i \in \mathbb{R}_+ \, \forall i = 1, \ldots, n$.*

*A set $C$ is a* cone *if $\forall x \in C$, $\alpha x \in C \, \forall \alpha \in \mathbb{R}_+$. For every cone $C$, the origin $0 \in C$.*

*A set $C$ is a* convex cone *if it is closed under conic combinations, i.e. every conic combination of elements of $C$ belongs to $C$.*

*A cone $C$ is* pointed *if $C \cap -C = \{0\}$, where for every set $C$, $-C = \{x \mid -x \in C\}$.*

*Given a set of vectors $v_1, \dots v_n$, their* conic hull *is the set of all their conic combinations. It is a cone and its dimension is at most $n$.*

*Remark* 1.1. Trivially, linear spaces and affine, conic, convex hulls are convex sets.

**Definition 1.8.** *A set $P \in \mathbb{R}^n$ is a* polyhedron *if*

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}.$$

*for a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$, for $m \in \mathbb{N}$. If $b = 0_m$, $P$ is a cone and it is called* polyhedral cone.

**Definition 1.9.** *A cone $C \in \mathbb{R}^n$ is* finitely generated *if it is the conic hull of a finite set of vectors $r_1, \dots, r_r \in \mathbb{R}^n$.*

**Definition 1.10.** *Let $P = \{x \in \mathbb{R}^n \mid Ax \leq b\} \in \mathbb{R}^n$ be a polyhedron.*

*The* recession cone *of $P$ is the set $\{r \in \mathbb{R}^n \mid x + \alpha r \in P \, \forall \alpha \in \mathbb{R}_+\} = \{r \mid Ar \leq 0\}$.*

*The* lineality space *of $P$ is the set $\{r \in \mathbb{R}^n \mid x + \alpha r \in P \, \forall \alpha \in \mathbb{R}\} = \{r \mid Ar = 0\}$.*

*$P$ is* pointed *if its lineality space is $\{0\}$.*

**Definition 1.11.** *Let $P \in \mathbb{R}^n$ be a polyhedron. An inequality $a^\top x \leq b$ is* valid *for $P$ if it holds true for every $x \in P$.*

*A* face *of $P$ is a set $S \in \mathbb{R}^n$ such that $S = P \cup \{x \mid a^\top x = b\}$, where $a^\top x \leq b$ is valid for $P$.*

*A* facet *of $P$ is any maximal size face of $P$.*

*A* vertex *of $P$ is a face of size 0.*

*An* edge *of $P$ is a face of size 1.*

**Definition 1.12.** *Given two sets $R, S \subset \mathbb{R}^n$, their* Minkowski sum *is the set*

$$R + S = \{x \in \mathbb{R}^n \mid \exists r \in R, \, s \in S \, x = r + s\}.$$

We report the Theorems of Minkowski-Weyl, used in later discussion. For the proof and further discussion, see for instance [39].

**Theorem 1.1** (Minkowski-Weyl Theorems)**.** *The theorem is expressed in two statements, respectively for cones and for polyhedra:*

*a cone is polyhedral if and only if it is finitely generated;*

*a polyhedron* $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ *can be equivalently written as the sum of the convex hull of a finite set of points* $x_p$ *and the conic hull of a finite set of vectors* $v_r$*:*

$$P = \sum_{p=1}^{n_p} \lambda_p x_p + \sum_{r=1}^{n_r} \mu_r v_r$$

*with* $\sum_{i=1}^{n_p} \lambda_p = 1$ *and* $\lambda_p \geq 0$, $\mu_r \geq 0 \, \forall p = 1, \ldots, n_p$, $r = 1, \ldots, n_r$.

**Definition 1.13.** *A* polytope *is a bounded polyhedron.*

*Remark* 1.2. A simplex with $n+1$ vertices is hence a particular polytope with dimension $n$ and it is often indicated with $\Delta_n$. In its barycentric coordinates, the constraints which define a simplex are hence the following:

$$\sum_{i=1}^{n} \lambda_i = 1, \qquad \lambda_i \geq 0 \quad \forall i = 1, \ldots, n. \tag{1.1}$$

*Remark* 1.3. Not all the cones are convex or polyhedral. For instance, many cones of matrices are not polyhedral, as we will see later.

**Definition 1.14.** *Given a set* $X$*, a function* $d : X \times X \to \mathbb{R}$ *is a* metric *if:* $d(x, y) = 0 \iff x = y$*,* $d(x, y) = d(y, x)$*, and* $d(x, y) + d(y, z) \leq d(x, z)$*. We consider* $\mathbb{R}^n$ *as a* metric space*, i.e. the set* $\mathbb{R}^n$ *equipped with a metric, and we consider the usual Euclidean distance. The distance between two points* $x, y \in \mathbb{R}^n$ *is indicated by* $d_n(x, y) \in \mathbb{R}$*. A* Ball*,* centered *in* $x \in \mathbb{R}^n$ of radius $r > 0$*, is a subset* $B_n(x, r) \subseteq \mathbb{R}^n$ *such that* $\forall y \in B_n(x, r)$*,* $d_n(x, y) < r$*. Given a convex set* $S \subset \mathbb{R}^n$*:*

*The* interior *of* $S$ *is* $int(S) = \{x \in S \mid \exists \varepsilon_x > 0, B_n(x, \varepsilon_x) \subset S\}$.

*The* closure *of* $S$ *is* $cl(S) = \{x \in \mathbb{R}^n \mid \forall \varepsilon > 0, \, B_n(x, \varepsilon) \cap S \neq \emptyset\}$*: it is the smallest closed set which contains* $S$*.*

*The* Relative interior *of* $S$ *is* $ri(S) = \{x \in S \mid \exists \varepsilon > 0, \, B_d(x, \varepsilon) \subset Aff(S) \mid B_d(x, \varepsilon) \subset S\}$*, where* Aff(S) *is the affine hull of* $S$ *and* $d$ *is the dimension of* $S$*.*

*The* boundary *of* $S$ *is* $\partial_{bd} S = cl(S) \setminus int(S)$

*The* Relative boundary *of* $S$ *is* $\partial_{rbd} S = cl(S) \setminus ri(S)$.

*Remark* 1.4. We will simply use $\partial S$ to indicate the relative boundary of $S$.

Another important tool is the Caratheodory theorem, which we here recall in the version for polytopes, which is what we will use later:

**Theorem 1.2** (Caratheodory's Theorem)**.** *Let* $P$ *be a polytope in* $\mathbb{R}^n$ *of dimension* $d$*. Any point* $x \in P$ *is a convex combination of at most* $d+1$ *affinely independent vertices of* $P$*.*

**Definition 1.15.** *A* (real) Hilbert space *is a real linear vector space with an inner product* $\langle \cdot, \cdot \rangle : H \times H \mapsto \mathbb{R}$.

The Hilbert spaces we will deal with are cones: $\mathbb{R}^n$ with the product $\langle x, y \rangle = x^\top y$, and spaces of symmetric matrices with the Hilbert product $\langle X, Y \rangle = \mathrm{Tr}(X^\top Y)$.

**Definition 1.16.** *For any Hilbert space* $H$ *and a cone* $C \subseteq H$*:*

*The* dual cone *of* $C$ *is the set* $C^* \subseteq H$ *given by:*

$$C^* := \{y \in H \,|\, \langle x, y \rangle \geq 0 \, \forall x \in C\}.$$

*The* polar cone *of* $C$ *is the set* $C^\perp \subseteq H$ *given by:*

$$C^\perp = \{y \in H \,|\, \langle x, y \rangle \leq 0 \, \forall x \in C\}.$$

*A cone* $C \in H$ *is said* self-dual *if* $C^* = C$.

In the following proposition (see [5]) we collect a few important properties of duality of cones.

**Proposition 1.1.** *Let* $C_1$, $C_2$ *be closed convex cones in* $\mathbb{R}^n$.

- *If* $C_1 \subseteq C_2$, *then* $C_2^* \subseteq C_1^*$;
- $C_1^{**} = C_1$;
- $(C_1 \cap C_2)^* = cl(C_1^* + C_2^*)$

## 1.2 Matrices

In this section we recall some classical sets of matrices and their properties. We always consider real valued matrices. The books [6] and [95] are a good reference for these concepts.

Let $A \in \mathbb{R}^{n \times n}$ be a square matrix. We indicate with $A_{ij}$ the element of $A$ in row $i$ and column $j$ and with $A^\top$ the transpose of $A$. $A$ is *Symmetric* if $A = A^\top$.

**Definition 1.17.** *Let* $A \in \mathbb{R}^{n \times n}$ *be a square matrix.* $A$ submatrix *$S$ of $A$ is a matrix obtained by removing some rows and columns from $A$. $S$ is a* principal submatrix *if the indices of the removed rows and columns are the same.* $A$ minor *of $A$ is the determinant of a square submatrix of $A$.* $A$ principal minor *of $A$ is the determinant of a principal submatrix.*

In the rest of this section, we always consider symmetric matrices. We indicate with $\mathcal{S}^n$ the set of symmetric matrices in $\mathbb{R}^n$. Here we report the Spectral Theorem for real symmetric matrices (see [95], Theorem 4.1.5).

**Proposition 1.2** (Spectral Theorem)**.** *$A \in \mathcal{S}^n$ if and only if $\exists P, D \in \mathbb{R}^n$ s.t. $PP^\top = \mathbb{I}$, $D$ is diagonal, and $A = PDP^\top$. The diagonal of $D$ contains the eigenvalues of $A$.*

*Remark* 1.5. As a consequence, symmetric matrices have real eigenvalues.

**Definition 1.18.** *Let $A \in \mathcal{S}^n$. We say that $A$ is:*

Positive semidefinite *(PSD) if $\forall x \in \mathbb{R}^n, x^\top Ax \geq 0$;*

Positive definite *(PD) if $\forall x \in \mathbb{R}^n s.t.\ x \neq 0, x^\top Ax > 0$.*

*We indicate, respectively, with $\mathcal{S}^n_+$ and $\mathcal{S}^n_{++}$ the sets of positive semidefinite and positive definite matrices in $\mathbb{R}^n$.*

We recall a few of the well known properties of PSD matrices which will be useful later. Their proofs are, for instance, in [95].

**Proposition 1.3.** *Let $A \in \mathcal{S}^n$. Then the following are equivalent:*

- *$A$ is PSD;*
- *All the eigenvalues of $A$ are nonnegative;*
- *All the principal minors of $A$ are nonnegative;*
- *$\exists B \in \mathbb{R}^{n \times k}$ such that $A = BB^\top$.*

*Remark* 1.6. As a consequence of the last statement, $A$ can be written as the sum of $k$ rank-1 matrices: $A = \sum_{i=1}^k b_i b_i^\top$ where $b_i \in \mathbb{R}^n$ are the columns of $B$.

Here we define other useful matrices.

**Definition 1.19.** *Let $A \in \mathcal{S}^n$. We say that $A$ is:*

Non Negative *($A \geq 0$) if $\forall i, j = 1, \dots n, A_{ij} \geq 0$;*

Totally non negative *(TN) if every minor of $A$ is non negative;*

Doubly non negative *(DNN) if $A$ is both PSD and non negative;*

*Remark* 1.7. If $A$ is TN, then it is both non negative and positive semidefinite, so it is DNN.

We indicate, respectively, with $\mathcal{N}^n$ and $\mathcal{DNN}^n$ the sets of non negative and doubly non negative matrices in $\mathbb{R}^n$.

Finally, we introduce the copositive and completely positive matrices, which we will treat more in detail in Chapter 2 and will be useful for the second part of the thesis.

**Definition 1.20.** *Let $A \in \mathcal{S}^n$. We say that $A$ is:*

Copositive *(COP) if $\forall x \in \mathbb{R}^n_+, x^\top Ax \geq 0$;*

Completely positive *(CPP) if* $\exists k, B \in \mathbb{R}_+^{n\times k}$ *such that* $A = BB^\top$.

*Remark* 1.8. We notice that the definitions of COP and CPP matrices come as generalizations of the definition of PSD matrices. Wider generalizations are possible, and are the so-called set-semidefinite matrices.

*Remark* 1.9. We notice that, similarly to the case of PSD matrices, the definition of a CPP matrix can equivalently be stated as

$$A \text{ is } CPP \iff A = \sum_{i=1}^{k} b_i b_i^\top,$$

where $b_i \in \mathbb{R}_+^n$ are the columns of $B$. Hence, $A$ is in the conic hull of nonnegative rank-1 matrices.

We indicate, respectively, with $\mathcal{C}^n$ and $\mathcal{C}^{*n}$ the sets of copositive and completely positive matrices in $\mathbb{R}^n$.

*Remark* 1.10. From the definition of dual cones and from Remark 1.6, it is easy to notice that $\mathcal{S}_+^n$ and $\mathcal{N}^n$ (and also $\mathbb{R}^n$) are cones and they are are self-dual, and the dual of $\mathcal{C}^n$ is actually $\mathcal{C}^{*n}$. If we consider also Proposition 1.1, we can observe that the dual cone of $\mathcal{DNN}^n$ is $\mathcal{DNN}^{n*} = \mathcal{S}^n + \mathcal{N}^n$.

*Remark* 1.11. It is easy to notice that the sets $\mathcal{S}^n$, $\mathcal{S}_+^n$, $\mathcal{N}^n$, $\mathcal{DNN}^n$ are closed, convex, full-dimensional pointed cones. $\mathcal{S}_{++}^n$ is a cone and is the interior of $\mathcal{S}_+^n$. It has been observed in [91] and in [52] that also the sets $\mathcal{C}^n$ and $\mathcal{C}^{*n}$ are closed, convex, full-dimensional pointed cones. All the mentioned cones share the same vertex $\mathbb{O}_n$.

From the definitions, along with Proposition 1.1 and the Remarks, we can establish the following relations:

$$\mathcal{C}^{*n} \subseteq \mathcal{DNN}^n \subseteq \mathcal{S}_+^n \subseteq \mathcal{S}_+^n + \mathcal{N}^n \subseteq \mathcal{C}^n. \tag{1.2}$$

*Remark* 1.12. It has been shown by Maxfield and Minc in [107] that the first and last inclusions hold as equalities for $n \leq 4$, while the inclusions are strict for $n \geq 5$. They provide an example of a matrix which is DNN but not CPP, and in [91] there is an example of a matrix which is COP but not nonnegative.

## 1.3 Graphs

Now we recall classical and specific notions of graph theory that are useful for later discussion. For this section we refer principally to [6].

**Definition 1.21.** *A* graph *G(V,E) is the data of two finite sets: V is the set of vertices and E is the set of edges. The vertices are represented by dots and the edges are lines connecting vertices. An edge connecting nodes i and j is represented by a couple* $\{i, j\}$.

**Definition 1.22.** *A* directed graph *D(N,A) is the data of two finite sets: N is the set of nodes and A is the set of arcs. The vertices are represented by dots and the arcs are directed arrows connecting nodes. Every arc has an origin node i and a destination node j and is represented by the ordered couple* $(i, j)$.

In this part we focus on graphs. We list some standard definitions.

**Definition 1.23.** *Two vertices* $u, v \in V$ *of a graph G(V,E) are* adjacent *if there exists an edge* $\{u, v\} \in E$. *In this case, u and v are* neighbors.

*G is* complete *if every vertex is adjacent to every other.*

*If G is not complete, the* completion *of G is a complete graph with vertex set* $V$.

*A graph* $H(V_H, E_H)$ *is a* subgraph *of G if* $V_H \subseteq V$ *and* $E_H \subseteq E$.

*A subgraph* $H(V_H, E_H)$ *of G is* induced *by* $W \subseteq V$ *if* $V_H = W$ *and* $E_H \subseteq E$ *is the set of all edges of E which have endpoints in* $W$.

*A* clique $K$ *is a subset of* $V$ *which induces a complete subgraph of G. It is* maximal *if it is not contained in any other clique.*

*A* path *is a sequence of edges* $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\} \subseteq E$ *where* $v_2, \ldots, v_n$ *are distinct. The* length *of a path is its number of edges.*

*A* cycle *is a path where the first and the last vertices are the same.*

*A* chord *of a cycle C is an edge connecting two nonconsecutive vertices of C.*

*G is* connected *if, for every* $u, v \in V$ *there exists a path connecting them. Otherwise it is* disconnected *and it has at least two distinct connected subgraphs, called* connected components *of G.*

*A* cut-vertex $v$ *is a vertex of G such that the graph induced by* $V \setminus \{v\}$ *has more connected components than G.*

*A* block *is a connected graph with no cut-vertices.*

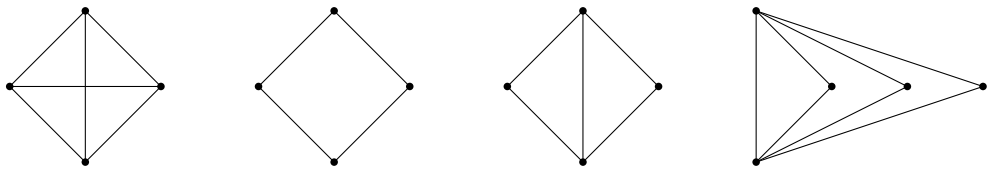
*A* block of G *is a subgraph of G which is a block and which is not contained in any other subgraphs of G which are blocks.*

**Definition 1.24.** *We use the following notation for some special families of graphs.*

$K_n$*: the complete graphs of n vertices.*

$C_n$*: the cycles of n vertices.*

$T_n$*: a graph with* $n-2$ *triangles. That is, a graph with n vertices, where two vertices* $u, v$ *are adjacent to all other vertices, and every other vertex is adjacent only to u and v.*



(a) $K_4$ (b) $C_4$ (c) $T4$ (d) $T_5$

Figure 1.1 – Examples of graphs

In Figure (1.1), there are examples of these three classes of graphs.
Here we describe the two main classes of graphs which we are interested in.

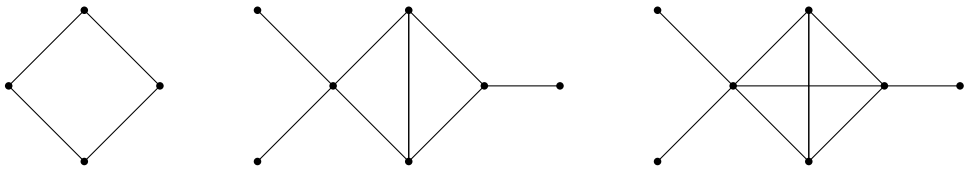
**Definition 1.25.** *Let G(V,E) be a graph.*

*G is* chordal *if for every cycle C of length at least 4, there is an edge in G which is a chord for C.*

*G is* block-clique *if it is connected and each block of G is a complete graph.*

*Remark* 1.13. It follows from the definition that a block-clique graph is chordal. In fact, the block-clique graphs can be defined in two other ways: as the chordal graphs in which any two maximal cliques intersect in at most one vertex, or the chordal graphs which do not contain $T_2$ as a subgraph.

In Figure 1.2 some examples are provided.



(a) Non chordal graph (b) Chordal non block-clique graph (c) Block-clique graph

Figure 1.2 – Chordal and block clique graphs

One of the reasons of the importance of chordal graphs is motivated by the following property. For more details, see, for instance, [6] or [87].

**Definition 1.26.** *Given a graph $G(V, E)$, an ordering $<$ on the nodes set $V$ is a* perfect elimination ordering *if for every $v \in V$, the set of vertices $w \in V$ which are adjacent to $v$, and such that $v < w$, is a clique.*

The following result is presented in [123]:

**Proposition 1.4.** *A graph has a perfect elimination ordering if and only if it is chordal.*

We conclude this section with one last definition:

**Definition 1.27.** *Given a graph $G(V, E)$ with $n$ vertices, its* adjacency matrix *is a matrix $A \in \mathbb{R}^{n\times n}$ where $A_{ij} = A_{ji} = 1$ if $\{i, j\} \in E$ and 0 otherwise.*

## 1.4 Related classes of cones and polytopes

We here define a polytope which has interesting relations to quadratic programming. We follow the presentation given by Ziegler in [145]. This subject has been described in detail by Deza and Laurent in [51].

**Definition 1.28.** *The* Correlation polytope *in dimension $n$ is the convex hull of the rank-1 $n \times n$ 0-1 matrices :*

$$COR^n := conv\{X \in \mathbb{R}^{n \times n} \mid X = xx^\top,\ x \in \{0,1\}^n\}.$$

The same polytope is studied by Padberg in [117], under the name of *Boolean Quadric polytope*, but historically it has been firstly introduced by Pitowsky in [119] under the name of correlation polytope. The choice of the name is justified by a probabilistic approach: to each point in the correlation polytope, it is possible to assign a positive probability in a probability space. Moreover, suppose that every $x_i$ are propositions, $i = 1, \ldots, n$, which can be true or false; then, each vertex of this polytope corresponds to a truth assignment of the proposition $x_1 \wedge x_2 \wedge \cdots \wedge x_n$, where $\wedge$ is the conjunction operator. Hence, a convex hull of them can be seen as a measure of the correlation of these propositions.

*Remark* 1.14. Padberg, instead, in [117] was interested in this polytope because it can be seen as the domain of an unconstrained quadratic binary problem. Since our point of view is similar to that of Padberg, we will often refer to this polytope as the *Boolean Quadric* Polytope (BQP):

$$BQP^n := COR^n.$$

It can be easily shown that this polytope is fully dimensional in the space of symmetric $n$ by $n$ matrices, whose dimension is $n(n+1)/2$. However, the complete description of all its facets is not easy. Several families of facet defining inequalities are found and some of them are exponentially large.

Other similar sets have been studied: Burer and Letchford, in [31] studied the convex hull of feasible solutions of a binary unconstrained QP, that is a problem where $x \in [0,1]^n$ : this set is not a polytope and is called *QPB*. The authors showed some relations between the two sets: in particular, every valid inequality for *BQP* is also valid for *QPB*. Berman and Xu in [7], and Dahl and Haufmann in [43], describe the closely related polytope of completely positive matrices which are expressed as sum of binary rank-1 matrices. Moreover, these polytopes and their relations with other binary polytopes are presented in [103].

## 1.5 Matrix completion problems

A concept which is of particular interest in the second part on this thesis is that of *matrix completion problems*. We write here the definitions along with some important results.

The definitions and results of the matrix completion problem are available in the book [6] and in references therein, which we will highlight at the end of this paragraph.

Let $\mathcal{K}^n$ be a matrix cone in dimension $n$.

**Definition 1.29.** *Given a matrix $A \in \mathbb{R}^{n\times n}$ and a cone $\mathcal{K}^n$ of matrices in dimension $n$. We say that:*

*$A$ is* partial *if some of its entries are not specified.*

*A partial matrix $A$ is* partial symmetric *if it is symmetric in the specified entries and its diagonal is specified.*

*A partial symmetric matrix is* partial-$\mathcal{K}$ *if every principal submatrix of $A$, such that all of its entries are specified, is in $\mathcal{K}$.*

*Given a partial-$\mathcal{K}$ matrix $A$:*

*A* completion *of $A$ is a fully specified matrix $C$ such that $C$ is equal to $A$ in the specified entries of $A$.*

*$C$ is a* $\mathcal{K}$-completion *of $A$ if $C \in \mathcal{K}$.*

*$A$ is* $\mathcal{K}$-completable *if there exist a $\mathcal{K}$-completion of $A$.*

The *$\mathcal{K}$-completion problem* is the problem of finding a completion of a partial-$\mathcal{K}$ matrix. In this context another graph has to be introduced:

**Definition 1.30.** *The* specification graph *of a $n$ by $n$ partial symmetric matrix $A$ is a graph $G$ with vertices $\{1,\dots,n\}$ in which $i$ and $j$ are adjacent if and only if $i \neq j$ and $A_{i,j}$ is specified.*

The completion problem is stated in terms of the specification graph:

**Definition 1.31.** *A graph $G$ is said to be* $\mathcal{K}$-completable *if any matrix whose specification graph is $G$ is $\mathcal{K}$-completable.*

*Remark* 1.15. If a matrix $A$ is $\mathcal{K}$-completable with specification graph $G$, then the specification graph of any completion of $A$ is complete and is the completion of $G$.

Here we state the known results for two aforementioned cones.

**Proposition 1.5.** *A graph is PSD-completable if and only if it is chordal.*

**Proposition 1.6.** *A graph is CPP-completable if and only if it is block-clique.*

The proofs of these statements are shown, respectively, in [87] and in [58].

# Chapter 2

# Introduction to Quadratic and Conic Programming

## 2.1 Context

Mathematical Optimization (or Mathematical Programming) can be seen as the formulation and solution of mathematical models to solve decision problems. The definition is totally generic, because the decision problems can take very different forms. Every decision problem contains an *objective* function to be optimized, subject to some constraints, expressed by a system of conditions to be satisfied. The formulation of a generic optimization problem is:

$$\min \ f(x) \tag{2.1a}$$
$$\text{s. t. } g_i(x) \leq 0, \quad i = 1, \ldots, m \tag{2.1b}$$
$$x \in X \tag{2.1c}$$

where $x \in \mathbb{R}^n$, $X \subseteq \mathbb{R}^n$, $f, g_i : \mathbb{R}^n \mapsto \mathbb{R}$ are the objective function and the constraint functions; the set $\{x \in \mathbb{R}^n \,|\, x \in X, g_i(x) \leq 0, \forall i = 1, \ldots, m\}$ is the domain, or *feasible region*, of the problem.

Mathematical optimization is called Linear Programming (LP) if all the functions in the formulation are linear, and Nonlinear Programming (NLP) otherwise. Both these branches are extremely vast areas of research and have an enormous number of applications. In this thesis we are interested in a particular subset of NLP, which is Quadratic Programming (QP). We will see that is is strongly related to another class of nonlinear programs which is Conic Programming (CP).

QP can be seen as the "simplest" among the nonlinear classes of problems: often a quadratic approximation is used to solve some more difficult nonlinear problems. Because of properties of quadratic functions, specific techniques can be developed for this class of problems: in some cases, they allow to solve the problems very efficiently.

In the next sections we will describe in detail the formulations of general quadratic and conic problems, we show the most important subclasses and we concentrate on those which

are object of study in this thesis. We then dedicate a section to revise the fundamental concepts of duality theory for the nonlinear case, in particular we will show the application to some quadratic and conic cases. Afterwards we will describe the most important first order solution methods and we will focus on column generation technique, central for the work of this thesis. We will then revise other solution methods for quadratic problems; among them there is the Branch and Bound (B&B), which will be used in Chapter 4. We finally introduce the fundamental results in conic programming for the two relevant classes of PSD and COP-CPP cones, which have been object of growing interest in recent years and are related to the work in the second part of this thesis.

## 2.2 Formulations of Quadratic Programs

Given a function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$, and $x \in X$, $f$ is said *quadratic* if it is a polynomial in the components of $x$ of degree 2. In particular, it can be written in the following way:

$$f(x) = \frac{1}{2}x^\top Qx + q^\top x + q_0, \tag{2.2}$$

with coefficients given by a matrix $Q \in \mathbb{R}^{n\times n}$, a vector $q \in \mathbb{R}^n$ and a constant term $q_0 \in \mathbb{R}$.

Quadratic Programming is an extraordinary wide portion of Mathematical Programming. It encloses all the optimization problems in which the objective function, the constraints or both are quadratic functions. This category contains instances of several type and with applications in many different fields, such as, for example, Telecommunications, Finance, Biology, Energy, Robotics, just to cite a few of them (see [71], [85]).

A quadratic program in its most generic formulation can be written as follows:

$$\begin{aligned} \min\ f(x) =& \frac{1}{2}x^\top Qx + q^\top x + q_0 \\ \text{s. t.}\quad & \frac{1}{2}x^\top A_i x + a_i^\top x + \bar{a}_i \le 0, \quad \forall i = 1,\dots,m \\ & l_i \le x_i \le u_i \quad \forall i = 1,\dots,n \\ & x_i \in \mathbb{Z} \quad \forall i \in I, \end{aligned} \tag{2.3}$$

where $n$ is the number of variables, $m$ the number of constraints. $x$, $q$, $a_i \in \mathbb{R}^n$, $\bar{a}_i \in \mathbb{R}$. The constant terms of the constraints are usually moved at the right-hand side of the inequality sign and $b_i := -\bar{a}_i$ are called the *right-hand-side* terms of the constraints. $Q \in \mathbb{R}^{n\times n}$ is a quadratic matrix and $\forall i = 1\dots,m$, $A_i \in \mathbb{R}^{n\times n}$ are the matrices of the constraints. If some of them are null, the corresponding constraints are said to be linear. $-\infty \le l_i \le u_i \le +\infty$ are the (extended) real lower and upper bounds for each variable $x_i$. $I \subseteq \{1,\dots,n\}$ is the set of integer variables.

Both the objective function and the constraint matrices can be assumed symmetric without loss of generality: indeed, one can always replace off diagonal pairs with their average. Formulation (2.3) includes equality constraints as well: indeed, it is sufficient to write two inequality constraints with coefficients of opposite sign.

Without loss of generality we mainly consider minimization problems. Indeed, all maximization problems can be changed into minimization ones by taking the opposite of the objective function. Nevertheless, the sense of the optimization has to be taken into account when considering convexity of the problems: in all the thesis we assume to have only minimization problems, unless where explicitly specified, for the sake of clarity.

The problem (2.3) can be referred to as a Mixed Integer Quadratically Constrained, Quadratic Problem (MIQCQP). A deeper classification can be made depending on specific characteristics of the data. The problems in this class can be:

- *linear problems*, if both the objective function and the constraints are linear;
- *unconstrained*, if there are no constraints; *linearly constrained*, if all the constraints are linear; *quadratically constrained* otherwise;
- *continuous*, *mixed integer* or *integer*, if respectively $I = \emptyset$, $I \subset \{1, \dots, n\}$, or $I = \{1, \dots, n\}$;
- *(mixed) binary* if all the integer variables have lower bound of 0 and upper bound of 1.

Given a mixed integer problem, its *continuous relaxation* is a problem with the same formulation, but without the integrality constraint.

We notice that we refer to linearly constrained quadratic problems simply as QPs, while we write QCQPs in order to specify that not only the objective function, but also the constraints are quadratic.

One other very important distinction has to be made between convex and non convex problems.

**Definition 2.1.** *A problem of type* (2.3) *is said to be* convex *if all the functions in its formulation are convex and the equality constraints are given by linear functions.*

*Remark* 2.1. It is a classic result that a function is convex if its Hessian matrix is positive semidefinite, and it is strictly convex if the Hessian is positive definite. The Hessian of a function as in (2.2) is the matrix $Q$, hence convexity of a quadratic problem is determined by positive definiteness of the matrices in the formulation of the quadratic functions.

*Remark* 2.2. It is worth noting that the inequality constraints in the formulation (2.3) are expressed as lower inequalities between the function and the right-hand-side, which is a given real number. If all these constraints are given by convex functions, they define a convex set: the sub-level set of a convex function. An equivalent definition for a convex problem is that it is a minimization problem of a convex function over a convex domain. If, instead, the inequality had the opposite sign, this would not have been true.

*Remark* 2.3. Since all the linear constraints are convex, as a special case the linearly constrained problems with linear or convex objective function are convex.

Distinguishing if a problem is convex is crucial, because, in general, convex problems are much easier that non convex quadratic problems. As we will see later, indeed, several important classes of continuous convex problems are solvable in polynomial time.

Moreover, also the solution methods for quadratic problems are heterogeneous, because they depend a lot on other features of the instances. We will see the most important techniques in subsequent sections.

## 2.3 Formulations of Conic Programs

Another specific class of nonlinear programming is that of conic problems. We are interested in *linear* conic problems, which can be written as in (2.1), where $f$ is linear, $g_i$ are affine functions and $X$ is a cone. We use the following notation:

$$\begin{aligned} &\min\ c^\top x \\ &\text{s. t. } Ax = b \\ &\qquad x \in \mathcal{K}, \end{aligned} \tag{2.4}$$

where $A \in \mathbb{R}^{m\times n}$, $n, m \in \mathbb{N}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $\mathcal{K}$ is a cone.

If $\mathcal{K}$ is a cone of matrices, as $\mathcal{S}_+$ or $\mathcal{C}$ for instance, we can state the problem using the matrix notation and the Hilbert product: it then becomes:

$$\begin{aligned} &\min\ \langle C, X\rangle \\ &\text{s. t. } \langle A_i, X\rangle = b_i, \quad i = 1, \dots, m \\ &\qquad X \in \mathcal{K}, \end{aligned} \tag{2.5}$$

where $n, m \in \mathbb{N}$, $C$, $A_i \in \mathbb{R}^{n\times n}$, $b_i \in \mathbb{R}$, $\forall i = 1, \dots, m$. Here the variables lie in the space of the symmetric real-valued matrices $\mathcal{S}^n$. It can be equivalently expressed in the following form (although the coefficients would be different):

$$\begin{aligned} &\max\ \sum_{i=1}^{\tilde{m}} \tilde{b}_i y_i \\ &\text{s. t. } \tilde{C} - \sum_{i=1}^{\tilde{m}} \tilde{A}_i y_i \in \mathcal{K}, \end{aligned} \tag{2.6}$$

where the variables are $y_i$, $i = 1, \dots, \tilde{m}$. It is shown, for instance, in [52] (Section 1.2.2). Two main classes of linear conic optimization are well known, and they are defined based on the type of cones which are considered. The first one is the *semidefinite programming (SDP)* and the second one is *copositive programming (COP)*. In the first case $\mathcal{K} = \mathcal{S}^n_+$ and in the second case $\mathcal{K} = \mathcal{C}^n$ or $\mathcal{C}^{*n}$. As we will see in the next section, the notion of dual cone is fundamental; we recall that $\mathcal{S}^n_+$ is self dual and the dual of $\mathcal{C}^n$ is $\mathcal{C}^{*n}$.

Among the semidefinite programming problems, the subclass of the Second Order Cone Programming (SOCP) plays an important role. Here the cone $\mathcal{K}$ is the Cartesian

product of several cones $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_N$, where each of the cones $\mathcal{K}_i$, $i = 1, \ldots, N$ is the *second order cone*:

$$\mathcal{K}_i = \{x = (x_0, \bar{x}),\, \bar{x} \in \mathbb{R}^{n_i},\, | \, x_0 \geq ||\bar{x}||\}.$$

$||\bar{x}||$ is the Euclidean norm in $\mathbb{R}^{n_i}$. This class of problems is well-known and has several applications. Specific algorithms are formulated to solve problems with this form. For a more detailed description, see [1].

## 2.4 Duality theory

Duality is one of the most important features in mathematical optimization. Here we recall duality for nonlinear problems and we specifically derive the formulation for the dual of a linearly constrained quadratic problem and a linear conic problem. We also notice that duality for linear problems arises naturally, as a particular case. For this section we mainly refer to [23], but also [3] and [9] are good references.

We consider a generic nonlinear problem in the following form:

$$\begin{aligned} &\min\ f(x) \\ &\text{s. t. } g_i(x) \leq 0, \quad i = 1, \ldots, m \\ &\qquad h_i(x) = 0, \quad i = 1, \ldots, p. \end{aligned} \tag{2.7}$$

We assume that $f, g_i, h_i : D \subseteq \mathbb{R}^n \to \mathbb{R}\, \forall i$, and $D$ is nonempty. Let $F \subseteq D$ be the feasible region of the problem. The *Lagrangian function* is the following function $L : D \times \mathbb{R}^m_+ \times \mathbb{R}^p \mapsto \mathbb{R}$:

$$L(x, \lambda, \mu) := f(x) + \lambda^\top g(x) + \mu^\top h(x), \tag{2.8}$$

where $\lambda \in \mathbb{R}^m_+$ and $\mu \in \mathbb{R}^p$ are the so-called *Lagrangian multipliers* of each constraint and are chosen so that, for every feasible point $x \in F$,

$$L(x, \lambda, \mu) \leq f(x) \quad \forall \lambda \in \mathbb{R}^m_+,\, \forall \mu \in \mathbb{R}^p. \tag{2.9}$$

*Remark* 2.4. From the definition (2.8), clearly $L(x, 0, 0) = f(x)$ for every $x \in F$. Then, observing (2.9), it follows that $\max_{\lambda \geq 0, \mu} L(x, \lambda, \mu) = f(x)$, for every feasible point $x \in F$. Moreover, it is clear that if $x \in D \setminus F$, then $\exists i$ such that $g_i(x) > 0$ or $h_i(x) \neq 0$. In both cases, $\sup_{\lambda \geq 0, \mu} L(x, \lambda, \mu) = +\infty$, because the corresponding multiplier $\lambda_i$ or $\mu_i$ can grow to $+\infty$ (or $-\infty$, if $h_i(x) < 0$). Hence, $\min_{x \in D} \sup_{\lambda \geq 0, \mu} L(x, \lambda, \mu)$ is attained with $x \in F$. Therefore, problem (2.7) can be written as:

$$\min_{x \in D} \sup_{\lambda \geq 0, \mu} L(x, \lambda, \mu). \tag{2.10}$$

We can now introduce the *Lagrange dual function*, which is the infimum of $L$ over $x$:

$$u(\lambda, \mu) := \inf_{x \in D} L(x, \lambda, \mu). \tag{2.11}$$

*Remark* 2.5. If $f^*$ is the optimal value of the original problem, easy calculations (reported, for instance, in [23]) show that $u(\lambda, \mu) \leq f^*$, for each $\lambda$ and $\mu$, so this function always provides a lower bound for the optimal value of the problem.

Finally, we can define the *Dual problem* of (2.7), which is:

$$\begin{aligned} \max\ & u(\lambda, \mu) \\ \text{s. t. } & \lambda \in \mathbb{R}^m_+ \\ & \mu \in \mathbb{R}^p \end{aligned} \tag{2.12}$$

The original problem (2.7) is now called *primal* and the Lagrangian multipliers are called *dual variables*.

*Remark* 2.6. We can see that, from Remark 2.5, the optimal point of the dual problem (2.12) is lower than, or equal to, the optimum of the primal problem. This is the *weak duality property*.

*Remark* 2.7. We would like to notice that the definitions of Lagrangian dual function and dual problem do not require any hypothesis on the convexity of the primal problem. However, for nonconvex problems, the infimum in (2.11) can be $-\infty$ for every $\lambda, \mu$: in this case, the dual problem provides a lower bound which is trivial and useless, as pointed out, for instance, in [102].

It is worth noticing which are the conditions for the equality between the primal and the dual optimal values. This property is called *strong duality* and requires more conditions, called *constraint qualification*. Typical constraint qualification conditions are the *Slater's conditions*:

**Definition 2.2.** *A problem of the form* (2.7) *satisfies the Slater's conditions if the problem is convex,* $\forall\, i = 1, \ldots, p$, $h_i$ *are affine functions, and* $\exists x \in ri(D) : g_i(x) < 0$ $\forall\, i = 1, \ldots, m$ *s.t.* $g_i$ *is not affine.*

If a problem (2.7) satisfies the Slater's conditions, then strong duality holds.

*Remark* 2.8. For linearly constrained convex problems, these conditions are always satisfied. For more general problems, like convex QCQPs or linear conic problems, these conditions must be verified case by case.

### 2.4.1 Specific cases

In order to write conveniently the dual problem, it is worth noting that this problem is the maximization of an infimum: the value of $u(\lambda, \mu)$ could be $-\infty$. It is clear that the points $x \in D$ such that $\inf_{x \in D} L(x, \lambda, \mu) = -\infty$ must be discarded, and so this appears as a constraint in the dual. Here the cases of a linearly constrained quadratic problem and a linear conic problem are presented.

### Dual of a QP

Suppose that the primal problem P has the following form:

$$\begin{aligned} \min\ & \frac{1}{2}x^\top Qx + c^\top x \\ s.t.\ & Ax = b \\ & x \geq 0. \end{aligned} \tag{2.13}$$

The case with also inequality constraints is similar. We add Lagrangian multipliers $\lambda$ for the equality constraints and $\mu$ for the nonnegativity constraints. Then the Lagrangian function is:

$$\begin{aligned} L(x, \lambda, \mu) &= \frac{1}{2}x^\top Qx + c^\top x + \lambda^\top (Ax - b) - \mu^\top x \\ &= \frac{1}{2}x^\top Qx + (c + A^\top \lambda - \mu)^\top x - \lambda^\top b. \end{aligned}$$

If we consider $L$ as a function of $x$, it is quadratic and its domain is $D = \mathbb{R}^n$, hence its infimum is either attained at a stationary point, or is $-\infty$. If the problem is not convex, its minimum is $-\infty$ for any value of $\lambda$ or $\mu$ and the dual problem is useless. Hence, here we assume that the problem is convex. Since our goal is to maximize its infimum, we force it to be a stationary point, by fixing $\nabla_x L(x, \lambda, \mu) = 0$. This translates in the following constraint:

$$c + A^\top \lambda - \mu = -Qx, \quad \text{or} \quad \mu = Qx + A^\top \lambda + c \geq 0.$$

By substituting it in the expression of $L$ and adding it as a constraint, the dual problem is then:

$$\begin{aligned} \max\ \ L(x, \lambda) = -\frac{1}{2}x^\top Qx - \lambda^\top b \\ s.t.\quad Qx + A^\top \lambda + c \geq 0 \\ Ax = b \\ x \geq 0. \end{aligned}$$

We note that, if Q is the null matrix, the problem is linear and the original variabes $x$ do not appear in the dual, which is the classic linear dual problem.

### Dual of a CP

If the primal problem has the form in (2.5), then the Lagrangian function is

$$L(X, \lambda) = \langle C, X \rangle + \sum_{i=1}^{m} \lambda_i (\langle A_i, X \rangle - b_i),$$

where $\lambda_i \geq 0 \, \forall \, i = 1, \ldots, m$ are the Lagrangian multipliers. Now,

$$\inf_{X \in \mathcal{K}} L(X, \lambda) = \inf_{X \in \mathcal{K}} \langle C + \sum_{i=1}^{m} \lambda_i A_i, X \rangle - \lambda^\top b.$$

It is easy to see that this infimum is $> -\infty$ if and only if

$$C + \sum_{i=1}^{m} \lambda_i A_i \in \mathcal{K}^*,$$

by definition of the dual cone. In this case,

$$\inf_{X \in \mathcal{K}} L(X, \lambda) = 0 - \lambda^\top b.$$

Hence, the dual problem is

$$\begin{aligned} \sup \quad & -\lambda^\top b \\ s.t. \quad & C + \sum_{i=1}^{m} \lambda_i A_i \in \mathcal{K}^* \\ & \lambda \geq 0. \end{aligned}$$

Again, if $\mathcal{K} = \mathbb{R}^n$, since it is self-dual, we obtain the classic linear dual problem.

## 2.5 Solution methods

The solution of quadratic or conic problems require different techniques and has different computational complexity depending on the type of problems. Regarding conic optimization, it has been shown that the complexity of SDP is polynomial: results obtained with the interior point method are given by Nesterov and Nemirowski [112] and a good description of semidefinite programming is in [135]. Nevertheless, copositive programming is in general NP-hard, as we will see in a dedicated section.

Quadratic programming is in general NP-hard, (it is sufficient to see that it includes Max cut problems, among others) but subclasses of it can be polynomial as well. Indeed, it has been shown that several convex continuous problems, such as LPs, convex QPs, SOCPs can be reformulated as SDP problems -hence they are polynomial- and specific interior point methods have been developed for these subclasses: see [1, 112, 120, 135]. More detailed sections describing solution methods are below: we firstly concentrate on first order and feasible direction methods, because they are preliminary to column generation techniques, as we will see, and on Branch and bound (B&B), which will be used in the first part to solve mixed binary quadratic problems. Then, we will examine other important techniques.

## 2.6 First order methods for quadratic problems

In this section we briefly recall some classic methods that are used in unconstrained optimization and some standard first order methods. They serve as an introduction and they are also exploited in the first result described in Chapter 3. In the following sections we will describe in detail other methods which are extensively used in this thesis. The first part of this section is based on [9] and [114].

Typically in nonlinear optimization, the first important distinction is made between constrained and unconstrained problems. In unconstrained optimization, specifically when the problem is quadratic, a useful strategy is that of iterative methods.

Every algorithm for unconstrained optimization generates sequences of points $x_0, x_1, \ldots$ which converge to the optimal point, up to a certain accuracy. In order to generate successive iterates, these methods use information on the objective function, which in this thesis is always differentiable. The main idea is to find so-called *descent directions*:

**Definition 2.3.** *$d \in \mathbb{R}^n$ is a* descent direction *for a function $f : \mathbb{R}^n \mapsto \mathbb{R}$, if for every $x \in \mathbb{R}^n$, $\exists \alpha > 0$ s.t. $f(x + \alpha d) < f(x)$.*

A typical descent direction at a point $x$ is clearly $d = -\nabla f(x)$. It can easily be seen that this is the direction which best improves the value of the function. Methods of the *steepest descent* calculate this direction at each iterate $x$ and use techniques to choose a suitable step length $\alpha$.

For convex problems, a different unconstrained optimization algorithm based on descent directions is the so-called *conjugate directions* method. We describe some details of this method, which are strongly exploited in Chapter 3.

**Definition 2.4.** *Given a $n \times n$ symmetric, positive definite matrix $Q$, let $d_1, \ldots, d_n$ be $n$ distinct vectors in $\mathbb{R}^n$. They are called* conjugate directions *w.r.t. $Q$ if*

$$d_i^\top Q d_j = 0 \quad \forall i \neq j. \tag{2.14}$$

It is important to note that the following proposition holds:

**Proposition 2.1.** *Let $d_1, \ldots d_m \in \mathbb{R}^n$ be $m$ nonzero vectors, which are mutually conjugate with respect to a symmetric and positive definite $n \times n$ matrix $Q$. Then, $d_1, \ldots d_m$ are linearly independent.*

In particular, $n$ mutually conjugate directions form a basis of $\mathbb{R}^n$. Moreover, the following proposition shows how powerful the conjugate directions are:

**Proposition 2.2.** *Let $Q \in \mathbb{R}^{n \times n}$ a symmetric positive definite matrix and $d_1, \ldots d_n \in \mathbb{R}^n$ be $n$ nonzero and mutually conjugate directions with respect to $Q$. Given a quadratic function*

$$f = \frac{1}{2} x^\top Q x + c^\top x, \tag{2.15}$$

*we define the algorithm:*

$$x_{k+1} := x_k + \alpha_k d_k, \; \forall k = 0, \ldots, n, \tag{2.16}$$

*where $x_0$ is any point in $\mathbb{R}^n$ and*

$$\alpha_k = -\frac{\nabla f(x_k)^\top d_k}{d_k^\top Q d_k} \tag{2.17}$$

*is the coefficient such that $x_{k+1}$ minimizes $f$ along the line $x_k + d_k$. For each $k = 1, \ldots, n$, let $\mathcal{S}_k \subset \mathbb{R}^n$ be the linear subspace generated by the vectors $d_1, \ldots, d_k$.*
*Then, each $x_k$ is the optimal point of $f$ in the affine space $x_0 + \mathcal{S}_k$.*

As a consequence:

**Proposition 2.3.** *Conjugate direction method converges to the minimum point of a strictly convex quadratic function $f(x) : \mathbb{R}^n \to \mathbb{R}$ in at most $n$ steps.*

Based on these properties the *conjugate gradient* is a method which was firstly introduced in the 1950s to solve linear systems of equations. Equivalently, it can be used for quadratic strictly convex unconstrained optimization. A variant of this method, introduced by Fletcher and Reeves in [63] is a technique to solve nonlinear optimization problems. Conjugate gradient methods are based on the following idea. In order to generate subsequent conjugate directions, at each step compute the gradient of the function in the current point, and make it conjugate with respect to the previously generated directions. Then, following the conjugate direction method, find the minimal point of the function along the direction and proceed. The great advantage of using the gradient is given by an important property: it is sufficient to make it conjugate with respect to one single previous direction to have it conjugate with respect to all the others. Conjugate gradient methods can be adapted to solve problems where the matrix of the objective function is positive semidefinite.

If we consider constrained optimization, these methods cannot be directly applied, because moving along descent directions may lead to infeasible points. In this context, the fundamental approach is that of *feasible direction* methods: given a feasible point $x_k$, the direction $d_k$ is generated so that $\exists \alpha > 0$ such that the point $x_k + \alpha d_k$ is feasible and has a smaller objective function value than $x_k$. In this class there are the method of Zoutendijk and its variants (see [3]). Then, an important strategy is that of *projected gradient* methods. The basic idea is to project the gradient in such a way that the direction is a descent direction and maintains feasibility. Several methods have been studied, starting from the projected gradient of Rosen [124] in 1960 for linear constrained problems, and its generalizations for problems with nonlinear constraints. Other feasible direction methods are the reduced gradient projection and its generalized variant. A detailed description can be found for instance in [3, 9].

One other classic strategy is a first order method for solving linearly constrained, non linear problems: the Frank-Wolfe algorithm.

### 2.6.1 Frank-Wolfe method

The idea of Frank-Wolfe method dates back to 1956 (see [69]), but this algorithm received a lot of success and is widely used. For instance, as shown in [118], it is often

used for network flow problems. It alternates between a linear subproblem and a line search in the segment between the current point and the solution of the subproblem. In fact, given an iterate $x_k$, the subproblem is

$$\begin{array}{ll} \min & f(x_k) + \nabla f(x_k)^\top (x - x_k) \\ \text{s.t.} & x \in X \end{array} \tag{2.18}$$

and it is the linearization of the original one. It provides a descent direction and, if $f$ is convex, this is a lower bound of the original problem. The next iterate $x_{k+1}$ is the minimum along the line segment between $x_k$ and the optimum of problem (2.18).

This algorithm is simple and efficient in the first iterations, but the convergence rate is sub-linear.

In order to deal with large-size problems, especially with a large number of columns, a more useful strategy is to use a decomposition method and in particular *column generation* algorithms. Column generation has been firstly introduced for Linear Programming, but it can be applied to Quadratic Programming and more generally to Nonlinear Programming, under some assumptions on the convexity of the problem. We will show these results in the next chapter.

## 2.7 Column Generation methods

Column generation formulations for linear programs have been widely studied. A survey is given by Barnhart et al. in [2] and many papers have been written on this subject. Just to cite a few examples, see [49], [50], [96], [105], [111], [136], [138]. The basic idea behind column generation algorithms is the following. Consider a formulation of a problem in which the number of variables (columns) is large (exponential, for instance). Then, one can start working with a small subset of them and find the optimal solution of the restricted problem. By solving a subproblem, it is possible to determine if there are other columns which, if added to the formulation, can provide a lower objective function. If so, add iteratively one new variable that improves the objective function until a certificate of optimality can be obtained.

In order to better explain how this strategy is developed, we consider the most typical case when column generation is applied: the Dantzig-Wolfe Decomposition.

### 2.7.1 Dantzig-Wolfe Decomposition (DWD)

The Dantzig-Wolfe Decomposition (firstly introduced in [44]) is based on the Minkowski-Weyl Theorem: every polyhedron is the convex combination of its extreme points plus the conic combination of its extreme rays. This theorem is exploited in the following way

(see [136] or [50] for further details). Take a linear problem in the form, for instance:

$$\begin{aligned} &\min c^\top x && (2.19)\\ \text{s. t.}\quad &Ax \geq b, \\ &x \in X, && (2.20)\end{aligned}$$

where $X$ is a polyhedron. The condition $x \in X$ is replaced by imposing that $x$ must be expressed as a convex combination of the extreme points $x_p$ of $X$ plus a conic combination of the extreme rays $x_r$ of $X$: if $\mathcal{P}$ and $\mathcal{R}$ are respectively the index sets of the extreme points and extreme rays of $X$,

$$x = \sum_{p\in\mathcal{P}} x_p\lambda_p + \sum_{r\in\mathcal{R}} x_r\lambda_r, \quad \sum_{p\in\mathcal{P}} \lambda_p = 1, \quad \lambda_p \geq 0, \lambda_r \geq 0 \quad \forall p \in \mathcal{P}, r \in \mathcal{R}. \tag{2.21}$$

Then, this expression is substituted in the rest of the constraints and in the objective function. The result is the following:

$$\begin{aligned} \min\ & \sum_{p\in\mathcal{P}} c_p\lambda_p + \sum_{r\in\mathcal{R}} c_r\lambda_r && (2.22)\\ \text{s. t.}\quad & \sum_{p\in\mathcal{P}} a_p\lambda_p + \sum_{r\in\mathcal{R}} a_r\lambda_r \geq b \\ & \sum_{p\in\mathcal{P}} \lambda_p = 1 \\ & \lambda_p \geq 0 \quad \forall p \in \mathcal{P} \\ & \lambda_r \geq 0 \quad \forall r \in \mathcal{R}, \end{aligned}$$

where $c_j = c^\top x_j$ and $a_j = Ax_j$, for $j \in \mathcal{P} \cup \mathcal{R}$.

A reduced master program (RMP) considers only a small subset of the sets $\bar{\mathcal{P}} \subset \mathcal{P}$ of extreme points and $\bar{\mathcal{R}} \subset \mathcal{R}$ of extreme rays:

$$\begin{aligned} \min\ & \sum_{p\in\bar{\mathcal{P}}} c_p\lambda_p + \sum_{r\in\bar{\mathcal{R}}} c_r\lambda_r && (2.23)\\ \text{s. t.}\quad & \sum_{p\in\bar{\mathcal{P}}} a_p\lambda_p + \sum_{r\in\bar{\mathcal{R}}} a_r\lambda_r \geq b, \\ & \sum_{p\in\bar{\mathcal{P}}} \lambda_p = 1 \\ & \lambda_p \geq 0 \quad \forall p \in \bar{\mathcal{P}} \\ & \lambda_r \geq 0 \quad \forall r \in \bar{\mathcal{R}}. \end{aligned}$$

Given an optimal dual solution $\pi$ and $\pi_0$ of the current RMP, where variables $\pi$ is the dual of the first set of constraints and variable $\pi_0$ is the dual of the second constraint, the subproblem is to determine:

$$\min_{j \in \mathcal{P} \cup \mathcal{R}} c_j - \pi^\top a_j - \pi_0.$$

By our previous linear transformation this results in the following, called *Pricing problem*:

$$\min (c^\top - \pi^\top A)x - \pi_0 \tag{2.24}$$

$$\text{s. t.} \quad x \in X. \tag{2.25}$$

The DWD strategy can be naturally extended to non linear problems, based of the duality theory of non linear problems. A method which can derived directly by applying this framework, but which historically have been discovered differently, is the *Simplicial Decomposition* algorithm, which has been deeply analyzed in this thesis and will be presented in detail in the next section.

## 2.8 Simplicial Decomposition

A significant part of this thesis concerns the study and the adaptation and application of a specific column generation algorithm, namely *Simplicial Decomposition* algorithm.

Simplicial Decomposition (SD) represents a class of methods used for dealing with convex problems. It was first introduced by Holloway in [94] and then further studied in other papers (see, e.g., [93, 139, 140]). A complete overview of this kind of methods can be found in [118].

This method has been developed as an extension of the Frank-Wolfe algorithm, but it can be seen also as a particular case of decomposition based on the DWD for nonlinear problems. In fact, it is based on the Caratheodory theorem, that states that any compact, convex set $X$ is the union of all simplices whose vertices belong to $X$. The idea is the following: in order to solve the original convex problem, a sequence of columns, or extreme points, is generated, based on the linearization of the original objective funtion; the domain of the problem is reduced to the convex hull of some of these columns, the problem is solved, a new extreme point is added and the new solution is found; this, until a guarantee that the global optimum is found. So, this strategy is an inner approximation of the original domain. The generation of new columns is based on the linearization of the objective function in the optimum of the current reduced problem. So, in order to solve the original problem, it is decomposed into simpler ones, which are called respectively pricing and master programs, and are solved alternatingly and repeatedly. The pricing solves the original problem with a linear objective function and finds a new extreme point, while the master program, instead, is a problem with the original objective function, but with lower dimension and simplified constraints.

More specifically, we consider a minimization problem of the following form:

$$\min f(x) \tag{2.26}$$

$$\text{s. t.} \quad x \in X$$

where $f$ is a continuous, convex function and $X \subset \mathbb{R}^n$ is a convex and compact set.

The column generation is done in the following way: starting from a single point, the domain of each master program is the convex hull of a finite set of affinely independent points, i.e. a simplex, and these points are the solution of the previous pricing problems.

In practice, the feasible set $X$ is approximated with the convex hull of an ever expanding finite set $X_k = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_m\}$ where $\tilde{x}_i$, $i = 1, \ldots,$ $m$ are extreme points of X. We denote this set with $conv(X_k)$:

$$conv(X_k) = \{x \mid x = \sum_{i=1}^{m} \lambda_i \tilde{x}_i, \sum_{i=1}^{m} \lambda_i = 1, \lambda_i \geq 0\} \tag{2.27}$$

At each iteration, it is possible to add new extreme points to $X_k$ in such a way that a function reduction is guaranteed when minimizing the objective function over the convex hull of the new (enlarged) set of extreme points. If the algorithm does not find at least one new point, the solution is optimal and the algorithm terminates.

The use of the proposed method is particularly indicated when the following two conditions are satisfied:

1. Minimizing a linear function over $X$ is much simpler than solving the original nonlinear problem;

2. Minimizing the original objective function over the convex hull of a relatively small set of extreme points is much simpler than solving the original nonlinear problem (i.e. tailored algorithms can be used for tackling the specific problem in our case).

The first condition is needed due to the way a new extreme point is generated. Indeed, this new point is the solution of the following linear programming problem

$$\begin{array}{ll} \min & \nabla f(x_k)^\top (x - x_k) \\ \text{s.t.} & x \in X \end{array} \tag{2.28}$$

where a linear approximation calculated at the last iterate $x_k$ (i.e. the solution obtained by minimizing $f$ over $conv(X_k)$ ) is minimized over the original feasible set $X$.

Below, we report the detailed scheme related to the classical simplicial decomposition algorithm [10, 118, 140] (see Algorithm 1). At a generic iteration $k$ of the simplicial decomposition algorithm, given the set of extreme points $X_k$, we first minimize $f$ over the set $conv(X_k)$ (Step 1), thus obtaining the new iterate $x_k$ then, at Step 2, we generate an extreme point $\tilde{x}_k$ by solving the linear program (2.30). Finally, at Step 3, we update $X_k$.

Finite convergence of the method is stated in the following Proposition (see, e.g., [10, 140]):

**Proposition 2.4.** *Simplicial Decomposition algorithm obtains a solution of Problem* (2.26) *in a finite number of iterations, if $X$ is polyhedral.*

As already written in [140], a *vertex dropping rule* is also used to remove those vertices in $X_k$ whose weight is zero in the expression of $x_k$, the solution of the master. This

**Algorithm 1** Simplicial Decomposition Algorithm

**Initialization:** Choose a starting set of extreme points $X_0$.

**For** $k = 1, 2, \ldots$

**Step 1)** Generate iterate $x_k$ by solving the **master problem**

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in conv(X_k) \end{array} \tag{2.29}$$

**Step 2)** Generate an extreme point $\tilde{x}_k$ by solving the **subproblem**

$$\begin{array}{ll} \min & \nabla f(x_k)^\top (x - x_k) \\ \text{s.t.} & x \in X \end{array} \tag{2.30}$$

**Step 3)** If $\nabla f(x_k)^\top (\tilde{x} - x_k) \geq 0$, **Stop**. Otherwise Set $X_{k+1} = X_k \cup \{\tilde{x}_k\}$

**End For**

dropping phase does not modify the formulation of the pricing, so it does not change the steps of the algorithm. However, it can guarantee significant savings in terms of CPU time, since it keeps the dimensions of the master problem small, and thus justifies the name of the algorithm, because it keeps the domain of the master a simplex.

The most significant advantage of Simplicial Decomposition is that the dimension of the master programs are always small: in practice, the maximal dimension of the master problems is hardly bigger than a few hundreds, (and often it is much smaller) hence it is generally much smaller than the dimension of the original problem. Thus, the overall computing time can be reduced with this decomposition. Another extremely important feature of this technique is that all the constraints are in the pricing, so, since the problem is convex, all the reduced master programs are feasible and no dual information is necessary.

It is worth noting one more characteristic, that will be particularly helpful for mixed integer problems. Since in the pricing problem a linearization of the original objective function is done, and the original function is convex, the optimization of the linear function always provides a valid lower bound.

### 2.8.1 Dantzig-Wolfe and Simplicial decompositions

In principle, DWD can be applied to convex nonlinear problems. If we apply it to a convex QP in the form (2.13) and we reformulate all the constraints with the Minkowsly-Weyl theorem, we express the feasible region as the convex combination of all its extreme points. If we apply column generation to solve the problem, we can easily see that the restricted master problem is exactly the same as in SD algorithm. Indeed, if we write it,

we have:

$$\begin{aligned} \min \quad & \frac{1}{2}x^\top Qx + c^\top x \\ s.\ t. \quad & x - B\lambda = 0 \qquad [\pi] \\ & e^\top \lambda = 1 \qquad [\pi_0] \\ & \lambda \geq 0, \end{aligned}$$

where $B = (\tilde{x}_1 \dots \tilde{x}_k) \in \mathbb{R}^{n \times k}$ is the matrix whose columns are the extreme points, and $e$ is the all-1 vector. $\pi$ and $\pi_0$ are the dual variables corresponding to the constraints. The (quadratic) dual is:

$$\max \quad -\frac{1}{2}x^\top Qx + \pi_0 \tag{2.31a}$$

$$s.\ t. \quad -Qx + \pi = c \tag{2.31b}$$

$$-\pi^\top \bar{x}_j + \pi_0 \leq 0, \quad \forall j = 1, \dots, k. \tag{2.31c}$$

Then, the pricing problem is:

$$\begin{aligned} \min \quad & {\pi^*}^\top x - \pi_0^* \\ s.\ t. \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

where $\pi^*$ and $\pi_0^*$ are the optimal variables. We note that constraint (2.31b) implies that $\pi = \nabla f(x)$, for every dual feasible $x$ and $\pi$, in particular for the optimal $x^*$ and $\pi^*$. Hence, the pricing can be rewritten as:

$$\begin{aligned} \min \quad & \nabla f(x^*)^\top x + const \\ s.\ t. \quad & Ax = b \\ & x \geq 0, \end{aligned}$$

which is equivalent to the subproblem of SD.

## 2.9 Convex quadratic programs

Other classic solution methods for a general convex quadratic program can be mainly categorized into either interior point methods or active set methods [115]. In interior point methods, a sequence of parameterized barrier functions is (approximately) minimized using Newton's method. The main computational burden is represented by the calculation of the Newton system solution (used to get the search direction). Even if those methods are relatively recent (they started becoming popular in the 1990s), a large number of papers and books exist related to them (see, e.g., [78, 112, 142–144]).

In active set methods, at each iteration, a working set that estimates the set of active constraints at the solution is iteratively updated. This gives a subset of constraints to

watch while searching the solution (which obviously reduces the complexity of our search in the end). Those methods, which have been widely used since the 1970s, turn out to be effective when dealing with small- and medium-sized problems. They usually guarantee efficient detection of unboundedness and infeasibility (other than returning an accurate estimate of the optimal active set). An advantage of active set methods over interior points is that they are well-suited for warmstarts, where a good estimate of the optimal active set or solution is used to initialize the algorithm. This turns out to be extremely useful in applications where a sequence of QP problems is solved, e.g., in a sequential quadratic programming method. A quite large number of active set methods have been developed in recent years (see, e.g., [41, 42, 46, 62, 90]). A detailed overview of active set methods can be found in [115].

## 2.10 Branch & Bound for Mixed Integer convex QPs

In many applications the requirement that some or all the variables must be integer is needed. This makes the problem more complicated to solve than its continuous counterpart. Its formulation is then the following:

$$\begin{aligned} \min\ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.}\ & Ax \geq b, \\ & Cx = d, \\ & l \leq\ x \leq u \\ & x_i \in \mathbb{Z} \quad \forall i \in I \subseteq \{1, \ldots, n\}, \end{aligned} \tag{2.32}$$

with the same notation as above.

We would firstly distinguish the subclass of mixed integer linear problems (MILPs), that is when the matrix $Q$ in the objective function is null, among the MIQPs. Even if the integrality constraints are clearly non convex constraints, MIQPs are usually called *convex* if their continuous relaxation is convex. A variety of exact solution methods for Mixed Integer Problems exists, as shown for example in [32]. The integrality constraint is typically addressed with a technique called Branch and bound (B&B), that has been extensively used in the literature for Mixed Integer Linear and Quadratic Problems.

It has been developed for MILPs, but can be adapted to convex MIQPs. It is based on a key operation, called *branching*: if an integer constrained variable $x_i$ takes a fractional value $x_i^*$ in the optimal solution of the continuous relaxation of the problem, then one can replace the problem with two subproblems. In one of the subproblems, the constraint $x_i \leq \lfloor x_i^* \rfloor$ is added, while in the other the constraint $x_i \geq \lceil x_i^* \rceil$ is added. In this way, a tree structure is obtained, where each node has two children. Clearly, the solution of the original problem is not feasible for the subproblems, but it provides a lower bound for them. The nodes can be removed, or *pruned*, due to three possible reasons. The first one is optimality: if the solution of the relaxation is feasible for the original problem, then it is both a lower and an upper bound, and no better solutions will be found in the subtree,

so it is pruned. Then, one other possibility is that the problem obtained by the split is infeasible, and in this case the node is pruned for infeasibility. The latter, but most frequent case, is that of pruning by bound: this is the case when the lower bound given by the solution of the continuous relaxation is higher than a known feasible solution, that in minimization problems is always an upper bound. In this case the node is pruned too, because in the subtree there cannot be any optimal point. No other cases can happen. The strategy, then, in order to solve efficiently an integer problem, is that of finding the best search of the tree in order to generate the least number of nodes by pruning the subtrees where an optimum cannot be present, while exploring rapidly the part of the tree that leads to the optimal point. To this aim a lot of different techniques have been developed, but which depend largely on the type of the instance.

When the number of constraints is high, the Branch and bound technique can be extended to the Branch and Cut, whose basic idea is that some constraints are not taken into account in the formulation, but are added during the branching in order to determine if the current optimal point is feasible or not: if not feasible, one excludes it from the feasible region, by adding a valid cut, and improves the formulation. Moreover, one other method is the branch and price: it is the combination of branch and bound with the decomposition technique described above and it will be explained in the next paragraph.

Indeed, Column Generation can be applied to mixed integer problems, as reported for instance in [136] and [138]. Given a problem with some integer constrained variables, one can reformulate a subset of the constraints with the Minkowski-Weyl Theorem and rewrite them in the rest of the constraints and in the objective function. Then, one obtains a new integer problem, where the integrality constraints link the original variables to the new ones. The combination of column generation in a branch and bound tree is called *Branch and price*. One reason for using this reformulation is that it can be tighter, due to the reformulation, so the lower bound in the nodes of the branch and bound tree is likely to be higher than that obtained with the original formulation.

Finally, in order to deal efficiently with convex quadratic problems with integer variables and exploit their properties, more refined techniques have been developed. One of them is the use of suited ellipsoids in order to improve the bound given by the continuous relaxation. This method consists of centering an ellipsoid in the optimal point of the continuous relaxation, and expanding it until it contains an integer point in the boundary and no integer points in the interior. Evaluating the objective function on it yields a stronger lower bound which can improve the B&B algorithm. A more detailed description, along with promising results, can be found in [25].

## 2.11 Mixed Integer Quadratically Constrained Quadratic Problems (MIQCQPs)

In this section we deal with a more generic class of quadratic programs.

A generic quadratic problem that we are interested in here takes the form of (2.3),

here reported:

$$\begin{aligned} \min\ f(x) =& \frac{1}{2}x^\top Qx + q^\top x + q_0 \\ \text{s. t.}\quad & \frac{1}{2}x^\top A_i x + a_i^\top x + \bar{a}_i \leq 0, \quad \forall i = 1, \dots, m \\ & l_i \leq x_i \leq u_i \quad \forall i = 1, \dots, n \\ & x_i \in \mathbb{Z} \quad \forall i \in I, \end{aligned} \tag{2.33}$$

where the notation is the same as introduced before. In particular, it may contain quadratic constraints and it can be non convex. This is the most general class of quadratic problems and can be seen as a particular class of Mixed Integer Non Linear Problems (MINLP).

In order to deal with these difficulties, some ad-hoc methods have been proposed. As written in the survey [32], a typical strategy is the use of under and over estimators, that is, the substitution of non convex constraints with convex functions that are lower bounds for them, or respectively with concave upper bounds. Another typical technique is linearization: the dimension of the variable space can be extended to $\mathcal{O}(n^2)$, taking into consideration new variables $y$ that represent the products between $x$ variables, and a new set of constraints is added to make them consistent. Another technique is convexification: the original non convex function is modified in order to make it convex, so more tractable. The typical example arises in quadratic purely binary optimization: the binary constraints can be equivalently rewritten as $x_i^2 = x_i$ for $i \in \{1, \dots, n\}$. This relations can be exploited in order to add a term in the diagonal of the matrix in the objective function (and subtract it to the linear part) in order to make the problem convex. This is the basic concept of convexification proposed by Hammer and Rubin in [92] and it has been successively extended to the Quadratic Convex Reformulation (QCR) methods (see [13] and [14]).

Finally, for non convex non linear problems, a relevant method is spatial branch and bound. This method can be used in addition to the classical one, but it makes the branching on the continuous variables: this is done when under or over estimators are used to approximate a non convex function. If a variable $y$ has lower and upper bounds $l < u$, then one can split the problem into two subproblems $l \leq y \leq \beta$ and $\beta \leq y \leq u$. In this way, the under and upper approximations can be strengthened, taking advantage on the reduced domain.

### 2.11.1 Extended space for solving MIQCQPs

A standard technique used to deal with MIQCQPs is to work on the extended space representing the products of the original variables. Instead of the vector of variables $x$, one considers the matrix variable $X$ that is given by

$$X = \begin{pmatrix} 1 \\ x \end{pmatrix} \begin{pmatrix} 1 & x^\top \end{pmatrix} = \begin{pmatrix} 1 & x^\top \\ x & xx^\top \end{pmatrix}.$$

In this way, all the quadratic terms are linearized, at the price of having $\mathcal{O}(n^2)$ variables instead of $n$. The formulation of the problem is given as a linear problem in the extended

space, with a constraint that links together $x$ and $X$. We use the following notation, along with the Hilbert product of two matrices.

We hence define:

$$\bar{Q} := \begin{pmatrix} q_0 & q^\top \\ 0_n & \frac{1}{2}Q \end{pmatrix} \tag{2.34}$$

$$\bar{A}_i := \begin{pmatrix} 0 & a_i^\top \\ 0_n & \frac{1}{2}A_i. \end{pmatrix}, \quad \forall i = 1, \ldots, m. \tag{2.35}$$

We also consider $b_i = -\bar{a}_i$. With this notation, the problem (2.33) can be rewritten as:

$$\min \langle \bar{Q}, X \rangle \tag{2.36}$$

$$\begin{aligned} \text{s. t.} \quad & \langle \bar{A}_i, X \rangle \leq b_i, \quad \forall i = 1, \ldots, m \\ & l_i \leq x_i \leq u_i \quad \forall i = 1, \ldots, n \\ & X = \begin{pmatrix} 1 & x^\top \\ x & xx^\top \end{pmatrix} \\ & x_i \in \mathbb{Z} \quad \forall i \in I \subset \{1 \ldots, n\}. \end{aligned} \tag{2.37}$$

**Conic relaxations**

Since the constraint (2.37) is non convex and it is difficult to deal with, a standard technique is relaxing this constraint, obtain lower bounds and then proceed with Branch and bound techniques. The most typical relaxation is the *SDP relaxation*: this constraint is replaced by the following:

$$X - \begin{pmatrix} 1 & x^\top \\ x & xx^\top \end{pmatrix} \succeq 0, \tag{2.38}$$

that is, this matrix must be PSD, or lie in the cone of positive semidefinite matrices. SemiDefinite Programming (SDP) is particularly interesting because it is polynomial, so a solution to the relaxation is relatively easy to obtain. Specific software has been developed for SDP, for example the commercial solver *MOSEK*, SeDuMi (see [128]), SDPT3 (see [132]), CSDP (see [22]) among others. Other types of conic relaxation exist. Two important classes are copositive and completely positive relaxations, that is optimization in the cone of *Copositive matrices* or *Completely positive matrices*, respectively. Results for these particular cones will be treated in detail in the following section.

## 2.12 Copositive optimization

The subject of copositive programming has firstly been introduced in [91]. Then, it has received increasing interest over the last few decades and a large number of works has been produced. For a description of copositive optimization, we start referring to [30] and to the survey [59]. Other papers which present reviews of recent advances in this field, along with applications, are [16] and [20]. The book [6] is also a useful source.

A *copositive* program is a linear conic optimization problem on the copositive cone, which can be written without loss of generality as:

$$\begin{aligned} &\min \ \langle C, X\rangle \\ &\text{s. t.} \ \langle A_i, X\rangle = b_i, \quad i = 1, \dots, m \\ &\qquad X \in \mathcal{C}^n. \end{aligned} \tag{2.39}$$

Its dual, as seen in Section 2.4, is a *completely positive* program. Since formulations (2.5) and (2.6) are equivalent, here we can consider the following as a completely positive problem:

$$\begin{aligned} &\min \ \langle C, X\rangle \\ &\text{s. t.} \ \langle A_i, X\rangle = b_i, \quad i = 1, \dots, m \\ &\qquad X \in \mathcal{C}^{*n}. \end{aligned} \tag{2.40}$$

In the literature copositive programming is used in reference to one of these two formulations, due to their strong connections.

The first important result about copositive programming is given by Bomze et al. in [18]: the authors consider the so-called *Standard quadratic problem*, which is the minimization of a quadratic function over a simplex:

$$\begin{aligned} &\min \ x^\top Q x \\ &\text{s. t.} \ e^\top x = 1 \\ &\qquad x \geq 0. \end{aligned} \tag{2.41}$$

They prove that its completely positive relaxation:

$$\begin{aligned} &\min \ \langle Q, X\rangle \\ &\text{s. t.} \ \langle E, X\rangle = 1 \\ &\qquad X \in \mathcal{C}^{*n} \end{aligned} \tag{2.42}$$

is a reformulation of the original problem. However, Standard quadratic program is NP-hard, since the maximum clique problem can be reduced to it. This is an example of a convex NP-hard problem.

This first reformulation of a quadratic program in copositive form is followed by other results. A noticeable result is given by Burer in [28]. He extended the first result to every linearly constrained quadratic problem with binary variables: he proved that the problem in variables $x \in \mathbb{R}^n$:

$$\min \quad \frac{1}{2} x^\top Q x + q^\top x \tag{2.43a}$$

$$\text{s. t.} \quad a_i^\top x = b_i, \qquad \forall i = 1 \dots, m \tag{2.43b}$$

$$x \geq 0 \tag{2.43c}$$

$$x_j \in \{0, 1\} \qquad \forall j \in B \subseteq \{1, \dots, n\}. \tag{2.43d}$$

is equivalent to the following completely positive relaxation:

$$\min \langle \frac{1}{2}Q, X\rangle + q^\top x \tag{2.44a}$$

$$\text{s. t.} \quad a_i^\top x = b_i, \qquad \forall i = 1\ldots, m \tag{2.44b}$$

$$a_i^\top X a_i = b_i^2, \qquad \forall i = 1\ldots, m \tag{2.44c}$$

$$x_j = X_{jj} \qquad \forall j \in B \tag{2.44d}$$

$$\begin{pmatrix} 1 & x^\top \\ x & X \end{pmatrix} \in \mathcal{C}^{*n+1}. \tag{2.44e}$$

under the so-called key-assumption, that is, $a_i^\top x = b_i$ for all $i$ and $x \geq 0$ imply $x_j \leq 1$ $\forall j \in B$. But this condition, as already noticed, is easily achieved without loss of generality, even when it is not already implied. His result extends to problems which have special quadratic constraints: the complementarity constraints.

Other equivalences between NP-hard problems and completely positive relaxations are obtained for the maximum stable set problem (see [45]), the chromatic number (see [89]), the quadratic assignment problem (see [121]), and also problems with uncertainty on the data (see [110]).

Solving a copositive problem remains obviously very hard. Typically, hierarchies of polyhedral cones are used to approximate the copositive cone. These are used, for instance, in [17] and in [27], while in [29] the author proposes a doubly non negative relaxation of the completely positive cone. A different approach is given by Bomze et al. in [19], where the authors propose a feasible direction heuristic to solve a problem in the completely positive cone. However, the initial point with a factorization is needed, which is trivial in some cases, but difficult in general.

Geometrical aspects of these cones have been extensively studied. The interior of the completely positive cone has been described in [60] and their result has been improved by Dickinson in [53]. Other important results are obtained by Dickinson in [54]. There are still some open questions, which are reported in [8].

# Part I

# Continuous and mixed binary convex QPs

# Chapter 3

# A conjugate direction based Simplicial Decomposition framework for solving a specific class of dense convex quadratic programs

In this chapter, we tackle convex problems with the following form:

$$\begin{aligned} \min\ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.}\ & Ax \geq b, \\ & Cx = d, \\ & l \leq x \leq u, \end{aligned} \tag{3.1}$$

with $Q \in \mathbb{R}^{n\times n}$, $c, l, u \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1\times n}$, $b \in \mathbb{R}^{m_1}$ , $C \in \mathbb{R}^{m_2\times n}$, $d \in \mathbb{R}^{m_2}$, $n, m_1, m_2 \in \mathbb{N}$.

We assume that the domain

$$X = \{x \in \mathbb{R}^n :\ Ax\ \geq\ b,\ \ Cx\ =\ d,\ l \leq\ x \leq u\}$$

is non empty and bounded and the quadratic matrix $Q$ is positive semidefinite. Furthermore, among all possible problems of type (3.1), we are particularly interested in those with the following additional properties:

- the Hessian matrix $Q$ is dense;
- an optimal solution can be obtained as a proper convex combination of a small subset of vertices in the original feasible set $X$;
- there exists an efficient method for minimizing a linear function over the feasible set $X$, i.e., there exists an efficient linear minimization oracle that for a given $y \in \mathbb{R}^n$ solves the problem:

$$\min_{x\in X} y^\top x.$$

We develop a *simplicial decomposition type* approach (see, e.g., [118,140]), that follows the algorithm described in section 2.8.

Simplicial decomposition is related to cutting plane/column generation approaches (see, e.g., [11, 118] for further details). Those methods are well suited to solve large scale structured convex programs, since they can efficiently exploit the structure in the problem. Many different cutting plane approaches have been proposed in the literature, including: *bundle method* [64, 65, 101, 133, 134], *center of gravity method* [104, 113], *maximum volume ellipsoid cutting plane method* [130], *Chebyshev center cutting plane method* [61] and *analytic center cutting plane method* [75, 76]. In the last two decades, some in depth computational studies related to the last class of methods, which seems to give a good trade-off between simplicity and practical performance, have been conducted (see, e.g., [74, 77, 83]). In some recent papers, a good number of structured problems has been analyzed from the column-generation (primal) perspective, and a Primal-Dual Column Generation Method (PDCGM) has been developed (see, e.g., [79–81]). PDCGM has also been embedded into a Branch-Price-and-Cut algorithmic framework for solving problems with integer variables (see, e.g., [109]).

Despite the vast literature in the context of cutting plane/column generation approaches, no in-depth computational and methodological analysis has been conducted so far to investigate the behaviour of simplicial decomposition on a statistically significant set of instances. In this work, we hence show how a well designed simplicial decomposition framework can efficiently handle convex quadratic instances satisfying the properties mentioned above. To obtain good performances, two important features are considered:

- a new ad-hoc method for solving the master problem, called Adaptive Conjugate Direction Method;
- some pricing strategies that help speeding up the solution process.

In particular, the new master solver represents the first attempt to embed and wisely reuse conjugate directions into a simplicial decomposition framework. We will describe in depth the idea behind the algorithm and show how it works in practice. We then analyze the connections between some pricing strategies we introduce to improve the efficiency of our approach and classic features/ideas in cutting plane approaches. We also show how those strategies can be embedded into the algorithmic framework without affecting its convergence. Finally, we perform numerical experiments on a benchmark of almost 1400 instances. We start our analysis by testing our method over a suitably chosen set of QPLIB instances [71]. This first part will show how the method performs when density of the Hessian changes. Then we focus on structured problems satisfying the properties described above. We consider:

- problems coming from Computational Geometry (the Chebishev problem, see, e.g., [40] and references therein), Machine Learning/Statistics (the LASSO problem, see, e.g., [131]) and Economics (classic Markowitz Portfolio Optimization problem [106]);

- problems with a combinatorial structure, that is continuous relaxations of instances related to the Quadratic Shortest Path problem [55, 72, 129] and Quadratic Multidimensional Knapsack Problem [24, 125, 126].

We further consider a randomly generated set of quadratic programs with dense objective function and a small number of constraints, and test the effectiveness of the different master/pricing strategies over this benchmark.

It is important to notice that, in principle, the proposed method can handle any problem of type (3.1) and can also be easily modified in order to deal with problems having a general convex objective function. However, the numerical experience we performed indicates that our approach really makes a difference when the problem has the structure mentioned above. In order to better understand the reasons why this happens, we need to take into account the way the method works. A generic iteration is characterized by two different steps:

- **master problem minimization:** minimization of the original function over an inner approximation of the original feasible set $X$ (given by the convex hull of an ever expanding finite subset of its vertices);

- **subproblem minimization:** minimization of a suitable linear approximation of the objective function over the original feasible set $X$.

When using a set of barycentric coordinates to re-parameterize the master, we obtain a nicely structured quadratic program (its feasible set is the unit simplex) that is efficiently handled by our conjugate direction method. Furthermore, since the optimal solution of the considered problems can be described as a sparse convex combination of the vertices in $X$, our framework normally takes a few iterations to get a good solution. Hence, it is easy to see that a dense Hessian matrix $Q$, which is hard to handle for other methods, is not an issue in our case (we usually have a small submatrix of the original Hessian $Q$ in the master). This feature, combined with a fast linear minimization oracle for the subproblem, guarantees good performances in the end.

The rest of the chapter is organized as follows. In Section 3.1 and 3.2, we present some strategies to improve the efficiency of the framework itself, in the master and in the pricing problems of the Simplicial Decomposition. In Section 3.3, we report our numerical experience. Finally, in Section 3.4, we draw some conclusions.

This work has been accepted for publication in the journal *Computational Optimization and Applications*. A preliminary version is available at [12].

## 3.1 Master program

We start with noticing the following useful property of the master problem for quadratic optimization.

**Master formulation for Quadratic Problems** If the original problem is quadratic, the master can be expressed in a simplified form. We write explicitly the master program for a quadratic form, at a generic iteration $k$:

$$\begin{aligned} \min \quad & x^\top Q x + c^\top x && (3.2)\\ \text{s. t.} \quad & x = \sum_{i=1}^{k} \lambda_i \tilde{x}_i, && (3.3)\\ & \sum_{i=1}^{k} \lambda_i = 1, \\ & \lambda_i \geq 0, \quad \forall i = 1, \ldots, k. \end{aligned}$$

with $Q \in \mathbb{R}^{n \times n}$, $c \in \mathbb{R}^n$.

It is possible to substitute constraints 3.3 in the objective function. We define

$$B := \begin{pmatrix} \tilde{x}_1 & \ldots & \tilde{x}_k \end{pmatrix} \tag{3.4}$$

the $n \times k$ matrix whose columns are the extreme points of $X_k$. Then, the constraints 3.3 can be rewritten as:

$$x = B\lambda, \tag{3.5}$$

with $\lambda \in \mathbb{R}^k$ the vector of the coefficients in the convex combination.

Then, if we define the following matrices:

$$\tilde{Q} := B^\top Q B, \quad \tilde{c} := B^\top c, \tag{3.6}$$

by simple linear algebra, we can substitute the terms in the objective function and eliminate the $x$ variables, obtaining a problem with only the $\lambda$ variables:

$$\begin{aligned} \min \quad & \lambda^\top \tilde{Q} \lambda + \tilde{c}^\top \lambda && (3.7)\\ \text{s. t.} \quad & \sum_{i=1}^{k} \lambda_i = 1, \\ & \lambda_i \geq 0, \quad \forall i = 1, \ldots, k. \end{aligned}$$

that is the *standard* quadratic problem in only $k \ll n$ variables and it is much more tractable.

*Remark* 3.1. Not for all the convex functions the original variables can be eliminated: in general, the master remains a problem with $n + k$ non separable variables and $n$ constraints, and potentially with quadratic terms in the objective function for all the original $n$ variables.

We now describe two strategies for solving the master problem that we have developed. The first one is an adaptation to our framework of the unconstrained method of the Conjugate Directions. It has been introduced to exploit the properties of the simplices and the information gathered from previous iterations (i.e., to reuse conjugate directions). To the best of our knowledge, this is the first time that such an algorithm is introduced and analyzed. The second one is an adaptation of the Spectral Projected Gradient Method presented in [15].

### 3.1.1 An adaptive conjugate directions based method (ACDM) for solving the master

We described the conjugate directions method in Section 2.6. Before describing the details related to the first method, we report a result, which is a consequence of the properties of the conjugate directions, which will be useful to better understand our algorithm.

**Proposition 3.1.** *Conjugate direction method converges to the minimum point of a strictly convex quadratic function $f(x) : \mathbb{R}^n \to \mathbb{R}$ in at most $n$ steps.*

We now describe a new conjugate direction strategy for solving the master problem in our simplicial decomposition framework.
We would like to notice that dealing with (linear) constraints in conjugate direction or conjugate gradient methods usually represents a complicated issue in practice. In our case, by exploiting some master problem features, we basically avoid the explicit handling of those constraints, and have a strategy that easily embeds the use of conjugate directions in the end. What we hence do at each iteration of the method is trying to suitably generate conjugate directions by taking into account the structure of the feasible set related to the master problem and use those directions to find its optimal solution. Since the master solution is a proper combination of some vertices in the inner approximation of the original feasible set, we can have two different cases: if the solution lies in the interior of the master feasible set, our conjugate direction strategy works just fine and we get our solution; otherwise one of the generated directions will give a point on the boundary of the feasible set, and this will get us a reduced simplex that we use to restart the procedure. Below we analyze in depth the way our algorithm works. We remark that $k$ represents the index of the iteration related to the simplicial decomposition framework.

In order to ease the description of our algorithm, in this first part of the section we assume that the objective function in (3.3) is strictly convex. Later we will prove that the result holds even for semidefinite matrices.

The result reported in Proposition 3.1 relies on the fact that the objective function is successively minimized along the individual directions in a conjugate set. This fact is heavily exploited when solving the master problem with ACDM. First of all, we assume that

- at each iteration the solution we get when solving the master is in the interior of the simplex approximating the feasible set (if not, we can just remove from master problem those components that have zero weight in the solution);
- a suitable set of conjugate directions is available in the affine hull described by those vertices.

Hence, if the master solution at iteration $k$ is in the interior of the simplex, ACDM easily calculates it by picking the master solution obtained at the previous iteration $k-1$ and applying a minimization step along a new suitably chosen conjugate direction (keep in mind that the pricing phase only adds one dimension at each iteration). In case the master

solution is on the boundary of the simplex, once the method applies the minimization along the new conjugate direction, it hits the boundary, reduces the variable space and needs to get a new set of conjugate directions. Luckily, this case does not happen so often in practice, thus making ACDM a viable option. In the rest of this section, we formally describe all the steps we need to implement the method.

Let $\Delta_k := conv(X_k)$ be the domain of the current master, $\Delta_{k-1}$ be the domain of the previous master, $x_{k-1}$ the optimum of the previous master and $\tilde{x}_k \notin \Delta_{k-1}$ the new extreme point generated with the pricing. At any SD iteration, the master problem we want to solve (Step 1 of Algorithm 1) has the form in (3.7) and the following property holds:

**Proposition 3.2.** *The master solution at iteration $k-1$ lies in the relative interior of a facet of the set $\Delta_k$.*

As we already noticed, if the master solution we get at iteration $k-1$ is on the boundary of the simplex, we can restrict the master problem to a smaller dimensional face (which is always a simplex) by means of the *column dropping rule*. Furthermore, the simplex of the current master is obtained by adding up the point provided by the pricing to the set of vertices describing that reduced face.

We now consider the descent direction $\bar{d}_{k-1} := \tilde{x}_{k-1} - x_{k-1}$. At the first iteration, that is $k = 2$, we keep $\bar{d}_1$ as the first direction $d_1$. If $k > 2$, we assume that a set of conjugate directions $D = \{d_1, \ldots, d_{k-2}\}$ is available from previous iterations. We then use a Gram-Schmidt like procedure to turn direction $\bar{d}_{k-1}$ into a new direction $d_{k-1}$ conjugate with respect to the set $D$. More specifically, we compute:

$$d_{k-1} = \bar{d}_{k-1} - \sum_{h=1}^{k-2} \delta_{k-1}^h d_h, \quad \text{where} \quad \delta_{k-1}^h = \frac{\bar{d}_{k-1}^\top Q d_h}{d_h^\top Q d_h}. \tag{3.8}$$

It is worth noticing that $d_{k-1}$ is a descent direction too (since $x_{k-1}$ is optimal, $\nabla f(x_{k-1})^\top d_h = 0, \forall\ h = 1, \ldots, k-2$). We use the basis $B = [\tilde{x}_1, \ldots, \tilde{x}_k]$ to express points $x^s = x_{k-1}$ and $x^t = x_{k-1} + d_{k-1}$ thus obtaining respectively points $\lambda^s$ and $\lambda^t$. We intersect the halfline emanating from $x^s$ (and passing by $x^t$) with the boundary of $\Delta_k$ by solving the following problem:

$$\begin{array}{ll} \max & \alpha \\ \text{s.t.} & (1-\alpha)\lambda^s + \alpha\lambda^t \geq 0. \end{array} \tag{3.9}$$

The solution of problem (3.9) can be directly written as

$$\alpha^* = \left( \max_i \frac{\lambda_i^s - \lambda_i^t}{\lambda_i^s} \right)^{-1}.$$

We finally define point $\lambda^p = (1-\alpha^*)\lambda^s + \alpha^*\lambda^t$ and solve the following problem

$$\min_{\beta \in [0,1]} f(B[(1-\beta)\lambda^s + \beta\lambda^p]). \tag{3.10}$$

If the optimal value $\beta^* < 1$ we get, by Proposition 3.1, an optimal solution for the master. Otherwise, $\beta^* = 1$ and we are on the boundary of the simplex. In this case, we just drop those vertices whose associated coordinates are equal to zero, and get a new smaller basis $B$. If $B$ is a singleton, we can stop our procedure, otherwise we minimize $f(x)$ in the new subspace defined by $B$. In order to get a new set of conjugate directions in the considered subspace, we use directions connecting point $x^* = B\lambda^* = B\lambda^p$ with each vertex $\tilde{x}_j$ in $B$ (that is $\bar{d}_j = \tilde{x}_j - x^p$) and then use the same Gram-Schmidt like procedure to make them conjugate (we want to remark that all directions $\bar{d}_j$ need to be expressed in terms of the new basis $B$). We report the algorithmic scheme below (see Algorithm 2). In order to ease the notation, we consider $|X_0| = 1$ in our simplicial decomposition framework. We thus do not perform any optimization at the first iteration of the framework (i.e., when $k = 1$), and we simply set $x_1$ equal to the only point that is in $X_0$.

We now see that the procedure still works when we assume that $f$ in Problem (3.1) is a quadratic convex functions (i.e., $Q$ is positive semidefinite). We have that there could exist some directions $d_h$ satisfying the following condition:

$$d_h^\top Q d_h = 0, \tag{3.11}$$

i.e., $d_h$ is degenerate, or in the null space of $Q$. Let $\bar{h}$ be the smallest index related to a direction satisfying condition (3.11). We can consider two different cases:

1. $\bar{h} = 1$) We then have $d_{\bar{h}} = d_1$ and the solution of (3.10), which is now a linear problem, is $\beta^* = 1$: indeed, the function is decreasing (keep in mind that $d_{k-1}$ in (3.8) is always a descent direction). So, we get a vertex ($B$ is hence a singleton), and the algorithm stops.

2. $\bar{h} > 1$) similarly to Case 1, we get that problem (3.10) is again linear and its solution is $\beta^* = 1$. The algorithm thus restricts to the face of the boundary that contains the intersection with the half-line defined by $d_{\bar{h}}$. If we get a vertex, then the algorithm stops; otherwise, the algorithm is still well-defined, because the intersection face does not contain any other degenerate direction and it can go on.

The finite convergence of this algorithm is clearly guaranteed:

**Proposition 3.3.** *An SD scheme which embeds Algorithm 2 for solving the master has finite convergence.*

*Proof.* The proof is based on the same arguments as in [140]: at each iteration the master problem finds the optimum in the simplex or in one of its faces, which are simplices as well. Moreover, even if the matrix is positive semidefinite, we always move along descent directions so the value of the objective function always strictly decreases. Since at each iteration we get a new simplex and the number of simplices is finite, we have that the number of iterations must be finite as well. □

**Algorithm 2** Adaptive Conjugate Direction Method (ACDM)

**Data:** Basis $B$, conjugate directions $D$, and point $x_{k-1}$

**Step 1)** Set $x^s = x_{k-1}$ and $D^s = \{\bar{d}_{k-1}\}$

**Step 2)** Select a $\bar{d} \in D^s$ and set $D^s = D^s \setminus \{\bar{d}\}$

**Step 3)** Use a Gram-Schmidt like procedure to turn $\bar{d}$ into a conjugate direction $d^s$ with respect to $D$

**Step 4)** Express points $x^s$ and $x^t = x^s + d^s$ in terms of $B$ (that is $x^s = B\lambda^s$ and $x^t = B\lambda^t$)

**Step 5)** Set

$$\alpha^* = \left( \max_i \frac{\lambda_i^s - \lambda_i^t}{\lambda_i^s} \right)^{-1}$$

**Step 6)** Calculate point $\lambda^p = (1 - \alpha^*)\lambda^s + \alpha^* \lambda^t$ and find solution $\beta^*$ of the problem

$$\min_{\beta \in [0,1]} f(B[(1-\beta)\lambda^s + \beta\lambda^p])$$

**Step 7)** If $\beta^* < 1$ then set $x^* = B[(1-\beta^*)\lambda^s + \beta^*\lambda^p]$ and $D = D \cup \{d^s\}$ go to Step 9
Else drop vertices with $\lambda^* = 0$ from $B$

**Step 8)** If $B$ is a singleton then STOP
Else set $D = \emptyset$ and for each $\tilde{x}_j \in B$ set $\bar{d}_j = \tilde{x}_j - x^*$ (direction represented using coordinates in $B$) to get a set of directions $D^s$ and go to Step 2

**Step 9)** If $D^s = \emptyset$ then STOP
Else go to Step 2

### 3.1.2 A fast gradient projection method for solving the master

The second approach is a Fast Gradient Projection Method (FGPM) and belongs to the family of gradient projection approaches (see e.g. [15] for an overview of gradient projection approaches). The detailed scheme is reported below (See Algorithm 3). We indicate with $l$ the index related to the iterations of FGPM. At each iteration of the method, the new point we generate is

$$\lambda_{l+1} = \lambda_l + \beta_l(p[\lambda_l - s_l \nabla f(\lambda_l)]_\Delta - \lambda_l),$$

where $\beta_l \in (0, \rho_l]$, $\rho_l, s_l > 0$ and $p[\lambda_l - s_l \nabla f(\lambda_l)]_\Delta$ is the projection over the master simplex $\Delta_l$ of the point $\lambda_l - s_l \nabla f(\lambda_l)$, chosen along the antigradient. When $p[\lambda_l - s_l \nabla f(\lambda_l)]_\Delta \neq \lambda_l$, it is easy to see that the direction we get is a feasible descent direction.
The method can be used in two different ways:

a) we fix $s_l$ to a constant value and use a line search technique to get $\beta_l$;

b) we fix $\beta_l$ and make a search changing $s_l$ (thus getting a curvilinear path in the feasible set).

In our algorithm we consider case *a*) where $s_l = s > 0$.

At each iteration, projecting the point $y_l = \lambda_l - s\nabla f(\lambda_l)$ over the simplex corresponds to solve the following problem:

$$\min_{x \in \Delta} \|x - y\|_2.$$

**Algorithm 3** Fast Gradient Projection Method (FGPM)

**Data:** Set point $\lambda_0 \in \mathbb{R}^{l-1}$, $\rho_0 \in [\rho_{min}, \rho_{max}]$ and a scalar value $s > 0$.

**For** $l = 0, 1, \ldots$

**Step 1)** Generate point

$$\hat{\lambda}_l = p[\lambda_l - s\nabla f(\lambda_l)]_\Delta$$

**Step 2)** If $\hat{\lambda}_l = \lambda_l$ STOP; otherwise set $d_l = \hat{\lambda}_l - \lambda_l$

**Step 3)** Choose a stepsize $\beta_l \in (0, \rho_l]$ along $d_l$ and maximum stepsize $\rho_{l+1}$ by means of a line search

**Step 4)** Set $\lambda_{l+1} = \lambda_l + \beta_l d_l$

**End For**

A fast projection over the simplex is used to generate the search direction [38]. This particular way of projecting a point over the simplex is basically a Gauss-Seidel-like variant of Michelot's variable fixing algorithm [108]. The threshold used to fix the variables is updated after each element is read, instead of waiting for a full reading pass over the list of non-fixed elements (See [38] for further details).

A nonmonotone line search [86] combined with a spectral steplength choice is then used at Step 3 (see [15] for further details) to speed up convergence. In Algorithm 4 we report the detailed scheme of the line search. Convergence of the FPGM algorithm to a minimum follows from the theoretical results in [15]. Therefore, the convergence of an SD method that uses FPGM to solve the master problem directly follows from the results in the previous sections.

**Algorithm 4** Non-monotone Armijo line-search (with spectral steplength choice)

0 Set $\delta \in (0, 1)$, $\gamma_1 \in (0, \frac{1}{2})$, $M > 0$

1 Update

$$\bar{f}_l = \max_{0 \le i \le \min\{M,l\}} f(\lambda_{l-i})$$

2 Set starting stepsize $\alpha = \rho_l$ and set $j = 0$

3 **While** $f(\lambda_l + \alpha d_l) > \bar{f}_l + \gamma_1 \alpha \nabla f(\lambda_l)^\top d_l$

4 set $j = j + 1$ and $\alpha = \delta^j \alpha$.

5 **End While**

6 Set $y_l = \nabla f(\lambda_l + \alpha d_l) - \nabla f(\lambda_l)$ and $b_l = \alpha d_l^\top y_l$

7 If $b_l \le 0$ set $\rho_{l+1} = \rho_{max}$ else set $a_l = \alpha^2 \|d_l\|^2$ and

$$\rho_{l+1} = min\{\rho_{max}, \max\{\rho_{min}, a_l/b_l\}\}$$

In the FGPM Algorithm, we exploit the particular structure of the feasible set in the master, thus getting a very fast algorithm in the end. We will show that the FGPM based SD framework is even competitive with the ACDM based one, when dealing with some specific quadratic instances. Moreover, differently from ACDM, FGPM is easily adapted to solve any convex nonlinear problems, not only quadratic problems. Adapting ACDM to solve more general problems is possible, but it is much more complicated, and it would

loose the fast convergence properties, which hold only in the quadratic case.

## 3.2 Pricing program

Now we describe two different strategies for speeding up the solution of the pricing problem (also called subproblem). The first one is an *early stopping* strategy that allows us to approximately solve the subproblem while guaranteeing finite convergence. The second one is the use of suitably generated inequalities (the so called *shrinking cuts*) that both cut away a part of the feasible set and enable us to improve the quality of extreme points picked in the pricing phase.

### 3.2.1 Early stopping strategy for the pricing

When we want to solve problem (2.26) using simplicial decomposition, efficiently handling the subproblem is, in some cases, crucial. Indeed, the total number of extreme points needed to build up the final solution can be small for some real-world problems, hence the total time spent to solve the master problems is negligible when compared to the total time needed to solve subproblems. This is the reason why we may want to approximately solve subproblem (2.30) in such a way that finite convergence is guaranteed (a similar idea was also suggested in [10]). In order to do that, we simply need to generate an extreme point $\tilde{x}_k$ satisfying the following condition:

$$\nabla f(x_k)^\top (\tilde{x}_k - x_k) \leq -\varepsilon < 0, \tag{3.12}$$

with $\varepsilon > 0$. Roughly speaking, we want to be sure that, at each iteration $k$, $d_k = \tilde{x}_k - x_k$ is a descent direction. Below, we report the detailed scheme related to the simplicial decomposition algorithm with early stopping (see Algorithm 5).

**Algorithm 5** Simplicial Decomposition with Early Stopping Strategy for the Subproblem

**Initialization:** Choose a starting set of extreme points $X_0$

**For** $k = 0, 1, \ldots$

**Step 1)** Generate iterate $x_k$ by solving the **master problem**

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in conv(X_k) \end{array}$$

**Step 2)** Generate an extreme point $\tilde{x}_k \in X$ such that

$$\nabla f(x_k)^\top (\tilde{x}_k - x_k) \leq -\varepsilon < 0.$$

In case this is not possible, pick $\tilde{x}_k$ as the optimal solution of (2.30)

**Step 3)** If $\nabla f(x_k)^\top (\tilde{x} - x_k) \geq 0$, **Stop**. Otherwise set $X_{k+1} = X_k \cup \{\tilde{x}_k\}$

**End For**

At a generic iteration $k$ we generate an extreme point $\tilde{x}_k$ by approximately solving the linear program (2.30). This is done in practice by stopping the algorithm used to

solve problem (2.30) as soon as a solution satisfying constraint (3.12) is found. In case no solution satisfies the constraint, we simply pick the optimal solution of (2.30) as the new vertex to be included in the simplex at the next iteration.

Finite convergence of the method can be proved in this case as well:

**Proposition 3.4.** *Simplicial decomposition with early stopping strategy for the subproblem obtains a solution of Problem* (2.26) *in a finite number of iterations.*

*Proof.* Extreme point $\tilde{x}_k$, obtained approximately solving subproblem (2.30), can only satisfy one of the following conditions

1. $\nabla f(x_k)^\top(\tilde{x}_k - x_k) \geq 0$, and subproblem (2.30) is solved to optimality. Hence we get

$$\min_{x \in X} \nabla f(x_k)^\top(x - x_k) = \nabla f(x_k)^\top(\tilde{x}_k - x_k) \geq 0,$$

   that is necessary and sufficient optimality conditions are satisfied and $x_k$ minimizes $f$ over the feasible set $X$;

2. $\nabla f(x_k)^\top(\tilde{x}_k - x_k) < 0$, whether the pricing problem is solved to optimality or not, that is direction $d_k = \tilde{x}_k - x_k$ is descent direction and

$$\tilde{x}_k \notin conv(X_k). \tag{3.13}$$

   Indeed, since $x_k$ minimizes $f$ over $conv(X_k)$ it satisfies necessary and sufficient optimality conditions, that is $\nabla f(x_k)^\top(x - x_k) \geq 0$ for all $x \in conv(X_k)$.

From (3.13) we thus have $\tilde{x}_k \notin X_k$. Since our feasible set $X$ has a finite number of extreme points, case 2) occurs only a finite number of times, and case 1) will eventually occur. □

We would like to highlight that the early stopping strategy is somehow related to the use of $\varepsilon$-subgradients in nonsmooth optimization (see [98] and references therein for further details) and to recent techniques used for bundle methods (see [133, 134]). When using decomposition schemes like, e.g., cutting plane schemes, it is indeed possible to weaken the optimality requirements in subproblems and get $\varepsilon$-subgradients, obtaining usually an improvement in terms of the overall computing time of the algorithm (see, e.g., [82, 84]). Anyway, if a shallow cut (i.e., a cut that does not exclude the current query point) is generated, the convergence of a cutting plane approach might fail. Hence, a check is needed in order to ensure that the cut is deep enough (i.e., shallow cut is discarded, $\varepsilon$ is suitably reduced and a new hopefully better cut is generated). In practice, as the cutting plane algorithm approaches the solution, the accuracy level with which the subproblem is solved should increase. In our simplicial decomposition framework, generating a good column (by approximately solving the pricing) while guaranteeing convergence is in general an easier task. Indeed, by taking a look at the proof of Proposition 3.4, we can notice that, in order to guarantee convergence, we only need the new column $\tilde{x}_k$ to be an extreme point of $X$ (actually might be enough getting a point $\tilde{x}_k$ from a finite subset

$\tilde{X} \subset X$ s.t. $conv(\tilde{X}) = X$) and to satisfy $\nabla f(x_k)^\top (\tilde{x}_k - x_k) < 0$. Thus, we can generate new columns for the master problem in a simple and natural way, without the need to check if those columns are nearly optimal (we only need to guarantee that the objective function can improve with respect to $f(x_k)$ in the new extended master).

### 3.2.2 Shrinking cuts

At each iteration $k$, after solving the master problem, we can consider the inequality:

$$\nabla f(x_k)^\top (x - x_k) \leq 0. \tag{3.14}$$

Since the objective function values of the subsequent iterates $x_{k+1}, x_{k+2}, \ldots$, generated by the method will be not greater than the objective function value obtained in $x_k$, inequality (3.14) will then be surely satisfied by all those subsequent iterates.

This can be easily seen by taking into account convexity of $f$. Indeed, choosing two points $x, y \in \mathbb{R}^n$, we have:

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x).$$

Thus, if $\nabla f(x)^\top (y-x) > 0$, we get $f(y) > f(x)$. Hence, $f(y) \leq f(x)$ implies $\nabla f(x)^\top (y - x) \leq 0$.

As also briefly discussed in [10], it is possible to use the cuts described above to suitably modify the subproblem. More specifically, the idea is the following: let $x_k$ be the optimal point generated by the master at a generic iteration $k$, we can hence add the following *shrinking cut* $c_k$ to the next pricing problems:

$$(c_k) \qquad \nabla f(x_k)^\top (x - x_k) \leq 0.$$

More precisely, let $\{x_1, \ldots, x_k\}$ be the set of optimal points generated by the master problems up to iteration $k$; then, for $k > 0$, we identify as $C_k$ the polyhedron defined by all the associated shrinking cuts as follows:

$$C_k = \{x \in \mathbb{R}^n : \ \nabla f(x_i)^\top (x - x_i) \leq 0, \ i = 0, \ldots, k-1\}.$$

(We are assuming $x_0 := \tilde{x}_0$). Therefore, at Step 2, we generate an extreme point $\tilde{x}_k$ by minimizing the linear function $\nabla f(x_k)^\top (x - x_k)$ over the polyhedral set $X \cap C_k$. Finally, at Step 3, if $\nabla f(x_k)^\top (\tilde{x} - x_k) \geq 0$, the algorithm stops, otherwise we update $X_k$ by adding the point $\tilde{x}_k$ and $C_k$ by adding the cut $\nabla f(x_k)^\top (x - x_k) \leq 0$.

Below, we report the detailed scheme related to the simplicial decomposition algorithm with shrinking cuts (see Algorithm 6).

In practice, we implemented the algorithm with the two following variants:

- At the end of Step 2, after the solution of the pricing problem, we remove all shrinking cuts that are not active. In this way we are sure to have a pricing problem that is computationally tractable by keeping its size under control.

**Algorithm 6** Simplicial Decomposition with Shrinking Cuts

**Initialization:** Choose a starting set of extreme points $X_0$

**For** $k = 0, 1, \ldots$

**Step 1)** Generate iterate $x_k$ by solving the **master problem**

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in conv(X_k) \end{array}$$

**Step 2)** Generate an extreme point $\tilde{x}_k$ by solving the **subproblem**

$$\begin{array}{ll} \min & \nabla f(x_k)^\top (x - x_k) \\ \text{s.t.} & x \in X \cap C_k \end{array} \tag{3.15}$$

**Step 3)** If $\nabla f(x_k)^\top (\tilde{x} - x_k) \geq 0$, **Stop**. Otherwise set $X_{k+1} = X_k \cup \{\tilde{x}_k\}$ and set $C_{k+1} = \{x \in \mathbb{R}^n : \nabla f(x_i)^\top (x - x_i) \leq 0,\ i = 0, \ldots, k\}$

**End For**

- After a considerably large number of iterations $\bar{k}$, no more shrinking cuts are added to the pricing. This is done to ensure the convergence of the Algorithm.

Finite convergence of the method is stated in the following Proposition:

**Proposition 3.5.** *Simplicial decomposition algorithm with shrinking cuts obtains a solution of Problem* (2.26) *in a finite number of iterations.*

*Proof.* We first show that at each iteration the method gets a reduction of $f$ when suitable conditions are satisfied. Since at Step 2 we get an extreme point $\tilde{x}_k$ by solving subproblem (3.15), if $\nabla f(x_k)^\top (\tilde{x}_k - x_k) < 0$, we have that $d_k = \tilde{x}_k - x_k$ is a descent direction and there exists an $\alpha_k \in (0, 1]$ such that $f(x_k + \alpha_k d_k) < f(x_k)$. Since at iteration $k+1$, when solving the master problem, we minimize $f$ over the set $conv(X_{k+1})$ (including both $x_k$ and $\tilde{x}_k$), then the minimizer $x_{k+1}$ must be such that

$$f(x_{k+1}) \leq f(x_k + \alpha_k d_k) < f(x_k).$$

Extreme point $\tilde{x}_k$, obtained solving subproblem (3.15), can only satisfy one of the following conditions

1. $\nabla f(x_k)^\top (\tilde{x}_k - x_k) \geq 0$. Hence we get

$$\min_{x \in X \cap C_k} \nabla f(x_k)^\top (x - x_k) = \nabla f(x_k)^\top (\tilde{x}_k - x_k) \geq 0,$$

that is necessary and sufficient optimality conditions are satisfied and $x_k$ minimizes $f$ over the feasible set $X \cap C_k$. Furthermore, if $x \in X \setminus C_k$, we get that there exists a cut $c_i$ with $i \in \{0, \ldots, k-1\}$ such that

$$\nabla f(x_i)^\top (x - x_i) > 0.$$

Then, by convexity of $f$, we get

$$f(x) \geq f(x_i) + \nabla f(x_i)^\top (x - x_i) > f(x_i) > f(x_k)$$

so $x_k$ minimizes f over $X$.

2. $\nabla f(x_k)^\top (\tilde{x}_k - x_k) < 0$, that is direction $d_k = \tilde{x}_k - x_k$ is descent direction and

$$\tilde{x}_k \notin conv(X_k). \tag{3.16}$$

Indeed, since $x_k$ minimizes $f$ over $conv(X_k)$ it satisfies necessary and sufficient optimality conditions, that is we have $\nabla f(x_k)^\top (x - x_k) \geq 0$ for all $x \in conv(X_k)$.

Since from a certain iteration $\bar{k}$ on we do not add any further cut (notice that we can actually reduce cuts by removing the non-active ones), then case 2) occurs only a finite number of times. Thus case 1) will eventually occur. □

Adding the shrinking cuts guarantees that the direction we obtain by solving the subproblem is a descent direction with respect to $x_k$ (which is usually the case in a SD framework) and all points $x_i,\ i = 1, \ldots,\ k-1$ that we consider in the definition of $C_k$. This enables us to suitably cut away a part of the feasible set.

We can also see that shrinking cuts can be used to discard some of the points in $X_k$ generated so far. In order to better understand this fact, let us consider the master problem described using the bary-centric coordinates:

$$\begin{aligned} \min \quad & f(B\lambda) \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^{k} \lambda_i = 1 \\ & \lambda_i \geq 0 \quad \forall i = 1, \ldots, k, \end{aligned} \tag{3.17}$$

where $B := [\tilde{x}_1 \ldots \tilde{x}_k]$. Now, if we call $\bar{\lambda}$ the optimal solution of problem (3.17), and consider the KKT conditions related to this problem (keep in mind that $x_k = \sum_{i=1}^{k} \bar{\lambda}_i \tilde{x}_i$), we can write the following condition:

$$\begin{cases} \nabla f(B\bar{\lambda})^T B(e_i - \bar{\lambda}) = \nabla f(x_k)^T (\tilde{x}_i - x_k) \geq 0 \quad \forall\ i :\ \bar{\lambda}_i = 0, \\ \nabla f(B\bar{\lambda})^T B(e_i - \bar{\lambda}) = \nabla f(x_k)^T (\tilde{x}_i - x_k) = 0 \quad \forall\ i :\ \bar{\lambda}_i > 0. \end{cases}$$

We hence get that there is no point $\tilde{x}_i$ with $\bar{\lambda}_i \neq 0$ such that condition $\nabla f(x_k)^T (\tilde{x}_i - x_k) > 0$ is satisfied. The cuts can thus make a difference when considering a version of the Algorithm 6 that embeds the *vertex dropping rule* (which can be included at the end of Step 1). Indeed, if we consider the vertices $\tilde{x}_i$ we get rid of in the inner approximation, all those points have a $\bar{\lambda}_i = 0$. When those point also satisfy condition $\nabla f(x_k)^T (\tilde{x}_i - x_k) > 0$, we have that the subproblems including the shrinking cut related to $x_k$ in $C_k$ cannot generate again $\tilde{x}_i$ as a vertex in a subsequent iteration.

Shrinking cuts are connected to cutting plane approaches as well. In fact, those constraints are somehow related to classic objective cuts (see, e.g., [98]). The main difference in this case is that the cuts are not added directly to the original problem in order to cut away a part of the feasible set, but instead they are used in the pricing problem in order to shrink the original feasible set and hopefully generate better columns. An additional interesting feature of shrinking cuts is that, when solving the pricing problem, we always have a feasible point to warmstart the solver we use. It is indeed easy to notice that the master solution $x_k$ both satisfies the original constraints and the shrinking cuts in $C_k$. This comes from the fact that the objective function value $f(x_k)$ is lower or equal than the objective functions of the master solutions generated at previous iterations. On the other hand, guaranteeing feasibility while including objective cuts might be an issue in some cutting plane strategies.

As a final remark, we would like to notice that combining the shrinking cuts with the early stopping strategy can be done (this is a part of what we actually do in practice) and finite convergence still holds for the simplicial decomposition framework.

## 3.3 Computational results

### 3.3.1 Instances description

In this section, we give a detailed description of the computational results obtained with the SD based algorithmic framework we described in the previous sections.

For this reason, we introduce a benchmark that includes both generic and real quadratic programming problems. We use instances with a moderately large number of variables (up to 10000). This choice aims at making the problems hard enough, without anyway requiring specific techniques for storing the Hessian matrix related to the objective function.

Due to the specific features of the given problems, we use *Cplex* version 12.6.3 (see [97] for further details) as the baseline software in our tests. This tool includes several solvers for quadratic programming: primal simplex, dual simplex, network simplex, barrier, sifting and concurrent. They perform differently and some of them can efficiently handle the specific classes of problems we consider in the chapter. We point out that cutting plane based algorithms might not be the best choice in this case, since no structure can be exploited when solving the subproblem.

All tested algorithms were implemented in C++ using *Cplex* Callable Libraries. As we will see later on, one among sifting, network and the default optimizer from *Cplex* is used as solver for linear subproblems in our experiments. Furthermore, we decided to use a tolerance of $10E-6$ for FGPM.

In the first part of our analysis we test the ACDM based SD framework (without pricing strategies) on a set of instances from the QPLIB library [71]. The results show that performances of our algorithm improve as the number of nonzeroes in the Hessian matrix grows. Then, we focus on dense instances, and we present results for three classes of continuous problems, namely the Chebishev center problem, the LASSO problem

and the portfolio optimization problem. Furthermore, we provide results on continuous relaxations of two combinatorial problems: more specifically, quadratic shortest path problems and multidimensional knapsack problem. In the second part, we analyze the computational time of the different master and pricing settings described in the paper.

In our analysis, when considering specific problems and the extended benchmark, we produced the performance profiles according to [56] and using the software *Mathematica* version 11.3 (see [141] for further details).
We assume that we have a set $S$ of $n_s$ solvers and a set $P$ of $n_p$ problems. We consider time as a performance measure. For each problem $p$ and solver $s$, we call $t_{p,s}$ the computing time required to solve problem $p$ by solver $s$. We compare the performance on problem $p$ by solver $s$ with the best performance by any solver on this problem; that is, we use the performance ratio

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : \ s \in S\}}$$

When problem is not solved $r_{p,s} = \infty$. Performance profiles give an overall assessment of the performance of the solver. If we define

$$\rho_s(\tau) = \frac{size\ \{p \in P : \ r_{p,s} \leq \tau\}}{n_p}.$$

then $\rho_s(\tau)$ is the probability for solver $s \in S$ that a performance ratio $r_{p,s}$ is within a factor $\tau$ of the best possible ratio. The function $\rho_s$ represents the (cumulative) distribution function for the performance ratio. In our plots we hence have function $\rho_s$ on the $y$ axis and $\tau$ on the $x$ axis.

### 3.3.2 QPLIB instances

We start our tests with instances from the QPLIB library. A set of 36 problems is available:

- 19 continuous convex QPs (we include both problems with linear and box constraints);
- 17 continuous relaxations of convex IQPs/MIQPs.

We notice that four instances out of 36 (namely *QPLIB_ 8500*, *QPLIB_ 8547*, *QPLIB_ 9008*, *QPLIB_ 10038*) have a number of variables from 160000 to 1 million variables. Those problems are hence not included in the analysis. Among the remaining 32 instances, we only have six instances with a dense Hessian. In order to show how the efficiency of our code depends on the density, in Figure 3.1 we plot the ratio between the CPU time of SD and the CPU time of *Cplex* to solve the problem, against the density of the problem, in logarithmic scale. The density of a matrix is calculated as the ratio between the number of nonzero entries and the number of entries. Since the matrices are symmetric, each pair of symmetric non diagonal entries counts as one, and the total number of entries is $n(n+1)/2$.

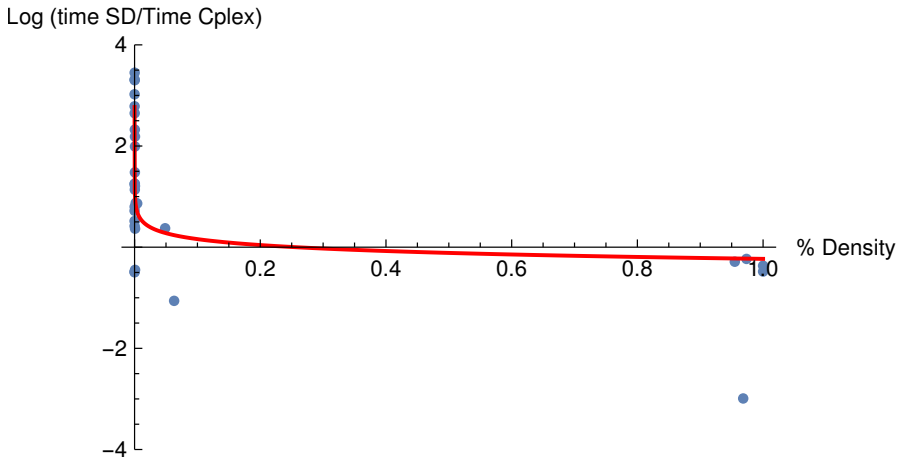



Figure 3.1 – Efficiency against density for QPLIB instances, logarithmic scale

It is clear from the plot and from its trend line, that the more the instance is dense, the more the algorithm is efficient with respect to Cplex, as we could expect. We hence focus on instances with dense Hessian in the next subsections.

### 3.3.3 Specific problems

We now consider medium/large scale continuous QPs having the structure mentioned before. As previously said, we consider:

- problems coming from Computational Geometry (the Chebishev problem, see, e.g., [40] and references therein), Machine Learning/Statistics (the LASSO problem, see, e.g., [131]) and Economics (classic Markowitz Portfolio Optimization problem [106]);

- problems with a combinatorial structure, that is continuous relaxations of instances related to the Quadratic Shortest Path problem [55, 72, 129] and Quadratic Multidimensional Knapsack problem [24, 125, 126].

We compare our algorithm with respect to *Cplex*. In the next subsections, we describe the problems, how we generated the instances and we present the results.

#### Problem description

**The Chebishev center problem:** In this specific problem, the goal is finding the circle of minimum radius that encloses all of the points in a finite set

$$C = \{c_1, \dots, c_n\} \subset \mathbb{R}^m.$$

See [40] for details and applications. The problem can be formulated as a quadratic problem over a simplex:

$$\begin{aligned} \min\ & f(x) = x^\top A^\top A x - \sum_{i=1}^{n} \|c_i\|^2 x_i \\ \text{s.t.}\quad & e^\top x = 1, \\ & x \geq 0, \end{aligned} \tag{3.18}$$

with $x \in \mathbb{R}^n$, $A = (c_1 \dots c_n) \in \mathbb{R}^{m \times n}$.

We constructed our instances by generating a matrix $A \in \mathbb{R}^{m \times n}$ whose entries are normally distributed. The linear term is given by the 2-norm of each column of $A$. We chose three values of $n$: 2048, 4096, and 8192. For each of them, three values of $m$: 10, 100, 1000. For each combination, we used three seeds, so that we have 27 instances in total.

**The LASSO problems:** LASSO, proposed by Tibshirani in 1996 [131], is a popular tool for sparse linear regression. Given the training set

$$T = \{(a^i, b^i),\ a^i \in \mathbb{R}^n,\ b^i \in \mathbb{R},\ i = 1, \dots, m\},$$

the goal is finding a sparse linear model (i.e., a model with a small number of non-zero parameters) describing the data. This problem is strictly connected with the Basis Pursuit Denoising (BPD) Problem in signal analysis (see, e.g., [36]). In this case, given a discrete-time input signal $b$, and a *dictionary*

$$D = \{a_j \in \mathbb{R}^m\ :\ j = 1, \dots, n\}$$

of elementary discrete-time signals, usually called atoms, the goal is finding a sparse linear combination of the atoms that *approximate* the real signal. We formulate LASSO/BPD problem as follows:

$$\begin{aligned} \min_{x \in \mathbb{R}^n}\quad & \|Ax - b\|_2^2 \\ s.t.\quad & \|x\|_1 \leq \tau. \end{aligned} \tag{3.19}$$

The parameter $\tau > 0$ controls the amount of shrinkage that is applied to the model (number of nonzero components in $x$).

We constructed our instances by firstly generating a matrix $A \in \mathbb{R}^{m \times n}$ whose entries are normally distributed. We chose three values of $n$: 2048, 4096, and 8192. For each of them, two values of $m$: $n$, or $n/4$. Then, we generated the solution point $\bar{x}$ with three different levels of sparsity: 0.01, 0.05, and 0.1. The right-hand-side vector $b$ is obtained by $A\bar{x} + \varepsilon$, with a random noise $\varepsilon$. For each combination we generated instances with three different seeds; hence we have in total a benchmark of 54 LASSO instances.

**The portfolio problem:** We consider the formulation for portfolio optimization problems (POP) proposed by Markowitz in [106]. We have $n$ available assets. We call $x_i$ the quantity of money invested on the $i$-th asset during the considered period and with $r_i$ the returns on the $i$-th asset. We have two different constraints. The first one giving a lower bound $\mu$ on the expected return. We then have the budget constraint:

$$\sum_{i=1}^{n} x_i = B,$$

the total amount of money invested needs to be equal to the budget $B$ ($B$ can be simply set to 1). In addition, we impose non-negativity for the variables (i.e., $x_i \geq 0$): it basically means that short selling (selling asset that we still don't own) is not allowed.
We consider a stochastic model for the returns: $r \in \mathbb{R}^n$ is a randomly generated vector with mean $\bar{r}$ and covariance $\Sigma$. Thus, the expected return will be

$$\bar{r}^\top x$$

and variance

$$x^\top \Sigma x.$$

The formulation of Portfolio Optimization Problem (POP) is then a convex quadratic programming problem:

$$\begin{aligned} \min\ & f(x) = x^\top \Sigma x \\ \text{s.t.}\ & r^\top x \geq \mu, \\ & e^\top x = 1, \\ & x \geq 0, \end{aligned} \tag{3.20}$$

The goal is thus finding the set of assets that minimizes the variance (risk connected to the given portfolio) while guaranteeing a specific level of expected return (satisfy budget and non-negativity constraints).

We used data based on time series provided in [4] and [35]. Those data are related to sets of assets of dimension $n = 226, 457, 476, 2196$. The expected return and the covariance matrix are calculated by the related estimators on the time series related to the values of the assets.

In order to analyze the behavior of the algorithm on larger dimensional problems, we created additional instances using data series obtained by modifying the existing ones. More precisely, we considered the set of data with $n = 2196$, and we generated bigger series by adding additional values to the original ones: in order not to have a negligible correlation, we assumed that the additional data have random values close to those of the other assets. For each asset and for each time, we generate from 1 to 4 new values, thus obtaining 4 new instances whose dimensions are multiples of 2196 (that is 4392, 6588, 8784, 10980).

For each of these 8 instances, we chose 5 different thresholds for the expected return: 0.006, 0.007, 0.008, 0.009, 0.01, we thus obtained 40 portfolio optimization instances.

**Continuous relaxations of combinatorial problems (CRCP)** Continuous problems can obviously be viewed as a way to obtain valid dual bounds for combinatorial problems (to be used in a Branch-and-Bound framework). For this reason, we further analyze performances of our SD framework on models related to the continuous relaxation of some combinatorial problems with a quadratic objective function.

In our tests, we used continuous relaxations of quadratic multidimensional knapsack problems (see, e.g., [55, 72, 129]) and quadratic shortest path problems (see, e.g., [24, 125, 126]).

Instances related to the Quadratic Multidimensional Knapsack Problem (QMKP) have the following form:

$$\begin{aligned} \max\ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.}\ & Ax \leq b, \\ & 0 \leq x \leq 1. \end{aligned} \tag{3.21}$$

where $Q \in \mathbb{R}^{n \times n}$ is negative definite, $c \in \mathbb{R}^n$, $A \in {\mathbb{R}_+}^{m \times n}$ and $b \in \mathbb{R}^m_+$.

The instances we used for the quadratic multidimensional knapsack problem are provided by J. Drake in [57]. This benchmark collects various instances, including the ORLib dataset proposed by Chu and Beasley in [37] and the GK dataset proposed by Glover and Kochenberger, mentioned in [73]. In particular, we considered only problems with $n$ greater than 1000. Hence, we kept instances gk09, gk10 and gk11 of Glover and Kochenberger from [57], and we generated other instances using the same criteria described in [37], but using larger values of $n$, that is 5000, 7500 and 10000. We kept $m = 100$ in this last case and we considered two different options to obtain the right hand side. So we generated 6 instances. As regards the objective function, the coefficients related to the linear part were already included in the instances, while for the quadratic part we used again matrices generated in the same way as for the general problems described before. In order to get meaningful results in the end, we suitably scaled the two terms in the objective function with a parameter $\rho$. We used two different seeds to generate the matrix and three different values for $\rho$. So, we have 6 combinations for the objective function for each of the 9 linear problems (the instances gk09, gk10 and gk11 from the literature and the 6 problems generated by us) so we have 54 instances globally.

The second set of continuous relaxation of combinatorial instances are the Quadratic shortest path problems (QSPP). Their formulation is the following:

$$\begin{aligned} \min\ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.}\ & \sum_{e \in \delta^+(s)} x_e = 1, \\ & \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0, \quad \forall v \neq s, t \\ & \sum_{e \in \delta^-(t)} x_e = 1 \\ & 0 \leq x \leq 1. \end{aligned} \tag{3.22}$$

with $c \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n \times n}$. $s, t$ are the source and termination nodes, respectively; $\delta^+(v)$ are the outgoing arcs and $\delta^-(v)$ are the incoming arcs in the node $v$.

The directed graphs used in the experiments are related to two different kind of problems:

1. Grid shortest path problem, that is graphs represented by a squared grid;
2. Random shortest path problem, that is randomly generated graphs (obtained by the generator ch9-1-1 used in the 9th DIMACS implementation challenge [48]).

For grid shortest path instances, we considered square grids of 5 different sizes $k$, that is $k = 30, 40, 50, 60, 70$. We fixed the source and the sink respectively as the top-left and the bottom-right node. The number of variables $n$, same as the number of arcs, is $2 * k * (k - 1)$. Hence we get, respectively: $n = 1740, 3120, 4900, 7080$ and 9660. The number of constraints is the same as the number of vertices of the graph, that is $k^2$.

When generating random shortest path instances, we fixed three values of $n$: 1000, 3000 and 5000; the number of constraints $m$ was chosen in order to get similar densities in the graphs: we respectively chose $m = 100$ and 150 for $n = 1000$, $m = 150$ and 250 for $n = 3000$ and $m = 200$ and 300 for $n = 5000$. In this way we obtained graphs with densities (number of arcs over number of arcs of a complete graph with the same number of nodes) that vary between 10% and 25%.

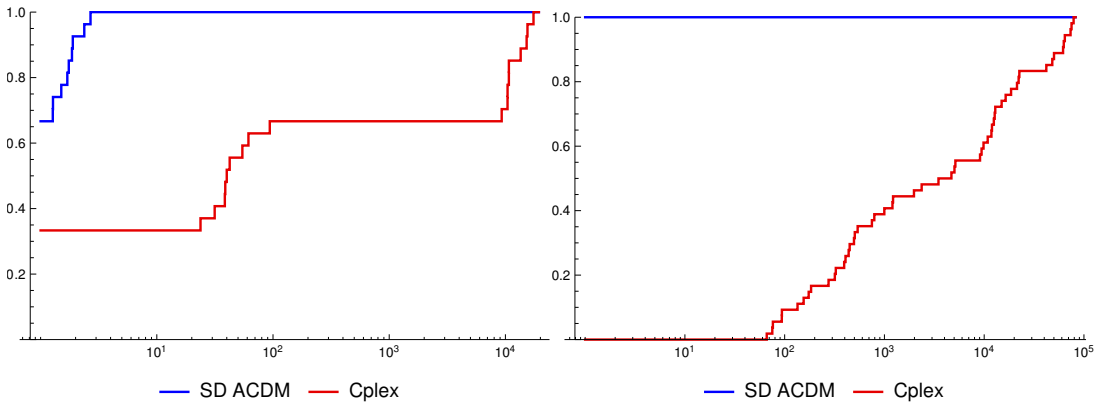
For both classes, we built up the objective function in this way: we defined the quadratic part with matrices generated in the same way as for the general problems described before; then we added linear coefficients for the linear part, generated in three different intervals: $[0.05, 0.4]$, $[0.5, 1.0]$ and $[2.0, 3.0]$. We used two different seeds for generating the quadratic part and for the linear part we considered three different choices, so we have 6 problems for each value of $n$ and $m$. In this way we obtained 30 different problems for the Grid shortest path, where $m$ is fixed depending on $n$, while we got 72 instances of Random shortest path, because for each $n$ we got 2 different values for $m$ and for each of them we used two different seeds for generating the graph.

The benchmark consists of 30 instances based on grid graphs (QGSPP) and 72 instances based on random graphs (QRSPP).

**Results**

In Figure 3.2, we present the results on these six classes of problems, by means of performance profiles. In all the instances we compared the ACDM based SD framework with *Cplex*.
On LASSO and Chebishev problems, we also generated a Frank-Wolfe ad hoc algorithm, adapted to the domain. We used the away-step version for Chebishev problems and the Pairwise version for the LASSO problem, because they have shown to be the most efficient ones, respectively, for the two classes. For the largest instances, it worked better than Cplex, but on average Cplex is better.

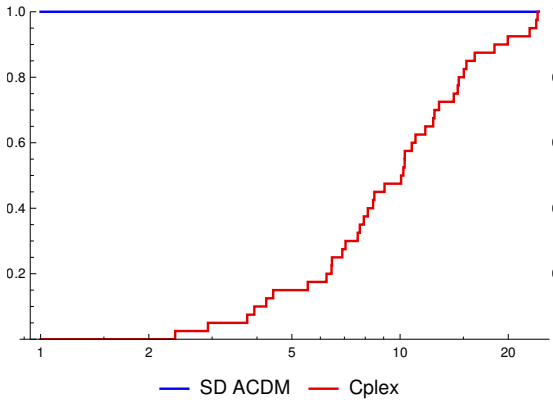
The results show that SD outperforms Cplex in all the considered problems. In order to show the improvements of the master and pricing techniques introduced in the previous



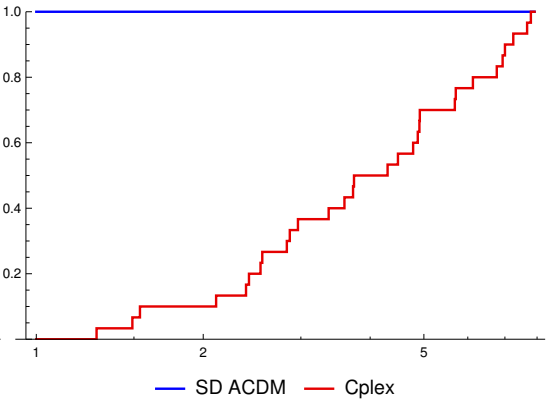

(a) Chebishev



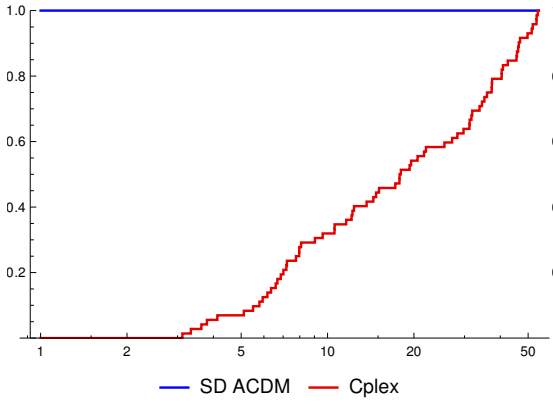

(b) LASSO



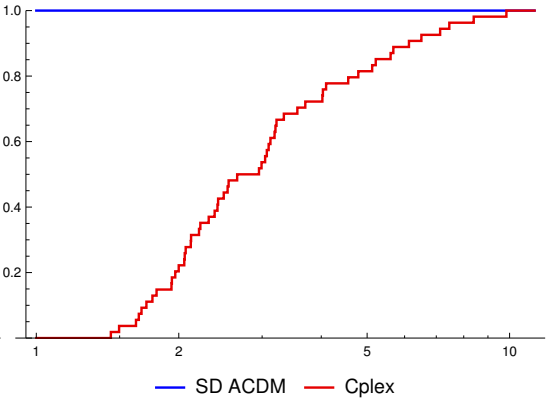

(c) Portfolio




(d) QGSPP




(e) QRSPP




(f) QMKP

Figure 3.2 – Performance analysis of SD ACDM and Cplex.

sections, we extend the benchmark with some randomly generated instances, which require a higher computational time.

### 3.3.4 Extended benchmark

We generated more test instances in order to evaluate the performance of the master and pricing options. To this aim we generated a statistically significant set of quadratic instances. They have the following form:

$$\begin{aligned} \min\ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.}\ & Ax \geq b, \\ & l \leq x \leq u. \end{aligned} \tag{3.23}$$

with $Q \in \mathbb{R}^{n\times n}$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ , $l, u \in \mathbb{R}^n$ and both finite. In particular, $Q$ was built starting from its singular value decomposition using the following procedure:

- the $n$ eigenvalues were chosen in such a way that they are all positive and equally distributed in the interval $(0, 3]$;
- the $n \times n$ diagonal matrix $S$, containing these eigenvalues in its diagonal, was constructed;
- an orthogonal, $n \times n$ matrix $U$ was supplied by the QR factorization of a randomly generated $n \times n$ square matrix;
- finally, the desired matrix $Q$ was given by $Q = USU^\top$, so that it is symmetric and its eigenvalues are exactly the ones we chose.

The coefficients of the linear part of the objective function were randomly obtained, in a small range, between 0.05 and 0.4, in order to make the solution of the problem quite sparse.

The $m$ constraints (with $m \ll n$) were generated in two different ways: step-wise sparse constraints (S) or random dense ones (R). In the first case, for each constraint, the coefficients associated to short overlapping sequences of consecutive variables were set equal to 1 and the rest equal to 0. More specifically, if $m$ is the number of constraints and $n$ is the number of columns, we defined $s = 2 * n/(m + 1)$ and all the coefficients of each $i$-th constraint are zero except for a sequence of $s$ consecutive ones, starting at the position $1 + (s/2) * (i - 1)$. In the second case, each coefficient of the constraint matrix takes a uniformly generated random value in the interval $[0, 1]$. The right-hand side was generated in such a way to make all the problems feasible: for the step-wise constraints, the right hand side was set equal to $f * s/n$, with $0.4 \leq f \leq 1$ and for a given random constraint, the corresponding right-hand side $b$ was a convex combination of the minimum $a_{min}$ and the maximum $a_{max}$ of the coefficients related to the constraint itself, that is $b = 0.75 * a_{min} + 0.25 * a_{max}$.

Each class of constraints was then possibly combined with two additional type of constraints: a budget type constraint (b) $e^\top x = 1$, and a "relaxed" budget type constraints (rb) $slb \leq e^\top x \leq sub$. Summarizing, we obtained six different classes of instances:

- S, instances with step-wise constraints only;
- S-b, instances with both step-wise constraints and budget constraint;
- S-rb, instances with both step-wise and relaxed budget constraints;
- R, instances with dense random constraints only;
- R-b, instances with both dense random constraints and budget constraint;
- R-rb, instances with both dense random and relaxed budget constraints.

For each class, we fixed $n = 2000, 3000, \ldots, 10000$, while the number of both step-wise and dense random constraints $m$ was chosen in two different ways:

1) $m = 2$, 22, 42 for each value of $n$;

2) $m = n/32$, $n/16$, $n/8$, $n/4$, $n/2$ for each value of $n$.

In the first case, we then have problems with a small number of constraints, while, in the second case, we have problems with a large number of constraints. Finally, for each class and combination of $n$ and $m$ we randomly generated five instances. Hence, the total number of instances with a small number of constraints was 450 and the total number of instances with a large number of constraints was 750. The benchmark of Generic instances is split into two sets: the first one consists of 450 instances *with Small number of constraints* (GS) and the second one of 750 instances *with Large number of constraints* (GL).

The instances we generated are positive definite; anyway, we would like to point out that our algorithm can also solve positive semidefinite instances, as is shown in Section 3.3.5. We restricted to positive definite ones because after reading ill-conditioned or positive semidefinite matrices, Cplex returns an error, since it is not able to provide guarantee of symmetry and of positive semidefiniteness. Nevertheless, for our algorithm the performance is very similar.

### 3.3.5 Preliminary tests

Here, we first describe the way we chose the *Cplex* optimizer for solving our convex quadratic instances. Then, we explain how we set the parameters in the different algorithms used to solve the master problem in the SD framework.

**Choice of the Cplex optimizer**

As already mentioned, we decided to benchmark our algorithm against *Cplex* version 12.6.2 (see [97] for further details). The optimizers that can be used in *Cplex* for solving convex quadratic continuous problems are the following: primal simplex, dual simplex, network simplex, barrier, sifting and concurrent. The aim of our first test was to identify, among the 6 different options, which is the most efficient for solving instances with a dense $Q$ and $n \gg m$.

In Table 3.1, we present the results concerning instances with 42 constraints and three different dimensions $n$: 2000, 4000 and 6000. We chose problems with a small number of constraints in order to be sure to pick the best *Cplex* optimizer for those problems where the SD framework is supposed to give very good performances. For a fixed $n$, three different instances were solved of all six problem types. So, each entry of Table 3.1 represents the averages computing times over 18 instances. A time limit of 1000 seconds was imposed and in brackets we report (if any) the number of instances that reached the time limit.

| $n$ | Default | Primal | Dual | Network | Barrier | Sifting | Concurrent |
|---|---|---|---|---|---|---|---|
| 2000 | 72.2 | 1.6 | 1.6 | 1.6 | 84.2 | 2.0 | 89.0 |
| 4000 | 641.8 (2) | 12.7 | 13.9 | 13.9 | 618.0 (2) | 11.5 | 689.4 (2) |
| 6000 | 1000.0 (18) | 31.5 | 30.7 | 30.5 | 1000.0 (18) | 26.3 | 1000.0 (18) |

Table 3.1 – Comparison among the different Cplex optimizers

The table clearly shows that the default optimizer, the barrier and the concurrent methods give poor performances when dealing with the quadratic programs we previously described. On the other side, the simplex type algorithms and the sifting algorithm seem to be very fast for those instances. In particular, sifting gives the overall best performance. Taking into account these results, we decided to use the *Cplex* sifting optimizer as the baseline method in our experiments. It is worth noticing that the sifting algorithm is specifically conceived by *Cplex* to deal with problems with $n \gg m$, representing an additional reason for comparing our algorithmic framework against this specific *Cplex* optimizer.

**Tolerance setting when solving the master problem**

We have three options available for solving the master problem in the SD framework: ACDM, FGPM and *Cplex*. In order to identify the best choice, we need to properly set tolerances for those methods. When using *Cplex* as the master solver, we decided to keep the tolerance to its default value (that is $1E10-6$). The peculiar aspect of ACDM is that no tolerance needs to be fixed a priori. On the other hand, with FGPM, the tolerance setting phase is very importance since, as we will see, it can significantly change the performance of the algorithm in the end.

In Table 3.2, we compare the different behaviors of our SD framework for the three different choices of master solver. Each line of the table represents the average values concerning the 54 instances used in the previous experiment. Column "T" represents the time (in seconds) spent by the algorithms. "Er" and "Max Er" represent the average and maximum relative errors with respect to the value found by *Cplex* (using sifting optimizer). "Ei" and "Max Ei" represent the average and maximum distance (calculated using $\ell_\infty$ norm) from the solution found by *Cplex*. In the last column, "Dim" represents the dimension of the final master program.

| Solver | Tol | T (s) | Er | Max Er | Ei | Max Ei | Dim |
|---|---|---|---|---|---|---|---|
| | 1E-02 | 0.25 | 8.64E-02 | 2.67E-01 | 2.24E-02 | 5.04E-02 | 9.9 |
| | 1E-04 | 1.15 | 2.21E-04 | 6.79E-04 | 7.80E-04 | 1.44E-03 | 55.6 |
| SD FGPM | 1E-06 | 2.46 | 5.65E-07 | 2.63E-06 | 5.72E-05 | 1.86E-04 | 102.2 |
| | 1E-08 | 6.09 | 5.98E-09 | 1.15E-07 | 4.61E-06 | 1.88E-05 | 114.0 |
| | 1E-10 | 9.81 | 2.35E-09 | 4.59E-08 | 3.48E-06 | 2.16E-05 | 113.4 |
| SD Cplex | 1E-06 | 4.66 | 8.86E-09 | 4.26E-08 | 5.50E-06 | 2.46E-05 | 156.0 |
| SD ACDM | None | 3.63 | 1.53E-09 | 1.97E-08 | 2.65E-06 | 1.99E-05 | 113.1 |
| Cplex | | 4.29 | | | | | |

Table 3.2 – Comparison for the three different choices of master solver (Cplex indicates the results obtained with sifting optimizer).

By taking a look at the table, we can easily see that the ACDM based SD framework gets the best results in terms of errors with respect to *Cplex*. We can also see that the performance of the FGPM based one really changes depending on the tolerance chosen. If we want to get for FGPM the same errors as ACDM, we need to set the tolerance to very low values, thus considerably slowing down the algorithm. In the end, we decided to use a tolerance of $10E-6$ for FGPM, which gives a good trade-off between computational time and accuracy. This means anyway that we gave up precision to keep the algorithm fast with respect to ACDM.

**Choice of the $\varepsilon$ parameter for the early stopping pricing option**

In this section we discuss how to fix the threshold $\varepsilon$ used in equation (3.12) for the Early Stopping option. We decided to fix the value of $\varepsilon$ as a fraction $\varepsilon_0$ of the quantity $|\nabla f(x_k)^\top x_k|$:

$$\varepsilon = -\varepsilon_0 |\nabla f(x_k)^\top x_k|. \tag{3.24}$$

The value of $\varepsilon_0$ has been chosen after testing three different values on a subset of instances. We chose the subset of the randomly generated instances with random dense constraints and budget constraint, where SD has the worst behavior with respect to *Cplex*. The results are presented in Table 3.3, where we compare the average computational time $T$

and the number of SD iterations *N its*. The table presents the results on the 67 instances solved by all the algorithms.

| Solver | $\varepsilon_0$ | T (s) | N its |
|---|---|---|---|
| SD | 0.0 | 77.4 | 165.0 |
| | 0.5 | 80.3 | 188.2 |
| | 1.0 | 70.7 | 165.0 |
| | 1.5 | 73.0 | 165.0 |
| Cplex | | 9.4 | |

Table 3.3 – Test on the $\varepsilon_0$ parameter for the Early Stopping technique.

One can see that, with $\varepsilon_0 = 0.5$ , the time and number of iterations are larger. On the other hand, if $\varepsilon_0 = 1.5$, the threshold is too weak and the early stopping is never used. Hence, we chose the value of $\varepsilon_0 = 1.0$, which improves the computational time while keeping the number of iterations unchanged.

**Ill-conditioned and positive semidefinite matrices**

In the section, we presented results for positive definite instances with eigenvalues equally distributed in the range $[10^{-4}, 3]$, so with a condition number of $3 \times 10^4$.
In here, we report results obtained when varying condition number and percentage of null eigenvalues in the Hessian matrix of the randomly generated QPs. More specifically, for each choice of condition number and percentage of null eigenvalues, we randomly generate 5 problems with 2000 variables and 5 problems with 4000 variables. We consider the following choices:

- 4 different percentages of null eigenvalues : 0%, 1%, 5%, 20% ;
- condition number of 5 different orders: $10^4$, $10^8$, $10^{12}$, $10^{16}$, $10^{20}$.

Hence, we have 20 different combinations for a total number of 200 instances.

We notice that *Cplex* had a significant number of failures on those instances. Indeed, it was able to solve only the positive definite instances with condition number up to $10^{12}$. It returned and error in the other cases: this is mainly due to the high density of the matrix that makes hard to detect the symmetry or the positive semidefiniteness of the Hessian. SD ACDM was instead able to solve the vast majority of the instances (we only got a few failures for condition number $10^{20}$) and it was faster on those instances that *Cplex* was able to solve. In Table 3.4, the average CPU time in seconds is reported for SD ACDM (with the default pricing option). Each column represents the percentage of null eigenvalues and each row stands for the order of magnitude of each condition number considered. SD ACDM with the best pricing option, that for these instances is *Sifting + Cuts*, obtains similar results in terms of CPU time.

We would like to notice that the performance of our algorithm is not much affected by the increase of the condition number and on the percentage of null eigenvalues. This is

| | 0% | 1% | 5% | 20% |
|---|---|---|---|---|
| $10^4$ | 0.53 | 0.55 | 0.65 | 0.71 |
| $10^8$ | 0.61 | 0.62 | 0.62 | 0.85 |
| $10^{12}$ | 0.60 | 0.65 | 0.70 | 0.84 |
| $10^{16}$ | 0.64 | 0.57 | 0.69 | 0.76 |
| $10^{20}$ | 0.57 | 0.58 | 0.69 | 0.82 |

Table 3.4 – Average CPU time (SD ACDM).

the reason why in our tests we mainly concentrate on positive definite instances, with the indicated condition number. We would anyway like to remark that specific techniques, e.g. preconditioning strategies, can be used (and are actually used in many solvers) to tackle the class of ill-conditioned problems. Some preconditioning might hence be embedded in our framework as well in order to improve the performance, but this might be subject of future research.

### 3.3.6 Numerical results related to the extended testbed

In this section, we report the numerical results of our SD framework.

In the first part of the analysis, we investigate how the use of different options for solving the master problem influences the overall performances of the algorithm. We show the results concerning the following three different settings for the master problem:

- ACDM: the new conjugate direction method, presented in Section 3.1.1.
- FGPM: the gradient projection method explained in Section 3.1.2.
- *Cplex*: the continuous optimizer of *Cplex*, default settings.

In the second part, we test the impact of the following pricing options:

- Default: the pricing problem is solved with the Linear Programming optimizers of *Cplex*, default settings.
- Cuts: we add to the Default option the Shrinking cuts, described in Section 3.2.2.
- Early stopping: we add to the Default option the Early stopping technique described in Section 3.2.1 .
- Cuts + Early stopping: both techniques are added to the Default option.

We further compare the default option of *Cplex* with more specific options like the Sifting optimizer and the Network optimizer.

### Master solvers

Now, we focus on the computational analysis of the different methods used for solving the master problem in the SD framework. Figure 3.3 provides the results concerning all the classes of problems previously introduced. We indicate with SD-Cplex, SD-ACDM and SD-FGPM the results concerning SD using respectively *Cplex*, ACDM and FGPM for solving the master problem. For the sake of comparison, we also include the performances of *Cplex*.

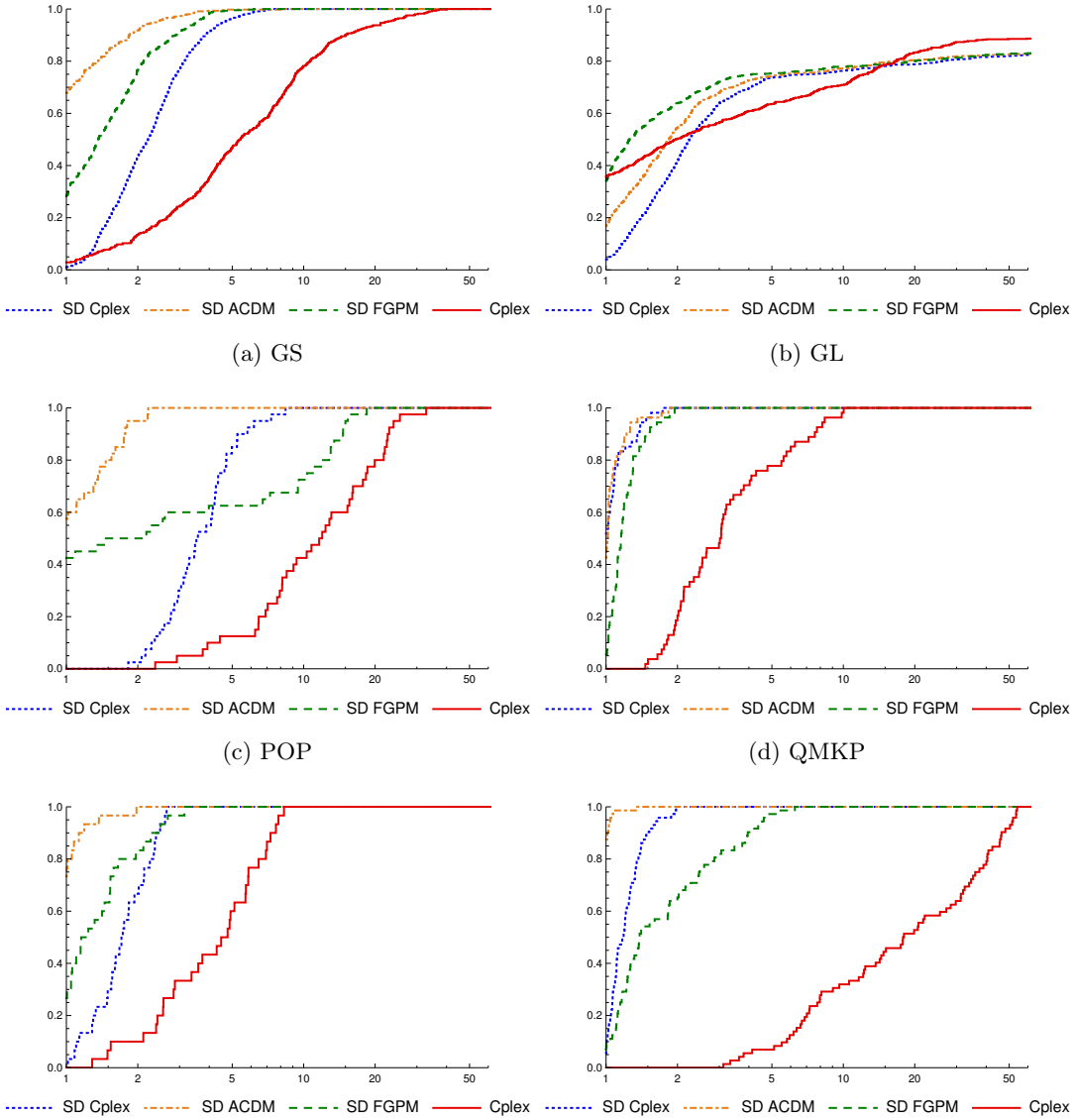
These plots show that the SD framework significantly outperforms *Cplex* in the vast majority of the cases. We can further see that SD-ACDM is the most efficient and robust for almost all the classes of problems considered (more precisely, GS, POP, QMKP, QGSPP and QRSPP). As regards the GL instances, we notice that SD-FGPM has better performances than both SD-ACDM and SD-Cplex and that the Cplex solver is competitive with it.

### Pricing Options

Here, we analyze the impact of the different pricing options in the SD framework. For each class of problems, we use as master solver the most effective method, according to the results of the previous section. Hence, ACDM is used for the GS, POP, QMKP, QGSPP and QRSPP instances and FGPM is used for the GL instances. Figure 3.4 shows the results concerning all the classes of problems considered. We indicate with Default the results concerning SD using *Cplex* with default settings to solve the pricing problem. Furthermore, we use Default+Cuts, Default+Early Stopping and Default + Early Stopping + Cuts to indicate the results obtained when respectively adding to Default the shrinking cuts, the early stopping procedure and both at the same time.

We notice that the option Default + Early Stopping shows the best performances (both in terms of efficiency and robustness) for the GL, GP, POP and QRSPP instances. With respect to QGSPP, Default + Early Stopping is still the most efficient, but the Default version is slightly better in terms of robustness. We further notice that the option Default + Early Stopping + Cuts is competitive with the option Default + Early Stopping on the GL instances, and it is as robust as the option Default + Early Stopping on the POP instances. Finally, if we consider the QMKP instances, the option Default is the most efficient, while the option Default + Early Stopping + Cuts is the most robust.

Figure 3.5 finally shows the effects of replacing the Default *Cplex* solver for the pricing problem with the Sifting/Network optimizer. We only focus on two specific classes of problems where it makes sense to use such tailored approaches. More specifically, we considered generic quadratic instances to test the Sifting and quadratic shortest path problems to test the Network solver. We compare the best pricing option obtained from the analysis carried out in Figure 3.4 with the different sifting variants. We would like to highlight that, when using those tailored solvers in the pricing, early stopping can only be implemented by means of callback functions. Since this would surely worsen the performances of the framework, we decided not to include the option in the analysis. As we can easily see by taking a look at the plots, the option Sifting+Cuts is the best

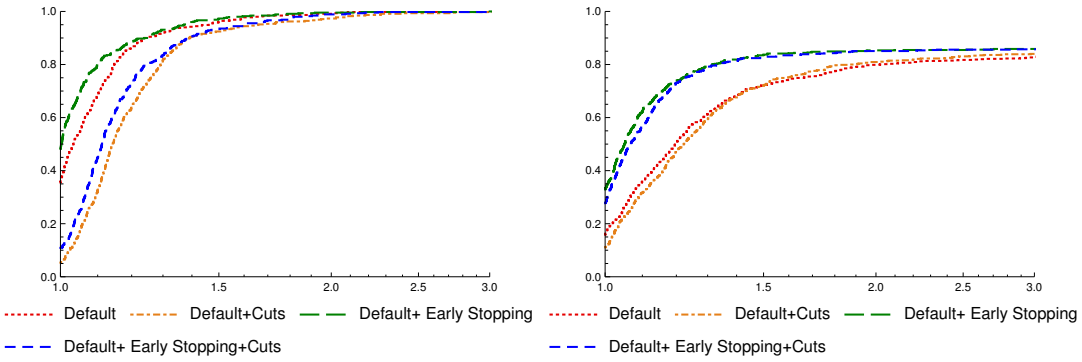



(a) GS (b) GL

(c) POP (d) QMKP

(e) QGSPP (f) QRSPP

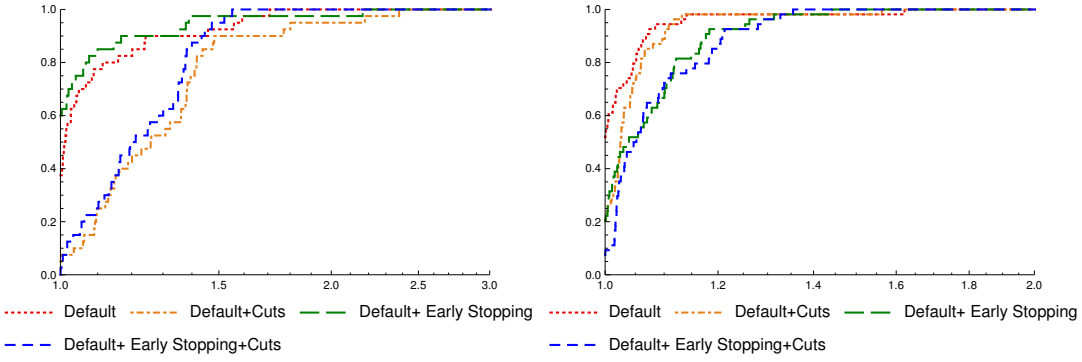
Figure 3.3 – Performance analysis of the different methods used for solving the master problem in the SD framework.




(a) GS



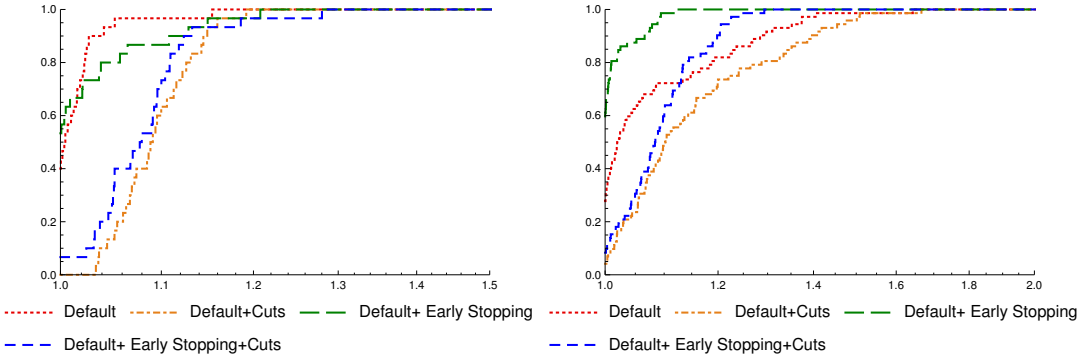

(b) GL




(c) POP




(d) QMKP

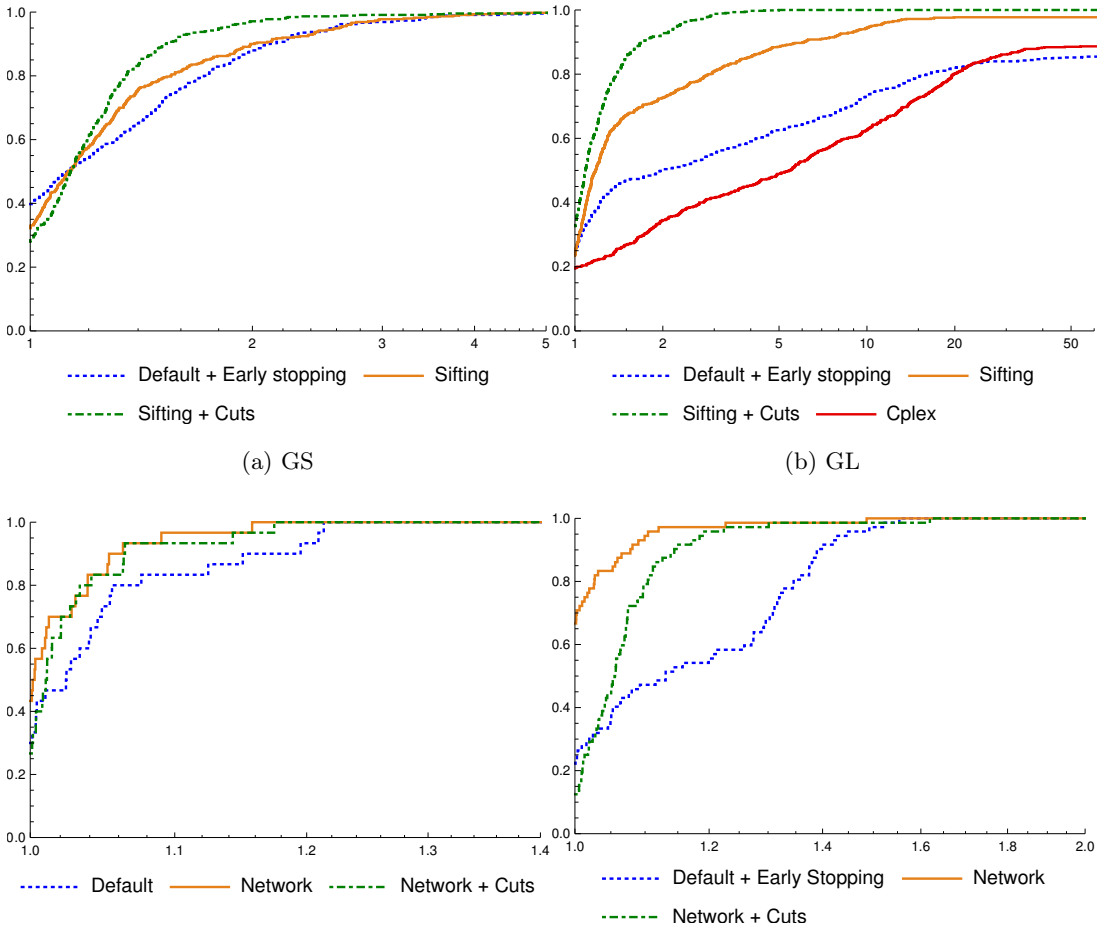



(e) QGSPP




(f) QRSPP

Figure 3.4 – Performance analysis of the different options used for solving the pricing problem in the SD framework.




(a) GS (b) GL

(c) QGSPP (d) QRSPP

Figure 3.5 – Additional performance analysis of the different options used for solving the pricing problem in the SD framework.

one when dealing with GS and GL instances. In particular, for the GL instances the SD framework significantly outperforms also the baseline *Cplex* solver and finds a solution for all the instances within the time limit. The Network option, on the other hand, guarantees good results on both QGSPP and QRSPP instances.

**Average computational time**

In our experiments, we fixed a time limit of 900 seconds for all the algorithms. For each class of problems, we report in Table 3.5 the number of available instances (*N inst*), the number of instances solved within the time limit and the average computational time in seconds spent by *Cplex* (*NS Cplex* and *T Cplex*) and by SD (*NS SD* and *T SD*). We consider the best master/pricing options for SD in the analysis. The average is done by taking into account only the instances solved by both the algorithms. Furthermore, we add the average number of SD iterations (*N it*) needed to solve the problems.

| Class | N inst | NS Cplex | T Cplex | NS SD | T SD | N it |
|---|---|---|---|---|---|---|
| GS | 450 | 450 | 11.7 | 450 | 2.4 | 171.4 |
| GL | 750 | 666 | 63.8 | 750 | 16.7 | 90.7 |
| POP | 40 | 40 | 9.6 | 40 | 0.7 | 116.6 |
| QMKP | 54 | 54 | 36.5 | 54 | 11.7 | 31.7 |
| QGSPP | 30 | 30 | 77.6 | 30 | 15.0 | 290.4 |
| QRSPP | 72 | 72 | 2.2 | 72 | 0.1 | 19.7 |

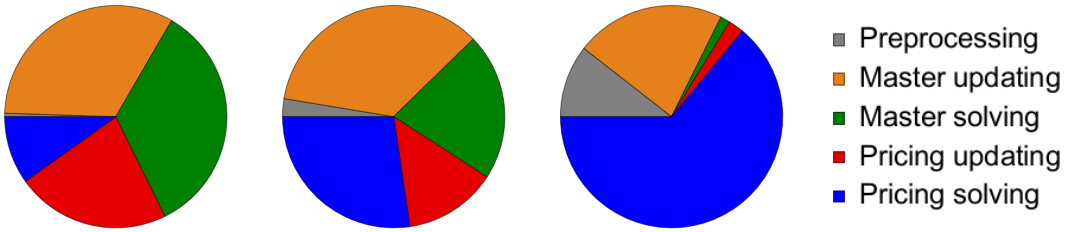
Table 3.5 – Solved instances and average CPU time.

In particular, we see that in GL problems, 84 instances out of 750 are not solved by *Cplex* within the time limit, while SD with the improving tools for both master and pricing solves all of them. We finally highlight that the most time consuming part in each SD cycle is the solution of the pricing problem, as we point out in the next paragraph.

### 3.3.7 CPU time usage in the SD framework

Now we analyze the way CPU time is used in the SD framework, that is we show the average CPU time needed for preprocessing data, solving the master problems and solving the pricing problems (failures are not considered in the analysis). In Figures 3.6 and 3.7, we report the aggregated results over the first three classes of instances and on the continuous relaxations of combinatorial instances, respectively. In each figure, we report the time spent by SD in the preprocessing phase of the algorithm (*preprocessing*) and in the solution of the master and pricing problem. The solving time of both the pricing and master problem is split in the time needed to update the data structures (*updating*) and the time needed to solve the problem (*solvers*). For each figure we provide also the average computing time over the testbed. Figures 3.6 clearly suggests that for generic instances the percentage of computing time of the pricing problem increases with the increase of the size of the instances. On the other hand, the subdivision of CPU

times differs significantly for the three continuous relaxations of combinatorial instances considered (QGSP, QRSP, QMK). First of all, we observe that for quadratic shortest path instances the percentage of computing time for the pricing is lower than the one for the quadratic multidimensional knapsack instances. This is due to the fact that the pricing problem for a quadratic shortest path instance reduces to a simple shortest path problem and thus it can be handled efficiently by a generic LP solver. Finally, we notice that the preprocessing time for random shorthest path is not negligible. This behaviour is due to the fact that the overall computing time is significantly small and hence the total time needed to prepare the initial data structures cannot be ignored.



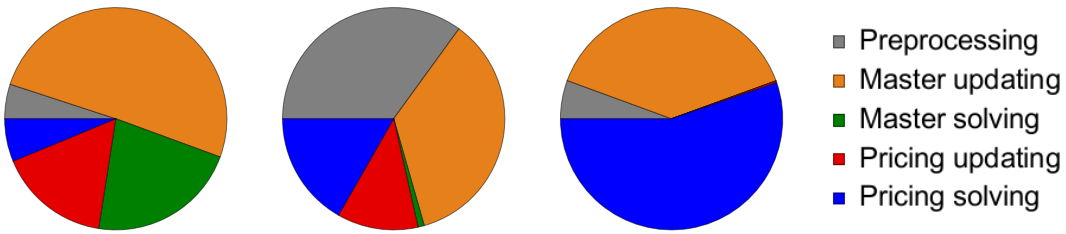

(a) Portfolio
Avg CPU Time:
SD-ACDM: 0.90s
Cplex: 9.58s

(b) GS
Avg CPU Time:
SD-ACDM: 3.66s
Cplex: 14.54s

(c) GL
Avg CPU Time:
SD-FGPM: 20.02s
Cplex: 59.84s

(d) Legend

Figure 3.6 – CPU time pie charts for Portfolio and General Problems.




(a) QGSP
Avg CPU Time:
SD-FGPM: 34.40s
Cplex: 93.72s

(b) QRSP
Avg CPU Time:
SD-ACDM: 0.25s
Cplex: 2.64s

(c) QMK
Avg CPU Time:
SD-ACDM: 24.40s
Cplex: 52.80s

(d) Legend

Figure 3.7 – CPU time pie charts (continuous relaxations of combinatorial instances).

### 3.3.8 In-depth analysis

In order to better analyze the behaviour of the SD framework, we show now how the objective function value changes with respect to the elapsed time. Since we want to get meaningful results, we only consider generic instances solved in more than 10 seconds (but always within the time limit of 900 seconds). In particular, we consider instances with random dense constraints and we take a set of 25 instances for each of the three types of additional constraints. Hence, we plot

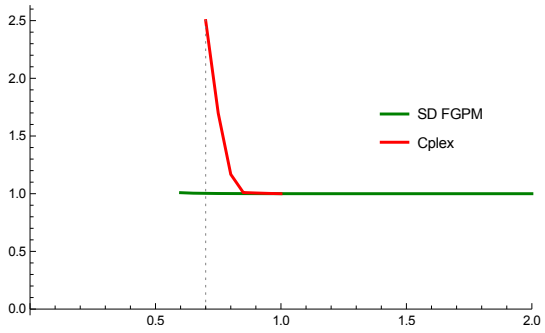
- on the *x-axis* the **CPU time ratio**, that is the CPU time elapsed divided by the overall time needed by *Cplex* to get a solution on the same instance.
- on the *y-axis* the **objective function ratio**, that is the objective function value divided by the optimal value obtained by *Cplex* on the same instance.

All the results are averaged over the whole set of instances. For the SD framework, we plot the results up to twice the time needed by *Cplex* to get a solution. In the analysis, we always consider the setting that includes all the pricing options (and gives same performance as the best one). Figures 3.8a and 3.8b show the overall results for the 75 instances considered: the first figure shows the comparison between *Cplex* and SD FGPM, while the second one shows the comparison of the three different SD framework versions. From the comparison of *Cplex* and SD FGPM, it is easy to notice that SD gets a good objective function value very soon. Indeed, at a CPU time ratio 0.6 (i.e., 60% of the overall *Cplex* CPU time) corresponds an objective function ratio slightly bigger than 1 for SD FGPM, while at the same CPU time ratio *Cplex* still needs to find a feasible solution. *Cplex* gets a first feasible solution for a CPU time ratio equal to 0.7 (in this case the objective function ratio is bigger than 2.5), and it obtains an objective function ratio close to 1 only for a CPU time ratio bigger than 0.8. By taking a look at the comparison of the three different versions of our SD framework, we notice that SD FGPM actually takes longer than the others to get an objective function ratio close to 1. The better results obtained for SD FGPM hence depend, as we already noticed, on the way we choose the tolerance in the master solvers. Finally, in Figure 3.8c, we report the plots related to those instances where *Cplex* outperforms the SD framework. Once again, we can see that SD FGPM gets a good objective function ratio very soon, while *Cplex* takes much longer to obtain a similar ratio.

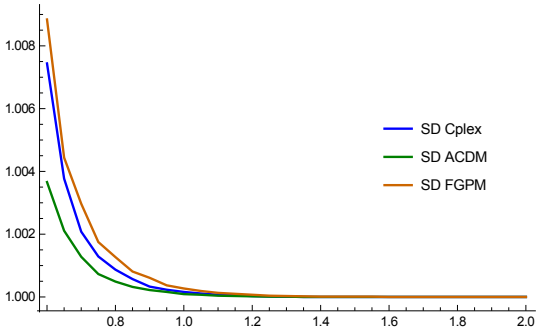
## 3.4 Conclusions

We presented an SD framework to solve continuous convex quadratic problems. In particular, we focused on solving instances with significantly more variables than constraints and with an objective function having a dense Hessian. We motivated our choice by showing literature problems with this form.

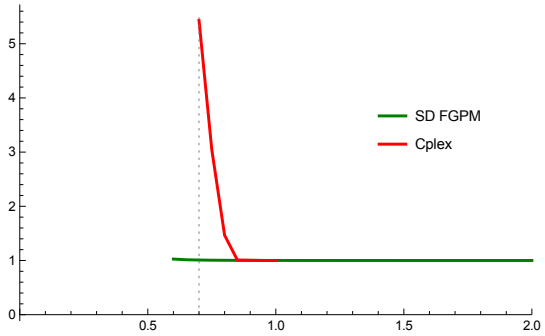
We introduced a new adaptive conjugate direction method (ACDM) that is specifically designed to repeatedly solve the master problem of a SD algorithm; we also used a method that conveniently adapts the projected gradient approach to this framework. Furthermore,




(a) SD FGPM vs Cplex.




(b) SD solvers comparison.




(c) SD FGPM vs Cplex - GL instances (Rb constraints).

Figure 3.8 – Objective function decay - Objective function ratio (y-axis) and CPU time ratio (x-axis).

two different strategies to speed up the pricing were analyzed: an early stopping technique and a method to shrink the feasible region based on ad-hoc cuts. Finally, specific options for solving the pricing problem were tested, namely the sifting and the network optimizer.

Our tests on real instances from the literature, and on randomly generated QPs, show that our algorithm is promising for instances with the aforementioned structure. We also generated extended instances to test our methods.

We carefully analyzed the impact of the different master and pricing settings and we showed that our algorithm is significantly better than *Cplex*. In particular, ACDM seems to be a key ingredient to obtain an effective SD framework. Finally, the pricing options allowed to further enhance the performances of our method.

## 3.5 Future research directions

In this chapter we exploited the properties of the well-known Simplicial Decomposition algorithm and added to them some adaptations in order to tackle a class of convex continuous quadratic problems. We obtained remarkable results and we submitted our work to a journal.

As a possible future investigation, we could try to adapt out framework to general convex problems. Indeed, the master algorithm *FGPM* can be applied to any kind of convex problem, while *ACDM* should be strongly modified. A first example could be that of *Perspective functions* (see [67]). In this framework, variants of the classic algorithm should be taken into consideration, in particular the so-called *Restricted Simplicial decomposition*, that is a modification of the original algorithm given by adding an upper bound on the dimension of the domains of the master problems.

As a different research direction, we could extend our framework in order to deal with mixed integer, convex quadratic problems. Our idea is to develop a branch and bound strategy that solves, at each node, the continuous relaxation of a mixed integer problem by Simplicial Decomposition. Some advantages can be exploited, in particular: firstly, the lower bounds given by the pricing problems help accelerating the computations in the nodes that can be pruned by bound; secondly, the columns generated at each node can provide a warmstart. This is what we present in Chapter 4, along with the results obtained in some combinatorial problems whose continuous relaxations have been considered above.

# Chapter 4

# A simplicial decomposition framework for dense convex quadratic mixed binary problems

## 4.1 Introduction

Many real-world applications can be modelled as *Mixed Binary Quadratic Problems* (MBQPs): this means the optimization of a quadratic objective function subject to linear or quadratic constraints, where all or a part of the variables must be binary numbers, i.e. 0 or 1. We focus on minimization of binary problems with quadratic convex objective function, subject to linear constraints. The form of these problems is the following:

$$\begin{aligned} \min\ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.}\ & Ax \geq b, \\ & Cx = d, \\ & l \leq x \leq u \\ & x_i \in \{0,1\} \quad \forall i \in I \subseteq \{1,\dots,n\} \end{aligned} \tag{4.1}$$

with $Q \in \mathbb{R}^{n\times n}$, $c, l, u \in \mathbb{R}^n$, $A \in \mathbb{R}^{m_1 \times n}$, $b \in \mathbb{R}^{m_1}$ , $C \in \mathbb{R}^{m_2 \times n}$, $d \in \mathbb{R}^{m_2}$, $n, m_1, m_2 \in \mathbb{N}$.

Moreover, we assume the same further hypotheses as the continuous case:

- $X = \{x \in \mathbb{R}^n : \ Ax \ \geq \ b, \ Cx \ = \ d, \ l \leq \ x \leq u\}$ is non-empty and bounded;
- the Hessian matrix $Q$ is positive semidefinite and dense.
- an optimal solution of the continuous relaxation can be obtained as a proper convex combination of a small subset of vertices in the original feasible set;
- there exists an efficient method for minimizing a linear function over the continuous relaxation of the feasible set.

In practice, typically we deal with convex problems with a dense objective function and relatively low number of constraints.
The class of these problems is NP-Hard and we present an algorithm to solve them. It is based on the classic Branch and Bound method, but integrates it with a *Simplicial Decomposition type* approach, to solve the continuous relaxation of the mixed binary problem. Indeed, in chapter 3 we showed that SD is specifically tailored to solve the continuous relaxation of problems with the aforementioned features. However, it is worth noting that the proposed algorithm can handle any convex problem of type (4.1) and can be easily modified in order to deal with mixed integer problems and also problems having a general convex objective function. Also within this framework we obtained promising results compared with the state-of-the-art solver *CPLEX* on some sets of instances.

## 4.2 SD integrated in a Branch and bound

The basic structure of our algorithm is a branch and bound, where at each node we solve the continuous relaxation of the problem through the Simplicial Decomposition algorithm. We wanted to embed the SD algorithm in this structure for several reasons. The first one is that in Chapter 3 we noticed that, at least for large-size problems of this class, this algorithm has a good performance in terms of computational time with respect to *CPLEX*, so it can improve the performance in solving each node. Moreover, SD and in particular the ACDM method described in Section 3.1.1, can take advantage of the structure of a B&B tree by storing information in each node to simplify the computations of the following nodes. A more detailed description of how we can efficiently embed SD in it is proposed in the following sections. We will discuss the branching strategies and rules, the properties of SD that will help in this context as well as computational aspects and results will be described.

### 4.2.1 Branching strategy, branching rules

The branching strategy that has been used is the *depth first search* (DFS): at each branching, the left child is the next node to explore. The advantages of this choice are the following:

- the number of open nodes is kept small: indeed, at any step of the algorithm, at most $n + 1$ nodes are opened, and this is the least possible;
- it lets us find rapidly both upper and lower bounds;
- it can be implemented recursively: in this way it is not too memory consuming and we can effectively take advantage of the SD framework, as will be explained later.

The branching rule for our experiments is the most fractional value: at each branching, we fixed to 1 the variable with the largest fractional part. This choice is driven by the fact that the solutions of our problems are generally sparse, so this should allow us to find rapidly a good upper bound, even without heuristics, and often to calculate the optimal

value soon, as will be showed later. Other branching rules are tested, for instance the so-called "most integer": we fix the fractional variable to the closest integer point, 0 or 1. However, the performances obtained with this rule are similar to those obtained with the other one.

### 4.2.2 Column exploitation

Another very useful enhancement of this algorithm is that many columns generated by the Simplicial Decomposition at each node can be reused in the children nodes, if they satisfy the constraints given by the branching. Indeed, when at a certain node a fractional solution is found for the continuous relaxation, and branching to a variable - say $i$ - is performed, all the columns with $i^{th}$ component equal to 1 can be stored for the left child and all the columns with $i^{th}$ component equal to 0 can be stored for the right child. In this way, we can have an initial set of extreme columns for every node of the B&B tree. This makes us able to *warmstart* the SD algorithm at every node, and also we can reuse a column every time it is feasible: the algorithm will never generate the same column again. Results which evidence the reduction of computational time are presented later. We note that this procedure requires storing information of several columns at each node. It is cheap with the depth first search strategy, because only two sets of columns are needed to be stored at each node (those with a component fixed to 0 and to 1), and the number of open nodes is limited. A different strategy, like breadth first search for instance, where the number of open nodes can be much higher, would be much more memory consuming. Other more refined ways of storing information given by the columns could be applied: for instance, one could project other columns generated in a parent node to the feasible set of the children node. Moreover, strategies can be found by taking into account the specific structure of the problem, based for instance on easy ways to generate feasible columns. These and other future research directions will be treated at the end of the chapter.

### 4.2.3 Lower bound and Early stopping

As mentioned also in Chapter 2, the pricing problem of SD, at each cycle, gives a valid lower bound on the solution: indeed, it solves a linearization of the original objective function, which is convex. One can alternatively see it as the dual bound given by the Dantzig-Wolfe decomposition method. Such a lower bound can be exploited very well for pruning the nodes by bound and it actually gives remarkable improvements. Indeed, in a certain node of the B&B tree, the dual bound provided by SD at each iteration, is valid not only for the current node, but also for every node in the subtree. Hence, if this value is larger than or equal to the current upper bound, the node can be pruned before the termination of the SD algorithm. The main advantage can be noticed by considering the typical *tailing-off* effect of column generation (see [105]): very often, indeed, already after a few iterations, the approximate solution and often also this dual bound are close to the optimal value. Then, several iterations only give little improvements to the convergence to the optimum. Hence, if the node can be pruned, performing the

complete column generation algorithm is generally not needed, and most of the times only very few iterations are sufficient. We also notice that the number of necessary SD cycles is further reduced by the warmstart technique described in the previous section.

We can hence make use of the early stopping technique which has been seen in Chapter 3, Section 3.2.1. A modification is however necessary, because of the use of the dual bound: one cannot prune a node for bound if the pricing problem has stopped with the early stopping, because this actually does not provide a valid lower bound, since the solution is not optimal. For this reason, the early stopping can be implemented only if the solution is already lower than the lower bound and, in this case, no pruning will be done.

## 4.3 Computational results

Here we give a detailed description of the computational results obtained with the SD based algorithmic framework we described in the previous sections.

Due to the specific features of the given problems, we use *Cplex* version 12.6.2 (see [97] for further details) as the baseline software in our tests.

### 4.3.1 Instances description

In our tests we consider instances related to combinatorial problems. In particular, we focus on quadratic shortest path problems.

The instances belonging to this class of problems take the following form:

$$\begin{aligned} \min\ & f(x) = x^\top Q x + c^\top x \\ \text{s. t.}\quad & \sum_{e \in \delta^+(s)} x_e = 1, \\ & \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0, \quad \forall v \neq s, t \\ & \sum_{e \in \delta^-(t)} x_e = 1. \end{aligned} \tag{4.2}$$

with $c \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n \times n}$. $s, t$ are the source and termination nodes, respectively; $\delta^+(v)$ are the outgoing arcs and $\delta^-(v)$ are the incoming arcs in the node $v$.

The directed graphs used in the experiments are related to two different kind of problems:

1. Grid shortest path problem, that is graphs represented by a squared grid;
2. Random shortest path problem, that is randomly generated graphs (obtained by the generator ch9-1-1 used in the 9th DIMACS implementation challenge [48]).

The benchmark consists of 12 instances based on grid graphs and 72 instances based on random graphs.

### 4.3.2 Numerical results

In the following two tables, average results are shown for the quadratic shortest path problem, for the grid and the randomly generated graphs, respectively. We compared the following algorithms: *CPLEX*; *SD*, the Simplicial Branch and Bound with no particular features; *SD-e*, with the early stopping, *SD-c* with the reuse of feasible columns in both the left and the right children, *SD-c-e*, with both the early stopping and the column reuse.

The tables represent, in average, the computational time (in seconds) for solving the instance, the total number of Branch and Bound nodes, the number of nodes necessary to find the optimal solution, the total time (in seconds) spent for solving all the master problems and all the pricing problems.

| Algorithm | Time (s) | N nodes | N nodes opt. | Time Master (s) | Time Pricing (s) |
|---|---|---|---|---|---|
| Cplex | 1036.8 | 163997 | 94633 | | |
| BBSD | 575.1 | 265192 | 19848 | 147.3 | 350.2 |
| BBSD-e | 564.6 | 265196 | 19845 | 146.1 | 341.9 |
| BBSD-c | 560.9 | 265192 | 19845 | 142.9 | 354.8 |
| BSD-c-e | 613.2 | 265194 | 19845 | 155.2 | 389.2 |

Table 4.1 – Average data for the Quadratic Shortest Path Problem, grid graphs.

Some comments have to be noticed about these results. First of all, all the algorithms solve the problem to optimality, within the imposed timelimit of 7500 seconds. Then, the combination of the depth first branch and bound with Simplicial Decomposition is effective, since the computational time of this algorithm is always better than that of the state-of-the-art software *CPLEX*. With respect to the grid graphs, the total number of nodes is higher with SD than with Cplex, but the optimum is reached earlier, thanks to the search strategy. Between master and pricing, the pricing is the most time consuming part, coherently to the results of the continuous case in Chapter 3. Solving the pricing is generally two to three times longer than solving the master, for both types of instances. More deeply, if we compare the different options for the Simplicial branch and bound, we can note that storing the columns at each node for the left and right children reduces the computational time, as expected, in particular the time of the pricing problem is reduced. And the early stopping strategy, even if adapted to this framework, helps only in the case of "no storing" of the columns. The problems with random graphs are easier, hence the total number of nodes is much smaller than with grid graphs. Nevertheless, the ratio

| Algorithm | Time (s) | N nodes | N nodes opt. | Time Master (s) | Time Pricing (s) |
|---|---|---|---|---|---|
| Cplex | 138.8 | 194 | 5 | | |
| BBSD | 12.2 | 163 | 31 | 2.9 | 8.3 |
| BBSD-e | 13.9 | 163 | 31 | 4.6 | 8.3 |
| BBSD-c | 11.1 | 163 | 31 | 2.5 | 7.2 |
| BBSD-c-e | 11.4 | 163 | 31 | 2.9 | 7.0 |

Table 4.2 – Average data for the Quadratic Shortest Path Problem, random graphs.

between the time spent for solving the master and the pricing problems is substantially unchanged.

## 4.4 Conclusions

We presented an efficient combination of the SD framework with a branch and bound scheme, to solve mixed binary convex quadratic problems. It embeds in its structure the ad-hoc method for solving the master problem, namely the adaptive conjugate directions based method, and the early stopping strategy for the pricing. It exploits all the advantages of the SD algorithm, as the lower (or dual) bound and the warmstart given by the structure of the simplices in an efficient way. We showed, through a numerical experience, that our algorithm is better than *Cplex* when dealing with particular instances with a dense Hessian matrix and with a number of constraints considerably smaller than the number of variables.

In conclusion, we showed how the SD algorithm, originally designed for continuous problems, can be profitably embedded in a framework for mixed binary quadratic problems. We trust that, with suited branching techniques, or appropriate cuts, different branching rules or node searches, we can provide a powerful tool for solving complicated mixed integer quadratic convex problems.

## 4.5 Future research directions

As a possible development of this algorithm, new branching rules and search strategies shall be tested. These variants lead to different B&B trees and could be combined with heuristics to obtain upper bounds.

Moreover, the sets of instances should be increased, by including instances from the QPLIB library (see [71]), or from other types of problems: the quadratic multidimensional Knapsack Problem, or the cardinality constrained, Quadratic Knapsack Problem, or even mixed binary portfolio problems with cardinality constraints.

Additional tools shall also be tested: one is the introduction of cuts based on the same idea of the *shrinking cuts* presented in Chapter 3. More specifically, whenever an integer vector $x_k$ is obtained in the continuous relaxation at a certain node, the following cut can be added to the pricing problem:

$$\nabla f(x_k)^\top (x - x_k) \leq 0. \tag{4.3}$$

Indeed, the optimal integer point shall satisfy this constraint, because a descent direction with respect to the previous feasible points shall exist. Other points that do not satisfy this constraint shall be computed and pruned by bound. If we add these *gradient cuts*, before generating them and solving the continuous relaxation, the corresponding nodes become infeasible, hence they are immediately pruned for infeasibility: the corresponding problems are not generated so an improvement in the algorithm could be achieved.

Moreover, the columns which are stored could be even better exploited: as noticed above in Section 4.2.2, the columns generated in the father node which are infeasible for

the children can be projected to the feasible space. While a strategy to do this projection is not straightforward or computationally efficient in general, in some cases it is actually trivial: for instance, in a quadratic knapsack problem, every column with a fractional variable $x_i$ can be projected to the subspace $x_i = 0$ trivially, without changing any other coefficient. On the other side, they can be projected on the subspace $x_i = 1$ by suitably changing one or a few other coefficients. The same idea could be generalized for other types of problems, for instance the quadratic multidimensional knapsack problems.

As another possible improvement of this algorithm, one could try to project the conjugate directions generated at a single node to the subspaces of the two children. This could result in both avoiding to generate new directions from scratch, and also, in some cases, in obtaining the optimum of the children by projection from the optimum of the father; so, without solving the continuous relaxation, but exploiting the properties of the Conjugate Direction method. In this way, the number of SD cycles could be substantially reduced.

Another direction is the improvement of the lower bounds of the continuous relaxations. That could be done by the use of ellipsoids, as explained in [25]. Indeed, the general idea is to center a given ellipsoid $E$ in the fractional optimal point and to compute the value $\lambda$ such that the scaled ellipsoid $\lambda E$ contains at least one integer point on its border and no integer point in its interior. This can be done quickly if E is chosen appropriately. Then, the minimum of the objective function $f$ over the border of $\lambda E$ yields an improved lower bound on $f$.

**Conclusion and connection to part II** What has been presented so far concludes our study on Simplicial Decomposition algorithm applied to quadratic programs. Actually, the main limit of SD is that it cannot solve problems that are not convex. In next part, we consider a class of binary quadratic problems, with other characteristics: quadratic functions can appear also in the constraints and any hypothesis about convexity is made. Hence, the problems are more general and more difficult to solve. For this class of problems we extend the formulation and propose a relaxation technique based on Dantzig-Wolfe decomposition. It turns out to be a relaxation in the BQP polytope, strictly contained in the so-called *Completely Positive* cone, and properties of this particular matrix cone are investigated. We will also need to recall some further theory about cone programming and the matrix completion problem, since we will present a result on matrix completion on the boolean quadric polytope. In the next part, two chapters show the most important results that we obtained. We will then conclude the thesis with a final chapter, which will summarize the results obtained and the future directions which could be taken.

# Part II

# Decomposition on matrices

# Chapter 5

# Matrix generation algorithms for binary quadratically constrained quadratic problems

In this chapter we use the term *matrix generation*, meaning that we adapt the approach of column generation to a formulation in the so-called *lifted space*, where the extreme points are expressed in matrix form. We address purely binary problems, with quadratic objective function and constraints. We start with the description of the formulation, then we propose a relaxation and an algorithm to solve it. We then describe some specific cases and we provide some computational results. This model will serve as a reference for the developments which will follow in Chapter 6.

## 5.1 Formulation

A generic Binary Quadratically Constrained, Quadratic Problem (BQCQP) can be written in the following form:

$$\begin{aligned} \min\ & f(x) && \text{(5.1a)}\\ \text{s. t. }\ & g_i(x) \leq 0 \quad \forall i = 1, \dots m, && \text{(5.1b)}\\ & x \in \{0,1\}^n, && \text{(5.1c)} \end{aligned}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$ is the objective function on the variables $x$, and $\forall i = 1, \dots, m$, $g_i\colon \mathbb{R}^n \to \mathbb{R}$ are the constraint functions. Both $f$ and $g_i, \forall i = 1, \dots, m$ are quadratic functions: more specifically,

$$\begin{aligned} f(x) &= x^\top Q x + q^\top x + q_0,\\ g_i(x) &= x^\top A_i x + a_i^\top x + \bar{a}_i, \quad \forall i = 1, \dots, m. \end{aligned}$$

$Q, A_i \in \mathcal{S}^n$ are symmetric matrices, $q, a_i \in \mathbb{R}^n$ are the linear coefficients and $q_0, \bar{a}_i \in \mathbb{R}$ are constant terms. We define $b_i = -\bar{a}_i$. Without loss of generality, we can assume $q_0 = 0$.

Moreover, since from constraint (5.1c), $\forall j = 1, \ldots, n$, $x_j^2 = x_j$, then all the linear terms can be omitted, because they can be added to the diagonal of the corresponding quadratic matrix. Hence, from now on we consider the following formulation:

$$\begin{aligned} \min\ & x^\top Q x && (5.2a)\\ \text{s. t. }\ & x^\top A_i x \le b_i \quad \forall i = 1, \ldots m, && (5.2b)\\ & x \in \{0,1\}^n. && (5.2c) \end{aligned}$$

We remark that no further assumptions on the matrices are required. In particular, the continuous relaxation of the problem can be non convex: i.e., we do not require $Q$ and $A_i$, $i = 1, \ldots, m$ to be positive semidefinite.

As usual, this can be rewritten in matrix form; to this aim we make use of the Hilbert product $\langle A, B \rangle$ and we introduce the matrix variable $X \in \mathbb{R}^{n \times n}$ to represent all products of the original variables: $X_{ij} = x_i x_j$, $\forall i, j = 1, \ldots, n$.

Now, we can rewrite the original problem in this equivalent way:

$$\begin{aligned} \min\ & \langle Q, X \rangle && (5.3a)\\ \text{s. t. } & \langle A_i, X \rangle \le b_i, \qquad \forall i = 1 \ldots, m && (5.3b)\\ & X = x x^\top && (5.3c)\\ & x \in \{0,1\}^n. && (5.3d) \end{aligned}$$

*Remark* 5.1. The constraint (5.3c) is a non convex constraint.

A typical way to solve a problem of this form is to do a branch and bound and to solve at each node the SDP relaxation of the problem, using an SDP solver. Instead of doing this, we try to provide a better bound by solving a different relaxation.

## 5.2 The BQP relaxation for BQCQPs

We intend to replace the constraint (5.3c) in the original formulation (5.3) by letting the matrix $X$ be a convex combination of rk-1 matrices $X_p$ of the type:

$$X_p = x_p x_p^\top,\ x_p \in \{0,1\}^n. \tag{5.4}$$

The problem, then, takes the following form:

$$\begin{aligned} \min\ & \langle Q, X \rangle && (5.5a)\\ \text{s. t. } & \langle A_i, X \rangle \le b_i, \qquad \forall i = 1 \ldots, m && (5.5b)\\ & X = \sum_{p \in \mathcal{P}} x_p x_p^\top \lambda_p && (5.5c)\\ & \sum_{p \in \mathcal{P}} \lambda_p = 1 && (5.5d)\\ & \lambda_p \ge 0 \qquad \forall p \in \mathcal{P}, && (5.5e) \end{aligned}$$

where $x_p \in \{0,1\}^n$ are binary vectors in $\mathbb{R}^n$ and $\mathcal{P}$ is the index set of all the possible extreme points $x_p$:

$$\mathcal{P} = \{p \in \mathbb{N} | \, x_p \in \{0,1\}^n\}. \tag{5.6}$$

*Remark* 5.2. $\mathcal{P}$ is a finite set, but exponentially large with respect to $n$: since the points $x_p$ are binary, $|\mathcal{P}| = 2^n$.

**Proposition 5.1.** *The problem* (5.5) *is a relaxation of* (5.2)*, and its domain is the n-dimensional Boolean Quadric Polytope* $BQP^n$*.*

*Proof.* It is a relaxation because all the solutions of the original problem (5.2) (or equivalently (5.3)) are achieved by our reformulation with $\lambda \in \{0,1\}^{|\mathcal{P}|}$, that is in the case when only one extreme point is considered (due to the constraint (5.5d)).

Then, by the constraints (5.5c) and the constraints on the $\lambda$ variables, we know that a solution $X$ of (5.5) is a convex combination of rk-1 binary matrices $X_p$ given by $X_p = x_p x_p^\top$, where $x_p$ is binary, hence by Definition 1.20 it is in $BQP^n$. □

*Remark* 5.3. Since every point in the BQP polytope is a convex combination of a finite set of doubly non negative matrices, their convex combinations are clearly in the completely positive cone. Indeed, if we define: $b_p := \sqrt{\lambda_p} x_p$, then constraint (5.5c) becomes:

$$X = \sum_{p \in \mathcal{P}} b_p b_p^\top,$$

which implies that $X \in \mathcal{C}^*$, as noticed in Remark 1.9. However, BQP is a bounded polyhedron, but CPP is a non polyhedral cone. Hence BQP is strictly contained in CPP.

*Remark* 5.4. Since the extreme points $x_p$ are binary, the following condition holds:

$$X_{jj} = \sum_{p \in \mathcal{P}} \lambda_p [(x_p)_j]^2 = \sum_{p \in \mathcal{P}} \lambda_p (x_p)_j \quad \forall j = 1, \ldots, n. \tag{5.7}$$

*Remark* 5.5. Since the domain of (5.5) is strictly contained in the CPP cone, this relaxation is stronger than the CPP relaxation (stronger than the SDP relaxation too, since the SDP cone contains CPP).

We would like to point out that, although stronger than completely positive and semidefinite relaxations, our relaxation is difficult to solve, mainly because the number of extreme points is exponentially large. In the following sections we present a way to solve this relaxation with column generation. However, we dedicate Chapter 6 to the description of a more efficient formulation of the problem when a block structure is present.

## 5.3 Solving the BQP relaxation with Dantzig-Wolfe decomposition

Formulation (5.5) expresses the constraint $X \in BQP^n$ as the convex combination of its extreme points, thus our problem is in the Dantzig-Wolfe form and we can solve it with

a column generation method. We consider a subset $\bar{\mathcal{P}} \subset \mathcal{P}$ and we solve a master program where the extreme points belong to the set $\{x_p x_p^\top \,|\, p \in \bar{\mathcal{P}}\}$. If we solve it to optimality, we find a feasible solution to our problem. Then we generate a pricing program that adds a suitable column to the master, in order to decrease the objective function, and restart by solving the new master program. If no such column can be found, then the algorithm terminates and our feasible solution is optimal for the original problem. Since the total number of columns is finite, the convergence is guaranteed.

Given the extreme points $\{x_p x_p^\top \,|\, p \in \bar{\mathcal{P}}\}$, the restricted master program (RMP) is the following:

$$\min \ \langle Q, X \rangle \tag{5.8a}$$
$$\text{s. t.} \ \langle A_i, X \rangle \leq b_i, \quad \forall i = 1 \ldots, m \tag{5.8b}$$
$$X = \sum_{p \in \bar{\mathcal{P}}} x_p x_p^\top \lambda_p \tag{5.8c}$$
$$\sum_{p \in \bar{\mathcal{P}}} \lambda_p = 1 \tag{5.8d}$$
$$\lambda_p \geq 0 \quad \forall p \in \bar{\mathcal{P}}. \tag{5.8e}$$

In this way, we can replace the constraints (5.8c) in the objective function and in the other constraints, thus obtaining a linear problem in the only $\lambda$ variables. We also introduce the notation $X_p := x_p x_p^\top$. Hence we have:

$$\min \ \sum_{p \in \bar{\mathcal{P}}} \langle Q, X_p \rangle \lambda_p \tag{5.9a}$$
$$\text{s. t.} \ \sum_{p \in \bar{\mathcal{P}}} \langle A_i, X_p \rangle \lambda_p \leq b_i, \quad \forall i = 1 \ldots, m \qquad [\rho] \tag{5.9b}$$
$$\sum_{p \in \bar{\mathcal{P}}} \lambda_p = 1 \qquad [\pi_0] \tag{5.9c}$$
$$\lambda_p \geq 0 \quad \forall p \in \bar{\mathcal{P}}. \tag{5.9d}$$

We write its dual problem. We consider the latter (more compact) formulation, where the dual variables are indicated in square brackets: let $\rho \in \mathbb{R}^m$ be the dual variables corresponding to the quadratic constraints (5.9b) and $\pi_0 \in \mathbb{R}$ be the dual variable corresponding to the constraint (5.9c). Then the dual is the following:

$$\max \ b^\top \rho + \pi_0 \tag{5.10a}$$
$$\text{s. t.} \ \sum_{i=1}^{m} \langle A_i, X_p \rangle \rho_i + \pi_0 \leq \langle Q, X_p \rangle, \quad \forall p \in \bar{\mathcal{P}} \tag{5.10b}$$
$$\rho \leq 0 \tag{5.10c}$$

Once the restricted master program is solved to optimality, the optimal dual variables $\rho^*, \pi_0^*$ are available as well and the constraints (5.10b) are satisfied with $\rho = \rho^*, \pi = \pi_0^*$ for every point $X_p \in \bar{\mathcal{P}}$. If these constraints are valid for every point $X_p \in \mathcal{P}$, then our primal feasible solution is feasible also for the dual of the master program, so it is optimal. Otherwise, there are points $X_p \in \mathcal{P} \setminus \bar{\mathcal{P}}$ that violate these constraints. In this case, we solve a pricing program that finds one of these points and adds it to the set $\bar{\mathcal{P}}$. In order to do so, once the master is solved and the dual variables $\rho^*, \pi_0^*$ are obtained, the pricing problem consists of finding an extreme point that minimizes the reduced cost. If the minimum is less than 0, we can add the corresponding constraint in the dual and the corresponding variable in the master, otherwise the algorithm terminates. For this reason, this column generation technique can be viewed as a separation problem in the dual space. The pricing problem takes the following form:

$$\min \ \langle Q, X \rangle - \sum_{i=1}^{m} \langle A_i, X \rangle \rho_i^* - \pi_0^* \tag{5.11a}$$

$$\text{s. t. } X = xx^\top \tag{5.11b}$$

$$x \in \{0,1\}^n \tag{5.11c}$$

and can be rewritten in vector form:

$$\min \ x^\top (Q - \sum_{i=1}^{m} \rho_i^* A_i) x - \pi_0^* \tag{5.12a}$$

$$\text{s. t. } x \in \{0,1\}^n \tag{5.12b}$$

While the master program is linear, the pricing is quadratic and it is an unconstrained binary quadratic program.

*Remark* 5.6. If the original problem is convex, that is the matrix $Q$ is positive semidefinite and so are all the matrices $A_i$, then all the pricing problems are convex as well: indeed, the matrix in its objective function is sum of positive semidefinite matrices because of the non positivity of the dual variables $\rho^*$ (5.10c).

A more detailed description of the computational issues and ad-hoc techniques for the solution of this algorithm is presented in 5.5. Computational results of this column generation procedure, showing CPU time and bounds obtained with this relaxation, will follow in Section 5.6.

In the next section, we present a specific case, where some more results can be discussed.

## 5.4 Binary QPs with linear equality constraints

We now focus on pure binary QPs, that is binary quadratic problems with linear constraints. We also restrict to the problems which have no inequalities in the constraints.

The formulation is hence the following, with the same notation as above:

$$\begin{aligned} \min \quad & x^\top Q x \\ \text{s. t.} \quad & a_i^\top x = b_i, \quad \forall i = 1, \dots, m \\ & x \in \{0,1\}^n. \end{aligned} \tag{5.13}$$

So, with the matrix notation:

$$\begin{aligned} \min \quad & \langle Q, X \rangle \\ \text{s. t.} \quad & \langle A_i, X \rangle = b_i, \quad \forall i = 1, \dots, m \\ & X = xx^\top \\ & x \in \{0,1\}^n. \end{aligned} \tag{5.14}$$

We notice that in this case the matrices $A_i$, $i = 1, \dots, n$ are diagonal matrices, with the vectors $a_i$ on the diagonal.

*Remark* 5.7. It is worth noticing that a specific case is when there are no constraints. in this case, the problem is:

$$\begin{aligned} \min \quad & \langle Q, X \rangle \\ \text{s. t.} \quad & X = xx^\top \\ & x \in \{0,1\}^n. \end{aligned} \tag{5.15}$$

If we apply the same relaxation described above, we obtain:

$$\begin{aligned} \min \quad & \langle Q, X \rangle \\ \text{s. t.} \quad & X \in BQP^n. \end{aligned} \tag{5.16}$$

The problem being linear, it has an optimal solution in a vertex, and the vertices are exactly the 0-1 rank-1 matrices. This means that the relaxation (5.16) is actually an exact reformulation of (5.15). This was already noticed in [117] and [51], and was one of the motivations which lead to the introduction of this polytope. However, if there are constraints in the formulation, this result is evidently not true, but in order to have an exact reformulation some additional constraints have to be introduced. In the next section we will show this fact, by exploiting the important result of Burer in [28].

### 5.4.1 Reinforcing the formulation

An effective way to strenghten the formulation is adding the quadratization of the linear constraints, as written, for example, in [127] and [47].

In particular, for linear equalities like:

$$a^\top x = b, \tag{5.17}$$

one can square both of the sides on the equality and obtain a new, quadratic constraint:

$$(a^\top x)^2 = b^2, \tag{5.18}$$

which can be rewritten as

$$x^\top a a^\top x = a^\top x x^\top a = \langle aa^\top, xx^\top \rangle = b^2, \tag{5.19}$$

using again the Hilbert product. This constraint can always be added to the formulation, because it is valid for the original problem and improves the relaxations.

We can add to our formulation the following valid constraints:

$$a_i^\top X a_i = b_i^2, \quad \forall i = 1, \dots m \tag{5.20}$$

By setting:

$$\tilde{A}_i := a_i a_i^\top, \tag{5.21}$$

the constraint (5.20) can be written as:

$$\langle \tilde{A}_i, X \rangle = b_i^2. \tag{5.22}$$

*Remark* 5.8. We notice that the matrices $\tilde{A}_i$, $i = 1, \dots, m$ are dense matrices.

If we compute the BQP relaxation of the problem with the addition of these constraints, we get:

$$\begin{aligned}
\min \quad & \langle Q, X \rangle && && (5.23a)\\
\text{s. t.} \quad & \langle A_i, X \rangle = b_i, && \forall i = 1, \dots m && (5.23b)\\
& \langle \tilde{A}_i, X \rangle = b_i^2, && \forall i = 1, \dots m && (5.23c)\\
& X = \sum_{p \in \mathcal{P}} X_p \lambda_p && && (5.23d)\\
& \sum_{p \in \mathcal{P}} \lambda_p = 1 && && (5.23e)\\
& \lambda_p \geq 0 && \forall p \in \mathcal{P}, && (5.23f)
\end{aligned}$$

where $X_p := x_p x_p^\top$ and $x_p \in \{0,1\}^n, \quad \forall p \in \mathcal{P}$.

*Remark* 5.9. The formulation (5.23) remains a relaxation of the original formulation (5.14), because we added valid constraints.

As we already noticed in Section 2.12, in [28] Burer proves that every mixed binary problem with linear constraints can be rewritten in CPP form. In particular, this holds for problems of the form (5.13). More specifically, he proved that the CPP relaxation of the problem is a reformulation of the original one, in the sense that the optimal values are the same and any optimal solution in CPP form is a convex combination of optimal solutions for the original problem. The relaxation is obtained by considering the extended formulation and adding two more constraints: the quadratization of the linear constraints

and the equality between the elements in the diagonal of X and the corresponding linear terms for every binary component (see Formulations (2.43) and (2.44)). Since in this case we consider fully binary problems, we can use the notation introduced in this chapter, without considering the linear terms on the variables and supposing that we added the linear coefficients to the diagonal of the matrices of the objective function and of the constraints. Considering formulation (5.14), the equivalent formulation is hence the following:

$$\begin{aligned} &\min \langle Q, X\rangle && && (5.24a)\\ &\text{s. t.} \quad \langle A_i, X\rangle = b_i, && \forall i = 1\ldots, m && (5.24b)\\ &\qquad \langle \tilde{A}_i, X\rangle = b_i^2, && \forall i = 1\ldots, m && (5.24c)\\ &\qquad X \in \mathcal{C}^{*n}, && && (5.24d) \end{aligned}$$

where we used the notation (5.21). Burer proved that the problem (5.24) is equivalent to (5.13). We now show that the BQP relaxation satisfies (5.24).

**Proposition 5.2.** *The BQP relaxation* (5.23) *satisfies* (5.24).

*Proof.* If a point $X$ satisfies (5.23), then, the first two constraints of (5.24) are satisfied since they appear also in (5.23). And the constraints (5.23d)-(5.23f) together imply (5.24d), as noticed in Remark (5.3). □

As an easy consequence, it follows that:

**Corollary 5.1.** *The BQP relaxation* (5.23) *is equivalent to the original formulation* (5.13).

*Proof.* The previous Proposition showed that (5.24) is a relaxation for (5.23), which is a relaxation of the original problem. Now, since (5.24) is equivalent to the original formulation from the result in [28], it follows that also (5.23) is equivalent to (5.13). □

This result shows that the BQP relaxation for purely binary equality-constrained problems with quadratized constraints is equivalent to the original problem and to the CPP relaxation; its domain is a polytope and in principle it allows us to solve the problem to the optimum, without recourse to branching. However, adding quadratized constraints, which are dense, makes the problem much more difficult: several more points are needed for the convergence and the computational time dramatically increases. In our preliminary tests, the algorithm always reached the time limit of several hours without convergence.

In the next section we describe in detail some computational issues and the ways we actually implemented the algorithm. Then, some computational results are given, and finally we present the conclusions and the possible improvements and future research directions.

## 5.5 Computational aspects

With this framework, it is sufficient to solve a sequence of linear master problems and unconstrained quadratic binary pricing problems to obtain the optimal value of the BQP-relaxation of the original problem. In our fist results, we used the solver *Cplex* to solve both the master and the pricing problems. Even if this Dantzig-Wolfe approach theoretically provides a solution method for the problem we are considering, we must pay attention to the efficiency of the algorithm and to some computational aspects. First of all, the number of master and pricing problems to solve could be intractable in practice, because it depends on the number of extreme points, which is exponential in the dimension of the problem. Secondly, every pricing problem is a binary unconstrained quadratic problem, which could be difficult in general. Finally, even the master problem is not well defined at the beginning, because we do not start with a feasible solution and we have to find some initial columns. In the next sections we describe the way we solve master and pricing problems, while in the next chapter we introduce a strategy to reduce the number of extreme points.

### 5.5.1 Feasibility of the master

Initializing the master is crucial: in general, finding feasible columns from which to start the column generation algorithm is not straightforward. We use an approach described in [137]. In fact, we use artificial columns that make the first master program feasible. More specifically, we add one variable for each constraint, all with a large objective function value (a so-called Big M) and the right-hand-side value as coefficient. In this way, a feasible solution always exists, even if $\bar{\mathcal{P}} = \emptyset$: all these variables can be set to 1. Then, as soon as the algorithm provides a feasible column and so these extra variables are fixed to 0 in the optimal solution of the master, they can be dropped. For constraints with positive right-hand-side and lower inequality sense, they are unnecessary and they are immediately removed. In this way, even if the first columns provided by the pricing are not feasible, the master program finds a feasible solution, provides dual variables, reduced costs and the algorithm can go on. The additional variables of the master always have a non negative reduced cost.

If we call $y$ the artificial variables, the restricted master program has the following

form:

$$\begin{aligned}
\min & \sum_{p\in\bar{\mathcal{P}}}\langle Q, X_p\rangle\lambda_p + \sum_{j=1}^{m+1} M_j y_j && (5.25)\\
\text{s. t.} & \sum_{p\in\bar{\mathcal{P}}}\langle A_i, X_p\rangle\lambda_p + \bar{b}_i y_i \le \bar{b}_i, \quad \forall i = 1\ldots, m\\
& \sum_{p\in\bar{\mathcal{P}}}\lambda_p + y_{m+1} = 1\\
& \lambda_p \ge 0 \quad \forall p \in \bar{\mathcal{P}}.\\
& y_j \ge 0 \quad \forall j = 1, \ldots, m+1.
\end{aligned}$$

The dual of this problem is the following:

$$\begin{aligned}
& \max \bar{a}^\top \alpha + \pi && (5.26)\\
\text{s. t.} \quad & \sum_{i=1}^{m}\langle A_i, X_p\rangle\alpha_i + \pi \le \langle Q, X_p\rangle, \quad \forall p \in \bar{\mathcal{P}} && (5.27)\\
& \bar{a}_j\alpha_j \le M_j, \quad \forall j = 1\ldots, m\\
& \alpha \le 0
\end{aligned}$$

and therefore, the pricing problem is unchanged.

### 5.5.2 Early stopping of the pricing

As we already mentioned, the most demanding part is solving the pricing problem. Hence we used some strategies to solve it faster, one of them is the so-called *early stopping* for the pricing. Indeed, it is possible to stop the computations for solving this subproblem when it finds a point with negative reduced cost. More specifically, we impose that the solver stops when it finds the first binary vector that has a negative reduced cost. If the algorithms finds it, then we can add this point to the set of extreme points for the master and proceed in the column generation algorithm. If there are no such points, this means that no other extreme point can be added so the algorithm has already found the optimum and it can stop. Otherwise, if the pricing reaches its time limit without finding any such extreme point, we do not have any guarantee of optimality: we will only have a lower bound for the optimum of the BQP-relaxation. The best valid bound that we can have is given by the solver: it is the minimum among the lower bounds of the open nodes of the B&B tree and it is provided by *Cplex*.

## 5.6 Results

At first, we tested our algorithm on some instances from the QPLIB library (see [71]). In particular, we selected some of them which contain linear or quadratic (non convex) constraints and objective function, and purely binary variables.

We compare the results obtained by solving the instances with our algorithm -with the early stopping technique- and with the solver *BiqCrunch* (see [99]), which is an open source solver for binary quadratic programs that is based on the SDP. We compare the root node bound for each instance, given by both algorithms. For *BiqCrunch*, we consider three sets of parameters, in order to solve the root node relaxation. The first set (*BC-default*) is the default set of parameters. The second set (*BC-bound*) is tuned to obtain the best root node bound, with no additional inequality. The third set (*BC-cuts*) allows to find a better root node bound, with the addition of valid triangular inequalities. Then, we show the results of our BQP relaxation, with the early stopping technique. The results are collected in Table 5.1. The first column contains the names of the instances, the second one contains the optimal value, provided in the QPLIB website. Then, for each of the *BiqCrunch* set of parameters and for our algorithm, two sub-columns represent the root node bound and the time, in seconds, spent to obtain it.

| Instance | Opt val | BC-default | | BC-bound | | BC-cuts | | BQP | |
|---|---|---|---|---|---|---|---|---|---|
| | | T (s) | Bound | T (s) | Bound | T (s) | Bound | T (s) | Bound |
| QPLIB-0067 | -110942 | 9 | -112840.2 | 0 | -116485.9 | 22 | -112799 | 0 | -112356 |
| QPLIB-1976 | -9594 | 6 | -80803.2 | 27 | -51094.5 | 193 | -45143 | 7 | -44898 |
| QPLIB-2017 | -22984 | 23 | -198910.4 | 113 | -124395.9 | 114 | -124396 | 124 | -78215 |
| QPLIB-2029 | -34704 | 35 | -314457.6 | 180 | -229913.3 | 180 | -229913 | 1865 | -101334 |
| QPLIB-2036 | -30590 | 44 | -388957.4 | 220 | -257082.9 | 220 | -257083 | 185 | -126386 |
| QPLIB-2055 | 3389110 | 6 | 1948874.3 | 21 | 1999553.8 | 104 | 2209752 | 92 | 2314020 |
| QPLIB-2060 | 2528144 | 15 | 1550265.6 | 36 | 1466569.4 | 655 | 1703346 | 153 | 1707160 |
| QPLIB-2067 | 3311060 | 12 | 678968.9 | 72 | 1063652.1 | 149 | 1152450 | 242 | 1260470 |
| QPLIB-2073 | 7600750 | 24 | 6152309.4 | 57 | 6217026.1 | 1078 | 6827451 | 285 | 6834930 |
| QPLIB-2085 | 7034580 | 25 | 4568498.2 | 85 | 4705156.8 | 2642 | 5420526 | 1066 | 5432400 |
| QPLIB-2087 | 3312579 | 36 | 784483.5 | 123 | 952181.8 | 172 | 957531 | 2935 | 1442440 |
| QPLIB-2096 | 7068000 | 73 | 5914217.1 | 82 | 5826147.8 | 2679 | 6305261 | 1210 | 6312620 |
| QPLIB-2357 | -647 | 46 | -647.1 | 16 | -733.3 | 46 | -647 | 3223 | -647 |
| QPLIB-2359 | -648 | 55 | -662.0 | 74 | -718.3 | 54 | -662 | 2888 | -648 |
| QPLIB-2512 | 135028 | 8 | -193036.5 | 2 | -443431.9 | 117 | -26966 | 6 | 0 |
| QPLIB-2733 | 5358 | 225 | -6039.9 | 10 | -35469.4 | 1006 | -4190 | 19258 | -2914 |
| QPLIB-2957 | 3596 | 728 | -11613.2 | 78 | -42428.5 | 2392 | -9248 | 11261 | 0 |
| QPLIB-3307 | 1240 | 118 | -2684.9 | 5 | -8652.4 | 1044 | -1211 | 472 | 0 |
| QPLIB-3413 | 2192 | 582 | -2432.6 | 33 | -183992.2 | 678 | -2412 | 11 | 0 |
| QPLIB-3587 | 15595 | 109 | -665.3 | 5 | -107792.2 | 109 | -665 | 2 | 0 |
| QPLIB-3614 | 14409 | 115 | -0.8 | 4 | -83637.5 | 123 | 0 | 2 | 0 |
| QPLIB-3714 | 1183 | 14 | 1180.7 | 2 | -8.7 | 20 | 1181 | 1607 | 1183 |
| QPLIB-3751 | 2312 | 18 | 2308.9 | 2 | -9.7 | 20 | 2309 | 7709 | 2312 |
| QPLIB-3757 | -563 | 349 | -772.3 | 482 | -664.0 | 344 | -772 | 8850 | -563 |
| QPLIB-3762 | -296 | 1 | -296.0 | 1 | -345.7 | 2 | -296 | 1037 | -296 |
| QPLIB-3775 | 3990 | 24 | 3989.2 | 5 | -10.6 | 25 | 3990 | 32932 | 3990 |
| QPLIB-3803 | -7360 | 54 | -7360.3 | 12 | -9764.6 | 57 | -7360 | 5620 | -7360 |
| QPLIB-3815 | -65 | 36 | -69.7 | 2 | -83.8 | 41 | -69 | 1807 | -66 |
| QPLIB-6647 | 2 | 791 | -1878.2 | 1009 | -344.4 | 11271 | -1 | 232 | 0 |
| QPLIB-7127 | 0 | 1673 | -7481.1 | 646 | -351.5 | 1662 | -7481 | 2750 | 0 |

Table 5.1 – Root node bound and time for QPLIB instances.

We can see that the time spent by our code is generally lower than the time needed by *BiqCrunch* with the setting that provides the best valid lower bound, and our bound is always slightly or sensibly higher. With the default settings, *BiqCrunch*, on the instances

which are solved, is much faster but obtains significantly weaker bounds. In Table 5.2, we present the number of iterations, the dimension of the final master, (in which we deleted all the columns with zero value) the partition of time between master and pricing of our algorithm, for the same set of instances. We notice that the most relevant portion of time

| Instance | BQP bound | N. its | Final dim | Total time (s) | Time master (s) | Time pricing (s) |
|---|---|---|---|---|---|---|
| QPLIB-0067 | -112356 | 3 | 2 | 0 | 0 | 0 |
| QPLIB-1976 | -44898 | 417 | 31 | 7 | 0 | 6 |
| QPLIB-2017 | -78215 | 2816 | 35 | 124 | 21 | 86 |
| QPLIB-2029 | -101334 | 5858 | 39 | 1865 | 1578 | 239 |
| QPLIB-2036 | -126386 | 3008 | 43 | 185 | 34 | 122 |
| QPLIB-2055 | 2314020 | 703 | 58 | 92 | 34 | 57 |
| QPLIB-2060 | 1707160 | 837 | 77 | 153 | 63 | 87 |
| QPLIB-2067 | 1260470 | 1010 | 77 | 242 | 112 | 126 |
| QPLIB-2073 | 6834930 | 989 | 66 | 285 | 127 | 153 |
| QPLIB-2085 | 5432400 | 1694 | 103 | 1066 | 626 | 425 |
| QPLIB-2087 | 1442440 | 2755 | 124 | 2935 | 1826 | 1082 |
| QPLIB-2096 | 6312620 | 1821 | 67 | 1210 | 926 | 261 |
| QPLIB-2357 | -647 | 28 | 2 | 3223 | 0 | 3222 |
| QPLIB-2359 | -648 | 82 | 7 | 2888 | 0 | 2887 |
| QPLIB-2512 | 0 | 139 | 17 | 6 | 0 | 6 |
| QPLIB-2733 | -2914 | 373 | 36 | 19258 | 0 | 19248 |
| QPLIB-2957 | 0 | 582 | 44 | 11261 | 0 | 11229 |
| QPLIB-3307 | 0 | 318 | 32 | 472 | 0 | 469 |
| QPLIB-3413 | 0 | 264 | 40 | 11 | 0 | 6 |
| QPLIB-3587 | 0 | 153 | 31 | 2 | 0 | 1 |
| QPLIB-3614 | 0 | 148 | 31 | 2 | 0 | 1 |
| QPLIB-3714 | 1183 | 1529 | 3 | 1607 | 1 | 1602 |
| QPLIB-3751 | 2312 | 3197 | 3 | 7709 | 7 | 7691 |
| QPLIB-3757 | -563 | 403 | 27 | 8850 | 43 | 8796 |
| QPLIB-3762 | -296 | 61 | 1 | 1037 | 0 | 1037 |
| QPLIB-3775 | 3990 | 6373 | 3 | 32932 | 32 | 32865 |
| QPLIB-3803 | -7360 | 45 | 1 | 5620 | 0 | 5620 |
| QPLIB-3815 | -66 | 5093 | 50 | 1807 | 24 | 1766 |
| QPLIB-6647 | 0 | 283 | 33 | 232 | 0 | 220 |
| QPLIB-7127 | 0 | 589 | 51 | 2750 | 0 | 2677 |

Table 5.2 – Repartition of time between master and pricing for QPLIB instances.

is due to the pricing phase, as expected, except for some instance which need several iterations, in which also the master problems are more challenging.

We then considered one of the classes of QPLIB instances, where our algorithm seems particularly promising. We chose the set of the *sonet* instances, provided by Bonami in [21]. We selected the instances of type *nc*. We have a set consisting of 51 instances. We show the results in Table 5.3, where the notation is the same as in the previous table, except for the fact that the optimal values are not given.

The results confirm that the performances of the BQP relaxation are generally better than those of the SDP-based relaxation, even if there are a certain number of failures: 22 out of 51 instances reached the maximum number of iterations (100000) or the time limit (10 hours). The BQP bound is always higher and in several cases much higher than the SDP bound, and the CPU time is comparable.

In Table 5.4 we collect the number of iterations and the size of the final master,

the portion of time spent by our algorithm to solve the master and the pricing phases, respectively. We note with *M.I.* the instances in which the maximum number of iterations is reached, and with *T.L.* those in which the time limit is reached. We can see that the pricing time is not so high, mostly due to the early stopping technique. However, a large number of iterations is needed and, as we noticed for the QPLIB instances, when the number of iterations grows, the time spent by the master problems becomes considerably high.

Finally, we consider the Quadratic Assignment Problem (QAP), a famous linearly constrained quadratic problem which is known to be hard to solve. It originally comes from facility location applications and models the following problem. There are a set of $n$ facilities and $n$ locations. For each pair of locations, a distance is specified and for each pair of facilities a weight or flow is specified (e.g., the amount of supplies transported between the two facilities). The problem is to assign all facilities to different locations with the goal of minimizing the sum of the distances multiplied by the corresponding flows. It can be expressed as a quadratic problem with the following form:

$$\begin{aligned} \min\ & f(x) = \frac{1}{2} x^\top C x \\ \text{s. t.}\quad & a_h^\top x = 1, \quad \forall h = 1 \ldots, 2n \\ & x \in \{0,1\}^{n^2}. \end{aligned} \tag{5.28}$$

We considered only small sized instances, from the QAPlib instance (see [33]). The results for the BQP bound appear in Table 5.5. We compared our approach to the SDP bound given by BiqCrunch and we can note the following: the SDP root node bound is negative, while the solution of the problem is clearly positive. Moreover, the BQP bound is 0, so it is not tight as well, but it is computed fast and it is definitely larger than the SDP bound. To these instances we added the quadratization of the linear constraints, in order to obtain the original optimal point. However, we reached the time limit of 10 hours for all the instances except the easy *nug5*, with only 5 facilities and locations, in which we obtained the optimal value in 0.5 seconds.

## 5.7 Conclusions

In this chapter we proposed a method to solve binary QCQPs through the application of the classic DWD onto the lifted space of products of variables. We analyzed the domains of master and pricing problems in this formulation and we noticed that the decomposition is done in the Boolean Quadric Polytope, strictly contained in the *completely positive* cone. This method seems interesting for the simple forms of the two subproblems and, to the best of our knowledge, it has never been proposed; the results and comparisons with the SDP relaxation are encouraging, at least for the root node bounds that this algorithm can obtain. A relation with the CPP reformulation of quadratic problems is shown, which is interesting from the theoretical point of view.

In conclusion, this method seems promising: it could be an useful tool to help in the solution of binary QCQPs, which are among the most difficult quadratic problems.

## 5.8 Future research directions

The study of this algorithm could be improved in several ways: more tests on BQCQP instances could be done and more efficient computational techniques should be implemented. As one of the possible improvements, the pricing problems should be solved more efficiently: it can be seen as a max-cut problem, and there are in literature some solvers suited for this class of problems: we could use BiqCrunch itself, or BiqMac (see [122]), for instance. More refined ways of implementing an early stopping, or other warmstart strategies for the pricing could be introduced. Also some techniques similar to those used for Bundle methods (see [133, 134]) shall be considered.

In addition, more tests on some literature instances for linearly equality-constrained BQP relaxations with quadratized constraints could be done: they could confirm in practice the theoretical results that our BQP relaxation can directly lead to the binary optimum. Alternatively, this approach could be used in a B&B scheme, analogously as what we proposed for convex problems in the first part of this thesis.

Then, the same important theoretical result could be extended to the case of problems with inequalities and mixed binary problems. Indeed, there are a few issues due to the bounds of the non integer variables in the pricing that shall be addressed, but stronger formulations can surely be obtained.

Nevertheless, we shall note that the main difficulties of this approach are structural and cannot be avoided. Instead, we could try to address structured problems, where a possibly simpler decomposition could be made. Since the most difficult issues are the huge number of extreme points, and the hardness of the pricing, the idea is to select problems with a sparse structure, and develop an adaptation of this method which can be computed on smaller subproblems. This is the main reason which lead us to develop the next Chapter, where we show how such an adaptation can be constructed, along with the main benefits and the main issues.

| Instance | BC-default | | BC-bound | | BC-cuts | | BQP | |
|---|---|---|---|---|---|---|---|---|
| | T (s) | Bound | T (s) | Bound | T (s) | Bound | T (s) | Bound |
| gr17-nc-qc.lp | 6 | -19364 | 38 | 15803 | 242 | 21831 | 8 | 21973 |
| ins.16.v1-nc-qc.lp | 5 | -524839 | 21 | -387987 | 22 | -387987 | 703 | 6636 |
| ins.16.v2-nc-qc.lp | 4 | -1194454 | 21 | -157176 | 21 | -157176 | 169 | 75322 |
| ins.16.v3-nc-qc.lp | 4 | -7048 | 64 | 929 | 120 | 1571 | 277 | 1596 |
| ins.16.v5-nc-qc.lp | 4 | -21920 | 28 | 1872 | 169 | 3120 | 187 | 3193 |
| ins.16.v6-nc-qc.lp | 4 | -30990 | 44 | 2119 | 113 | 3912 | 185 | 3990 |
| ins.17.v1-nc-qc.lp | 6 | -1819024 | 27 | -795830 | 28 | -795830 | - | - |
| ins.17.v2-nc-qc.lp | 6 | -991482 | 28 | -174478 | 28 | -174478 | 22 | 138612 |
| ins.17.v3-nc-qc.lp | 6 | -10478 | 22 | 2677 | 300 | 4026 | 15 | 4117 |
| ins.17.v5-nc-qc.lp | 6 | -8058 | 37 | 11000 | 227 | 16400 | 7 | 16526 |
| ins.17.v6-nc-qc.lp | 6 | -17380 | 26 | 12061 | 273 | 18300 | 7 | 18452 |
| ins.18.v1-nc-qc.lp | 8 | -1797546 | 36 | -384592 | 36 | -384592 | 920 | 909 |
| ins.18.v2-nc-qc.lp | 8 | -428969 | 39 | -117439 | 38 | -117439 | 1592 | 14816 |
| ins.18.v3-nc-qc.lp | 8 | -16701 | 80 | 997 | 218 | 1740 | 2544 | 1758 |
| ins.18.v5-nc-qc.lp | 8 | -63709 | 39 | -108 | 39 | -108 | 876 | 3516 |
| ins.18.v6-nc-qc.lp | 8 | -68397 | 38 | -1453 | 38 | -1453 | 1076 | 4394 |
| ins.19.v1-nc-qc.lp | 10 | -1251012 | 52 | -368130 | 52 | -368130 | - | - |
| ins.19.v2-nc-qc.lp | 10 | -1868551 | 52 | -1034144 | 51 | -1034144 | - | - |
| ins.19.v3-nc-qc.lp | 11 | -23488 | 55 | -2946 | 55 | -2946 | 701 | 1906 |
| ins.19.v5-nc-qc.lp | 11 | -83125 | 54 | -17124 | 54 | -17124 | 1651 | 3811 |
| ins.19.v6-nc-qc.lp | 11 | -123663 | 53 | -20670 | 53 | -20670 | 1724 | 4764 |
| ins.20.v1-nc-qc.lp | 14 | -3724644 | 64 | -996229 | 64 | -996229 | 500 | 211908 |
| ins.20.v2-nc-qc.lp | 13 | -1828616 | 67 | -571504 | 67 | -571504 | - | - |
| ins.20.v3-nc-qc.lp | 15 | -31199 | 69 | -1158 | 69 | -1158 | - | - |
| ins.20.v5-nc-qc.lp | 14 | -87227 | 67 | -1990 | 68 | -1990 | - | - |
| ins.20.v6-nc-qc.lp | 14 | -89954 | 68 | -12886 | 69 | -12886 | - | - |
| ins.21.v1-nc-qc.lp | 19 | -6874242 | 83 | -4910836 | 83 | -4910836 | - | - |
| ins.21.v2-nc-qc.lp | 17 | -2229906 | 85 | -1539431 | 84 | -1539431 | - | - |
| ins.21.v3-nc-qc.lp | 17 | -88542 | 86 | -28206 | 84 | -28206 | - | - |
| ins.21.v5-nc-qc.lp | 18 | -163943 | 86 | -12247 | 88 | -12247 | - | - |
| ins.21.v6-nc-qc.lp | 18 | -153072 | 89 | 2980 | 88 | 2980 | 128 | 20279 |
| ins.22.v1-nc-qc.lp | 23 | -8168025 | 102 | -7062597 | 103 | -7062597 | - | - |
| ins.22.v2-nc-qc.lp | 22 | -3547701 | 105 | -2995516 | 105 | -2995516 | - | - |
| ins.22.v3-nc-qc.lp | 23 | -32977 | 113 | 1548 | 113 | 1548 | 187 | 7033 |
| ins.22.v5-nc-qc.lp | 23 | -97889 | 112 | 1821 | 112 | 1821 | 197 | 14065 |
| ins.22.v6-nc-qc.lp | 22 | -162126 | 112 | 4360 | 111 | 4360 | 842 | 17581 |
| ins.23.v1-nc-qc.lp | 27 | -4545427 | 136 | 256040 | 137 | 256040 | - | - |
| ins.23.v2-nc-qc.lp | 27 | -849716 | 132 | 516197 | 141 | 516197 | - | - |
| ins.23.v3-nc-qc.lp | 29 | -79394 | 137 | -33822 | 137 | -33822 | 25710 | 2971 |
| ins.23.v5-nc-qc.lp | 29 | -244628 | 138 | -70162 | 139 | -70162 | - | - |
| ins.23.v6-nc-qc.lp | 28 | -341041 | 141 | -107088 | 141 | -107088 | - | - |
| ins.24.v1-nc-qc.lp | 34 | -7038710 | 164 | -5722660 | 159 | -5722660 | - | - |
| ins.24.v2-nc-qc.lp | 35 | -5013345 | 162 | -2777104 | 162 | -2777104 | - | - |
| ins.24.v3-nc-qc.lp | 37 | -150047 | 181 | -52206 | 181 | -52206 | 486 | 6289 |
| ins.24.v5-nc-qc.lp | 34 | -325428 | 176 | -137132 | 176 | -137132 | - | - |
| ins.24.v6-nc-qc.lp | 34 | -406983 | 175 | -185316 | 175 | -185316 | 2292 | 15722 |
| ins.25.v1-nc-qc.lp | 42 | -5272983 | 205 | -3859934 | 205 | -3859934 | - | - |
| ins.25.v2-nc-qc.lp | 42 | -2362254 | 207 | -1251807 | 207 | -1251807 | - | - |
| ins.25.v3-nc-qc.lp | 44 | -110194 | 217 | -9687 | 202 | -9687 | 163 | 11758 |
| ins.25.v5-nc-qc.lp | 41 | -257649 | 205 | -30970 | 216 | -30970 | 316 | 23516 |
| ins.25.v6-nc-qc.lp | 40 | -284938 | 207 | -62888 | 190 | -62888 | 231 | 29394 |

Table 5.3 – Root node bound and time for *sonet* instances.

| Instance | BQP bound | N. its | Final dim | Total time (s) | Time master (s) | Time pricing (s) |
|---|---|---|---|---|---|---|
| gr17-nc-qc.lp | 21973 | 542 | 31 | 8 | 1 | 7 |
| ins.16.v1-nc-qc.lp | 6636 | 8658 | 7 | 703 | 569 | 131 |
| ins.16.v2-nc-qc.lp | 75322 | 4064 | 6 | 169 | 104 | 63 |
| ins.16.v3-nc-qc.lp | 1596 | 4555 | 25 | 277 | 212 | 63 |
| ins.16.v5-nc-qc.lp | 3193 | 4011 | 23 | 187 | 122 | 64 |
| ins.16.v6-nc-qc.lp | 3990 | 3755 | 23 | 185 | 135 | 49 |
| ins.17.v1-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.17.v2-nc-qc.lp | 138612 | 1191 | 19 | 22 | 2 | 19 |
| ins.17.v3-nc-qc.lp | 4117 | 995 | 23 | 15 | 1 | 13 |
| ins.17.v5-nc-qc.lp | 16526 | 492 | 29 | 7 | 0 | 6 |
| ins.17.v6-nc-qc.lp | 18452 | 529 | 29 | 7 | 1 | 7 |
| ins.18.v1-nc-qc.lp | 909 | 3901 | 2 | 920 | 856 | 62 |
| ins.18.v2-nc-qc.lp | 14816 | 7281 | 9 | 1592 | 1440 | 148 |
| ins.18.v3-nc-qc.lp | 1758 | 5498 | 4 | 2544 | 2437 | 104 |
| ins.18.v5-nc-qc.lp | 3516 | 7430 | 5 | 876 | 725 | 147 |
| ins.18.v6-nc-qc.lp | 4394 | 6850 | 5 | 1076 | 940 | 133 |
| ins.19.v1-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.19.v2-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.19.v3-nc-qc.lp | 1906 | 5322 | 3 | 701 | 575 | 122 |
| ins.19.v5-nc-qc.lp | 3811 | 6637 | 5 | 1651 | 1499 | 148 |
| ins.19.v6-nc-qc.lp | 4764 | 8601 | 5 | 1724 | 1514 | 205 |
| ins.20.v1-nc-qc.lp | 211908 | 5640 | 15 | 500 | 340 | 157 |
| ins.20.v2-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.20.v3-nc-qc.lp | T.L. | - | - | - | - | - |
| ins.20.v5-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.20.v6-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.21.v1-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.21.v2-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.21.v3-nc-qc.lp | T.L. | - | - | - | - | - |
| ins.21.v5-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.21.v6-nc-qc.lp | 20279 | 3426 | 34 | 128 | 37 | 89 |
| ins.22.v1-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.22.v2-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.22.v3-nc-qc.lp | 7033 | 3960 | 35 | 187 | 51 | 132 |
| ins.22.v5-nc-qc.lp | 14065 | 3018 | 35 | 197 | 104 | 91 |
| ins.22.v6-nc-qc.lp | 17581 | 5009 | 35 | 842 | 687 | 151 |
| ins.23.v1-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.23.v2-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.23.v3-nc-qc.lp | 2971 | 19828 | 11 | 25710 | 24888 | 801 |
| ins.23.v5-nc-qc.lp | T.L. | - | - | - | - | - |
| ins.23.v6-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.24.v1-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.24.v2-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.24.v3-nc-qc.lp | 6289 | 6025 | 39 | 486 | 248 | 231 |
| ins.24.v5-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.24.v6-nc-qc.lp | 15722 | 6826 | 39 | 2292 | 2020 | 264 |
| ins.25.v1-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.25.v2-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.25.v3-nc-qc.lp | 11758 | 3240 | 43 | 163 | 33 | 125 |
| ins.25.v5-nc-qc.lp | 23516 | 4749 | 43 | 316 | 111 | 199 |
| ins.25.v6-nc-qc.lp | 29394 | 4109 | 43 | 231 | 61 | 165 |

Table 5.4 – Repartition of time between master and pricing for Sonet instances.

| Instance | Opt val | BC-bound | | BQP | |
|---|---|---|---|---|---|
| | | Bound | T (s) | Bound | T (s) |
| nug5 | 50 | -108.5 | 0.04 | 0 | 0.04 |
| nug12 | 578 | -3275 | 3.48 | 0 | 1.9 |
| lipa10a | 3683 | -5318 | 0.71 | 0 | 21.7 |
| chr12a | 9552 | -415541 | 5.87 | 0 | 0.4 |
| chr12b | 9742 | -415950 | 8.14 | 0 | 0.8 |
| chr12c | 11156 | -414401 | 11.18 | 0 | 0.24 |

Table 5.5 – Root node bound and time for QAP instances.

# Chapter 6

# Block-BQP decomposition

In this chapter we consider sparse problems in which the elements of the matrices can be decomposed into blocks. The motivations are that in this way we can use a similar approach to that of the last chapter, exploiting its good properties, but simplifying the drawbacks that we evidenced in the previous analysis. Moreover, in literature there exists several problems which are sparse and present this structure. For instance, instances from several domains reported in the Minlplib library (a library that collects mixed integer nonlinear problems among which several quadratic problems, see [34]) are quite sparse, as reported in [116]. In addition, to the best of our knowledge the theoretical results which we prove here have never been proposed before. We start with some definitions.

**Definition 6.1.** *We define the* support *of a matrix* $M \in \mathbb{R}^{m \times n}$ *as*

$$\text{Supp}(M) := \{(p, q) \in \{1, \dots, m\} \times \{1, \dots, n\} \mid M_{pq} \neq 0\}.$$

**Definition 6.2.** *Given* $n \in \mathbb{N}$ *and* $k \leq n$*, we define a* k-(Coordinate) block sequence *in* $\mathbb{R}^n$ *the set* $\{\underline{b_1}, \dots, \underline{b_k}\}$*, where the blocks* $\underline{b_j} \subseteq \{1, \dots, n\}$ $\forall j = 1, \dots, k$ *and:*

$$\bigcup_{j=1}^{k} \underline{b_j} = \{1, \dots, n\}.$$

*We indicate with* $d_j = |\underline{b_j}|$ *the dimension of each subset.*

*Remark* 6.1. We also assume that no block is a subset of one other and that they are sorted according to the order of their first element.

**Definition 6.3.** *Given a k-block sequence, we define the* Matrix blocks*:*

$$\underline{B_j} := \underline{b_j} \times \underline{b_j} \subseteq \{1, \dots, n\}^2 \quad \forall j = 1, \dots, k.$$

*We define the set* $\underline{\mathcal{B}_k} := \bigcup_{j=1}^{k} \underline{B_j}$ *as a* Block structure *in* $\mathbb{R}^{n \times n}$*.*

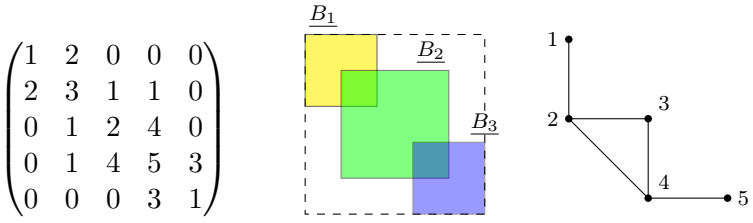
**Definition 6.4.** *The* Sparsity graph *of a block structure in* $\mathbb{R}^{n \times n}$ *is a graph* $G(V, E)$ *with* $V = \{1, \dots, n\}$ *and with edges on the block structure:* $\{p, q\} \in E$ *if and only if* $\{p, q\} \subseteq \underline{\mathcal{B}_k}$*.*

*Remark* 6.2. It follows from the definition that every subgraph of the sparsity graph induced by the vertices of a block is complete. Hence, the sparsity graph of a block structure is given by the union of cliques.

**Definition 6.5.** *Given a matrix $M \in \mathbb{R}^{n\times n}$ and a block structure $\underline{\mathcal{B}_k}$, $k \leq n$, we say that $M$ is* block-decomposable under $\underline{\mathcal{B}_k}$ *if:*

$$\mathrm{Supp}(M) \subseteq \underline{\mathcal{B}_k}.$$

In Figure 6.1 an example is provided, with $n = 5$: a matrix, its block structure with its sparsity graph are shown.




(a) Matrix (b) Block structure (c) Sparsity graph

Figure 6.1 – Example of a block-decomposable matrix

In general, several block structures could be used for a single matrix. For instance, every matrix is trivially decomposable in a structure consisting of just one n-dimensional block. We are interested in structures with $k > 1$ blocks and potentially we seek a large value of $k$. This means that the nonzero entries of the matrix fit in (possibly overlapping) smaller blocks. A special case is when the blocks are disjoint:

**Definition 6.6.** *If all the blocks in $\underline{\mathcal{B}_k}$ have pairwise empty intersection (that is $\{\underline{b_j}\}_{j=1,\dots,k}$ is a partition of $\{1,\dots,n\}$) then the matrix is called* block-separable.

The largest possible number of $k$ is $n$: if this holds, all the matrices are diagonal and the problem can be formulated as a binary linear problem.

**Definition 6.7.** *A quadratic problem of the form* (5.2) *is called* block-decomposable *(resp.* block-separable*) if there exists a common block structure $\underline{\mathcal{B}_k}$ under which all the matrices of the problem ($Q$ and $A_i\, \forall i = 1,\dots,m$) are block-decomposable (resp. block-separable).*

When dealing with block decomposable binary problems, we are interested in a relaxation of the problem that takes into account only the vertices of the boolean quadric polytope for each block. In this way we still consider an exponential set of extreme points, but much smaller than the one of the problem in the original space. We have to show, however, how to write the relaxation taking into account the intersections of the blocks, whether this is still a relaxation for the original problem and if it is equivalent to the one we introduced in Chapter 5. We will see that the problem of the equivalence is not trivial

and it is related to the matrix completion problems. The theory of matrix completion problems is well developed for semidefinite and completely positive completion. However, the BQP completion problem, to the best of our knowledge, has not been treated in depth, but it can be helpful and interesting, as we will see, because it is strictly related to binary quadratic problems.

In the rest of the chapter we will present our formulation, prove that it is a relaxation and we will show the equivalence to the previously introduced relaxation in particular cases.

## 6.1 Block-BQP relaxation

In this section we assume to have a problem of the form (5.3):

$$\begin{aligned} &\min \ \langle Q, X\rangle \\ &\text{s. t.} \ \langle A_i, X\rangle \leq b_i, \qquad\qquad \forall i = 1\ldots, m \\ &\qquad X = xx^\top \\ &\qquad x \in \{0,1\}^n \end{aligned}$$

and a block structure $\underline{\mathcal{B}_k}$ under which the problem is block-decomposable. We also introduce the following notation in order to split the data of the problem into the blocks:

**Definition 6.8.** *For every $j = 1, \ldots, k$ we define:*

$$Q^j \in \mathbb{R}^{n\times n} : (Q^j)_{pq} := \begin{cases} Q_{pq} & \text{if } (p,q) \in \underline{B_j} \setminus (\underline{B_1} \cup \cdots \cup \underline{B_{j-1}}) \\ 0 & \text{otherwise}, \end{cases}$$

$$A_i^j \in \mathbb{R}^{n\times n} : (A_i^j)_{pq} := \begin{cases} (A_i)_{pq} & \text{if } (p,q) \in \underline{B_j} \setminus (\underline{B_1} \cup \cdots \cup \underline{B_{j-1}}) \\ 0 & \text{otherwise}, \end{cases}$$

*for every $i = 1, \ldots, m$. We also use the following notation for the restriction to the blocks:*

$$\begin{aligned} X^{B_j} &:= X_{|\underline{B_j}} = \{X_{pq} | (p,q) \in \underline{B_j}\} \in \mathbb{R}^{d_j \times d_j}, \quad \forall X \in \mathbb{R}^{n\times n}, \\ x^{b_j} &:= x_{|\underline{b_j}} = \{x_p | p \in \underline{b_j}\} \in \mathbb{R}^{d_j}, \quad \forall x \in \mathbb{R}^n. \end{aligned}$$

If we introduce variables $Y_j \in \mathbb{R}^{d_j \times d_j}$ for every block $j = 1, \ldots, k$, we can provide a relaxation of the original problem based on the blocks. In the following, with an abuse of notation we consider $Q^j = (Q^j)^{B_j} \in \mathbb{R}^{d_j \times d_j}$ and $A_i^j = (A_i^j)^{B_j} \in \mathbb{R}^{d_j \times d_j}$. With this notation:

$$\langle Q, X\rangle = \sum_{j=1}^{k} \langle Q^j, X^{B_j}\rangle, \qquad \langle A_i, X\rangle = \sum_{j=1}^{k} \langle A_i^j, X^{B_j}\rangle, \quad \forall i = 1, \ldots, m.$$

Considering the variables $Y_j$, we can hence write the following problem:

$$\min \sum_{j=1}^{k} \langle Q^j, Y_j \rangle \tag{6.1a}$$

$$\text{s. t.} \sum_{j=1}^{k} \langle A_i^j, Y_j \rangle \leq b_i, \qquad \forall i = 1 \ldots, m \tag{6.1b}$$

$$Y_j^{B_j \cap B_h} = Y_h^{B_j \cap B_h} \qquad \forall 1 \leq j < h \leq k \tag{6.1c}$$

$$Y_j = \sum_{l \in \mathcal{P}_j} \mu_l^j (y_j^l)(y_j^l)^\top \qquad \forall j = 1, \ldots, k \tag{6.1d}$$

$$\sum_{l \in \mathcal{P}_j} \mu_l^j = 1 \qquad \forall j = 1, \ldots, k \tag{6.1e}$$

$$\mu_l^j \geq 0 \qquad \forall l \in \mathcal{P}_j, \qquad \forall j = 1, \ldots, k, \tag{6.1f}$$

where $y_j^l \in \{0,1\}^{d_j}$ $\forall l = 1, \ldots, 2^{d_j}$, $\forall j = 1, \ldots, k$ are binary vector of the dimension of the corresponding block $j$. Here, $\forall j = 1, \ldots, k$, $\mathcal{P}_j$ are the index sets of all the possible extreme points $y_j^l$:

$$\mathcal{P}_j = \{l \in \mathbb{N} | \, y_j^l \in \{0,1\}^{d_j}\}. \tag{6.2}$$

*Remark* 6.3. Clearly, if we consider the trivial decomposition in one single $n$ dimensional block, Formulation (6.1) is the same as what we proposed in (5.5) in Chapter 5.

*Remark* 6.4. $\mathcal{P}_j$, $j = 1, \ldots, k$ are still exponentially large, but their sizes depend on the size of the blocks: $|\mathcal{P}_j| = 2^{d_j}$. Hence, potentially the total number of points is reduced, with respect to the 1-block formulation: $\sum_{j=1}^{k} |\mathcal{P}_j| \ll 2^n$.

With this relaxation we allow a convex combination of extreme points for every block, with the additional requirement, given by constraint (6.1c), that the intersections of blocks must be consistent. From this formulation we can write the dual problem and a pricing problem for each block.

### 6.1.1 Restricted master, dual and pricing problems

Similarly to the case of one single block presented in Chapter 5, we proceed to solve this formulation by applying the Column Generation approach.

Hence, for each block $j = 1, \ldots k$, we consider a subset $\bar{\mathcal{P}}_j \subset \mathcal{P}_j$. The master problem restricted to points in $\bar{\mathcal{P}}_j$, has the following form:

$$\min \sum_{j=1}^{k} \langle Q^j, Y_j \rangle \tag{6.3a}$$

$$\text{s. t.} \sum_{j=1}^{k} \langle A_i^j, Y_j \rangle \leq b_i, \qquad \forall i = 1 \ldots, m \qquad [\alpha] \tag{6.3b}$$

$$Y_j^{B_j \cap B_h} = Y_h^{B_j \cap B_h} \qquad \forall 1 \leq j < h \leq k \qquad [\beta^{j,h}] \tag{6.3c}$$

$$Y_j = \sum_{l \in \bar{\mathcal{P}}_j} \mu_l^j (y_j^l)(y_j^l)^\top \qquad \forall j = 1, \ldots, k \qquad [\pi^j] \tag{6.3d}$$

$$\sum_{l \in \bar{\mathcal{P}}_j} \mu_l^j = 1 \qquad \forall j = 1, \ldots, k \qquad [\pi_0^j] \tag{6.3e}$$

$$\mu_l^j \geq 0 \qquad \forall l \in \bar{\mathcal{P}}_j, \quad \forall j = 1, \ldots, k. \tag{6.3f}$$

The variables in square brackets are the dual variables: $\alpha \in \mathbb{R}^m$ are the variables corresponding to the original constraints. $\beta^{j,h} \in \mathbb{R}^{d_{jh} \times d_{jh}}$ (where $d_{jh}$ is the dimension of the intersection between blocks $j$ and $h$, with $1 \leq j < h \leq k$) correspond to the intersection constraints. $\pi^j \in \mathbb{R}^{d_j \times d_j}$ and $\pi_0^j$, for every $j = 1, \ldots, k$, are the dual variables corresponding to the constraints of the convex combination of extreme points.

The dual of this restricted master is the following:

$$\max \; b^\top \alpha + \sum_{j=1}^{k} \pi_0^j \tag{6.4a}$$

$$\text{s. t.} \sum_{i=1}^{m} A_i^j \alpha_i + \sum_{h=1, h>j}^{k} \tilde{\beta}^{j,h} - \sum_{h=1, h<j}^{k} \tilde{\beta}^{h,j} + \pi^j = Q^j \qquad \forall j = 1, \ldots, k \tag{6.4b}$$

$$- \langle (y_j^l)(y_j^l)^\top, \pi^j \rangle + \pi_0^j \leq 0, \qquad \forall l \in \bar{\mathcal{P}}_j, \quad \forall j = 1, \ldots, k \tag{6.4c}$$

$$\alpha \leq 0, \tag{6.4d}$$

where, for every block $j, h = 1, \ldots, k$, $\tilde{\beta}^{j,h} \in \mathbb{R}^{d_j \times d_j}$ is equal to $\beta^{j,h}$ in the coordinates corresponding to the intersection with block $h$ and 0 otherwise. Following the DW decomposition approach, similarly to Section 5.3, we write the pricing problem, which is the problem of minimizing the reduced costs for the dual constraints (6.4c), once the optimal dual variables $\alpha$, $\beta^{j,h}$, $\pi^j$, and $\pi_0^j$ are obtained. It is the following:

$$\min \; \langle \pi^{j^*}, Y_j \rangle - \pi_0^{j^*} \tag{6.5a}$$

$$\text{s. t.} \; Y_j = y_j y_j^\top \tag{6.5b}$$

$$y_j \in \{0, 1\}^{d_j} \quad \forall j = 1, \ldots, k. \tag{6.5c}$$

*Remark* 6.5. The difference with respect to the pricing problem (5.11) in Section 5.3 is that here we have a pricing problem for each block and they are independent. Furthermore, the size of these problems is the size of the corresponding block, hence they are potentially much easier to solve than a single n-dimensional pricing problem.

## 6.2 Comparison to the original BQP relaxation

The question which we would like to address now is the relation between this block-decomposed formulation and the BQP-relaxation (5.5) of the problem that we introduced before. Our intention is to study in which cases they are equivalent. We start from proving that the latter formulation provides a lower bound for the first one by showing that any feasible point for (5.5) is feasible also for (6.1). Then, we will see that the vice versa problem is a matrix completion problem in the Boolean Quadric Polytope. We will describe the problem and we will prove it under some conditions.

To this aim, we start with the case of only two overlapping blocks, then we will see how the results can be extended to the case of several blocks and which additional conditions are needed on the block structure.

### 6.2.1 Two overlapping blocks

Here, we suppose that all the matrices $Q$, $A_i$ of the problem are decomposable in two overlapping blocks. Without loss of generality we can assume that each of the two blocks has consecutive components. Indeed, if it is not true, a permutation of the rows and columns can be done to make the blocks have the required structure.

#### Notations and settings

We introduce some notation specifically for the two-blocks case, which will be useful for our purposes. The matrix variables which we consider are: $X \in \mathbb{R}^{n\times n}$ the matrix for the one-block formulation, $Y \in \mathbb{R}^{p\times p}$ and $Z \in \mathbb{R}^{q\times q}$ the variables for the two-blocks formulation. We call $C$ the intersection of $Y$ and $Z$. All these matrices are square and we use the following notation for the dimensions: let $n$ be the side of $X$, and respectively $p$, $q$ and $r$ be the sides of $Y$, $Z$ and $C$. We consider the matrix $A$ obtained by removing the last $r$ rows and columns from $Y$, and the matrix $B$ obtained by removing the first $r$ rows and columns from $Z$. Let $s$ and $t$ be their sizes, respectively. Clearly, $s + r = p$, $r + t = q$ and $s + r + t = n$. We indicate with $\underline{B_3}, \underline{B_4}$, and $\underline{B_5}$ the blocks corresponding to $A$, $B$, and $C$. A picture of this notation is reported in Figure 6.2. It could be useful later to consider also the the block structures $\mathcal{B}' := \{\underline{B_1}, \underline{B_4}\}$ and $\mathcal{B}'' := \{\underline{B_3}, \underline{B_2}\}$.

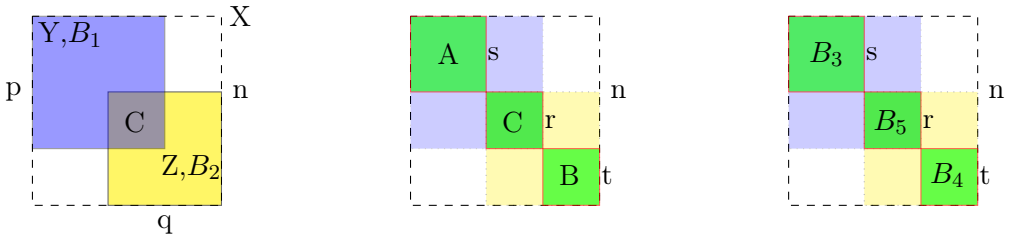



Figure 6.2 – Notations for the blocks.

Using the notation introduced in Definition 6.8, we can write:

$$\langle Q, X\rangle = \langle Q^1, X\rangle + \langle Q^2, X\rangle = \langle Q_1, X^{B_1}\rangle + \langle Q_2, X^{B_2}\rangle$$

and, similarly, the left-hand-side of the original constraints:

$$\langle A_i, X\rangle = \langle A_i^1, X\rangle + \langle A_i^2, X\rangle = \langle A_i^1, X^{B_1}\rangle + \langle A_i^2, X^{B_2}\rangle, \quad \forall i = 1\ldots, m.$$

In order to write the first relaxation of the problem (5.5), in particular from constraint (5.5c), we need to impose that the matrix $X$ is a convex combination of rank-1, $n$ by $n$ binary matrices. From Remark (5.2), we know that the number of extreme points is $2^n$. Hence, we can write:

$$X = \sum_{i=1}^{2^n} X_i \lambda_i, \tag{6.6}$$

for $\lambda_i \geq 0$, $\forall i = 1, \ldots, 2^n$, $\sum \lambda_i = 1$. If we follow the second relaxation (6.1) instead, based on the blocks, from (6.1d) we have similar conditions for the matrices $Y$ and $Z$. We write:

$$Y = \sum_{j=1}^{2^p} \mu_j Y_j \qquad Z = \sum_{h=1}^{2^q} \nu_h Z_h, \tag{6.7}$$

with $\mu_j, \nu_h \geq 0$, $\forall j = 1, \ldots, 2^p$, $\forall h = 1, \ldots, 2^q$, $\sum \mu_j = 1$ $\sum \nu_h = 1$.

Similarly, we can also write:

$$A = \sum_{k=1}^{2^s} \alpha_k A_k \qquad C = \sum_{l=1}^{2^r} \gamma_l C_l, \qquad B = \sum_{m=1}^{2^t} \beta_m B_m, \tag{6.8}$$

where $A_k$, $C_l$ and $B_m$ are rk-1 matrices, generated by $a_k \in \{0,1\}^s$, $c_l \in \{0,1\}^r$ and $b_m \in \{0,1\}^t$.

We now consider the extreme matrices $X_i$, $Y_j$, $Z_h$, $A_k$, $C_l$, and $B_m$. They are binary and rank-1, i.e. they are generated respectively by binary vectors $x_i$, $y_j$, $z_h$, $a_m$, $c_l$, and $b_m$ (of appropriate dimension) multiplied by the same vector transpose. The number of such matrices equals the number of such binary vectors.

**Definition 6.9.** *$\forall k \geq 1$, we introduce the following operation:*

$$\lceil \cdot \rfloor : \mathbb{R}^{p_1} \times \cdots \times \mathbb{R}^{p_k} \to \mathbb{R}^{(p_1+\cdots+p_k)\times(p_1+\cdots+p_k)}$$

$$(a_1 \ldots a_k) \mapsto \lceil a_1, \ldots, a_k \rfloor := \begin{pmatrix} a_1 \\ \vdots \\ a_k \end{pmatrix} (a_1^\top \ldots a_k^\top)$$

*where $a_1, \ldots, a_k$ are $k$ vectors of dimension $p_1, \ldots, p_k$.*

**Lemma 6.1** (Properties of $\lceil \cdot \rfloor$)**.** *Let $v_1, \ldots, v_n \in \mathbb{R}^p$, $w_1, \ldots, w_m \in \mathbb{R}^q$. Let $n = p + q$ and $M \in \mathbb{R}^{n\times n}$. Let $\underline{B_1} = \{1, \ldots, p\}^2$ be the first block and $\underline{B_2} = \{p+1, \ldots, n\}^2$ be the last block.*

- $\lceil x \rfloor = xx^\top$ *for any vector* $x$.
- *If* $M = \lceil v_i, w_j \rfloor \; \exists i, j$, *then* $M^{B_1} = \lceil v_i \rfloor$, $M^{B_2} = \lceil w_j \rfloor$.
- *If* $M = \sum_{i,j} \alpha_{i,j} \lceil v_i, w_j \rfloor$, *then*

$$M^{B_1} = \sum_{i,j} \alpha_{i,j} \lceil v_i, w_j \rfloor^{B_1} = \sum_{i,j} \alpha_{i,j} \lceil v_i \rfloor$$
$$M^{B_2} = \sum_{i,j} \alpha_{i,j} \lceil v_i, w_j \rfloor^{B_2} = \sum_{i,j} \alpha_{i,j} \lceil w_j \rfloor.$$

*Remark* 6.6. With our notation, every binary matrix $X_i = x_i x_i^\top \in \{0,1\}^{n \times n}$ can be rewritten as

$$X_i = \lceil x_i \rfloor = \lceil a_k, c_l, b_m \rfloor \tag{6.9}$$

where $a_k$, $c_l$, and $b_m$ are binary vectors of dimensions, respectively, $s$, $r$, and $t$. Indeed, it is sufficient to part the components of the vector $x_i$ into smaller vectors of the suitable dimension. Similarly, the same holds for splitting the generators of $Y$ and $Z$ into generators of, respectively, $A$ and $C$, $C$ and $B$. Since the number of generators in both cases is the same, also the converse holds: every matrix obtained by applying the operation $\lceil \cdot \rfloor$ to any couple or triple of (smaller) binary vectors is a binary rk-1 generator of the larger matrix.

With this results, we use the following notation for the rest of the chapter: $\forall k = 1, \ldots, 2^s$, $\forall l = 1, \ldots, 2^r$ and $\forall m = 1, \ldots, 2^t$ let $a_k \in \{0,1\}^s$, $c_l \in \{0,1\}^r$, and $b_m \in \{0,1\}^t$ be as before all the possible 0-1 vectors in dimension, respectively, $s$, $r$ and $t$. We can replace constraint (5.5c) by:

$$X = \sum_{k=1}^{2^s} \sum_{l=1}^{2^r} \sum_{m=1}^{2^t} \lambda_{k,l,m} \lceil a_k, c_l, b_m \rfloor \tag{6.10}$$

and, instead of constraints (6.1d) we write:

$$Y = \sum_{k=1}^{2^s} \sum_{l=1}^{2^r} \mu_{k,l} \lceil a_k, c_l \rfloor \tag{6.11}$$

$$Z = \sum_{l=1}^{2^r} \sum_{m=1}^{2^t} \nu_{l,m} \lceil c_l, b_m \rfloor \tag{6.12}$$

with suitable coefficients $\lambda_{k,l,m}$, $\mu_{k,l}$ and $\nu_{l,m}$.

To the last formulas we shall add a further constraint on the intersection $C$. We will see it later.

With this notation, we can write the two formulations which we are interested in, and state the problem of the equivalence of them. The first one considers the matrix $X$:

$$\min \ \langle Q, X \rangle \tag{6.13a}$$

$$\text{s. t.} \ \langle A_i, X \rangle \leq b_i, \qquad \forall i = 1 \ldots, m \tag{6.13b}$$

$$X = \sum_{k=1}^{2^s} \sum_{l=1}^{2^r} \sum_{m=1}^{2^t} \lambda_{k,l,m} \lceil a_k, c_l, b_m \rfloor \tag{6.13c}$$

$$\sum_{k=1}^{2^s} \sum_{l=1}^{2^r} \sum_{m=1}^{2^t} \lambda_{k,l,m} = 1 \tag{6.13d}$$

$$\lambda_{k,l,m} \geq 0 \qquad \forall k = 1, \ldots, 2^s, \forall l = 1, \ldots, 2^r, \forall m = 1, \ldots, 2^t. \tag{6.13e}$$

The second one takes into account the blocks, so it is expressed in terms of $Y$ and $Z$:

$$\min \ \langle Q_1, Y \rangle + \langle Q_2, Z \rangle \tag{6.14a}$$

$$\text{s. t.} \ \langle A_i^1, Y \rangle + \langle A_i^2, Z \rangle \leq b_i, \qquad \forall i = 1 \ldots, m \tag{6.14b}$$

$$Y = \sum_{k=1}^{2^s} \sum_{l=1}^{2^r} \mu_{k,l} \lceil a_k, c_l \rfloor \tag{6.14c}$$

$$Z = \sum_{l=1}^{2^r} \sum_{m=1}^{2^t} \nu_{l,m} \lceil c_l, b_m \rfloor \tag{6.14d}$$

$$Y^{B_5} = Z^{B_5} \tag{6.14e}$$

$$\sum_{k=1}^{2^s} \sum_{l=1}^{2^r} \mu_{k,l} = 1 \tag{6.14f}$$

$$\sum_{l=1}^{2^r} \sum_{m=1}^{2^t} \nu_{l,m} = 1 \tag{6.14g}$$

$$\mu_{k,l} \geq 0 \qquad \forall k = 1, \ldots 2^s, \quad \forall l = 1, \ldots 2^r \tag{6.14h}$$

$$\nu_{l,m} \geq 0 \qquad \forall l = 1, \ldots 2^r, \quad \forall m = 1, \ldots 2^t. \tag{6.14i}$$

*Remark* 6.7. Constraints (6.14e) corresponds to constraints (6.1c) in the previous formulation, and forces the equalities between elements in the intersection.

In order to prove the equivalence, at first we want to prove that every point which is feasible for the first formulation is also feasible for the second one, and then we will do the converse. If the first result is true, it means that our block-relaxation is not stronger than the first one. In particular, it is a valid relaxation for the original problem.

*Remark* 6.8. In order to state that the first and the second formulation are equivalent, the conditions $X^{B_1} = Y$ and $X^{B_2} = Z$ must be verified. Indeed, this guarantees that the objective function and the left-hand-side values of the original constraints do not change when passing from the one formulation to the other. In particular, if a point is feasible for

the first formulation, it is also feasible for the second one and the same objective function values are the same.

The first result is given by the following Proposition:

**Proposition 6.1.** *Given any feasible point for problem* (6.13)*, i.e. a feasible matrix* $X$ *and the corresponding coefficients* $\lambda_{k,l,m}$*, there exist* $Y$*,* $Z$ *and coefficients* $\mu_{k,l}$ *and* $\nu_{l,m}$*, such that* (6.14) *is feasible, and such that* $X^{B_1} = Y$ *and* $X^{B_2} = Z$*.*

*Proof.* It is sufficient to define the coefficients $\mu_{k,l}$ and $\nu_{l,m}$ in this way:

$$\mu_{k,l} := \sum_m \lambda_{k,l,m} \qquad \nu_{l,m} := \sum_k \lambda_{k,l,m}. \tag{6.15}$$

We shall prove that this solution satisfies the constraints (6.14e)-(6.14i). But $\mu_{k,l}$ and $\nu_{l,m}$ are sum of nonnegative numbers, from (6.13e), so they are nonnegative, and (6.13d) implies:

$$\sum_{k,l} \mu_{k,l} = \sum_{l,m} \nu_{l,m} = \sum_{k,l,m} \lambda_{k,l,m} = 1.$$

Moreover, using expression (6.13c) and the result in Lemma 6.1, we can write:

$$\begin{aligned} X^{B_1} &= \sum_{k=1}^{2^s} \sum_{l=1}^{2^r} \sum_{m=1}^{2^t} \lambda_{k,l,m} \lceil a_k, c_l, b_m \rfloor^{B_1} \\ &= \sum_{k=1}^{2^s} \sum_{l=1}^{2^r} (\sum_{m=1}^{2^t} \lambda_{k,l,m}) \lceil a_k, c_l \rfloor \\ &= \sum_{k=1}^{2^s} \sum_{l=1}^{2^r} \mu_{k,l} \lceil a_k, c_l \rfloor \\ &= Y, \end{aligned}$$

and with analogous calculations $X^{B_2} = \sum_{l=1}^{2^r} \sum_{m=1}^{2^t} \nu_{l,m} \lceil c_l, b_m \rfloor = Z$. In particular, $Y^{B_5} = X^{B_5} = Z^{B_5}$. □

*Remark* 6.9. We notice that the result also holds in the case of separable blocks, that is when $r = 0$. In this case there are no $l$ terms, but the definition (6.15) of $\mu_k$ and $\nu_m$ remains well defined without the index $l$, and the conditions are satisfied as well.

The second result that we would like to prove is the converse, which is expressed in the following statement:

**Statement 6.1.** *Given matrices* $Y$ *and* $Z$*, and the corresponding coefficients* $\mu_{k,l}$ *and* $\nu_{l,m}$ *such that* (6.14) *is feasible, is it possible to find a matrix* $X$ *and coefficients* $\lambda_{k,l,m}$ *such that* (6.13) *is feasible, and such that* $X^{B_1} = Y$ *and* $X^{B_2} = Z$*?*

*Remark* 6.10. Analogously to Remark 6.8, the latter condition implies that feasibility of the original constraints and the objective function value are maintained from the second to the first formulation. So, if the statement 6.1 holds, an optimal solution for (6.14) is feasible for (6.13). Together with the other implication, it proves that the two formulations are equivalent: any point can be expressed in both formulations with the same objective function value; in particular the optimal values are the same.

*Remark* 6.11. If we recall the definitions of cone completion problems seen in Section 1.5, it is easy to notice that statement 6.1 can be expressed as a *BQP-completion problem*. Indeed, the problem is to *complete* a matrix, specified only in the block-structure with the $Y_j$ terms, all of which are BQP matrices, to a full dimensional BQP matrix $X$. More specifically, the problem is to state if *every* block-decomposable partial BQP matrix can be completed, which is exactly the definition of a completion problem.

Proving if this result is always true is not a trivial task, even in the case of only two blocks. We firstly prove the following Lemma, which shows that the result of statement 6.1 holds if additional conditions are verified.

**Lemma 6.2.** *Under the hypotheses of statement 6.1, if in addition:*

$$\sum_{k=1}^{2^s} \mu_{k,l} = \sum_{m=1}^{2^t} \nu_{l,m} \tag{6.16}$$

*holds* $\forall l = 1, \ldots, 2^r$*, then Proposition 6.1 holds.*

*Proof.* Under these assumptions, we look for coefficients $\lambda_{k,l,m}$ that satisfy:

$$\sum_{m=1}^{2^t} \lambda_{k,l,m} = \mu_{k,l} \quad \forall k,\ \forall l \tag{6.17a}$$

$$\sum_{k=1}^{2^s} \lambda_{k,l,m} = \nu_{l,m} \quad \forall m,\ \forall l \tag{6.17b}$$

$$\lambda_{k,l,m} \geq 0 \quad \forall k,\ \forall l,\ \forall m. \tag{6.17c}$$

Indeed, if this holds, then clearly:

$$\sum_{k=1}^{2^s}\sum_{l=1}^{2^r}\sum_{m=1}^{2^t} \lambda_{k,l,m} = \sum_{k=1}^{2^s}\sum_{l=1}^{2^r} \mu_{k,l} = \sum_{l=1}^{2^r}\sum_{m=1}^{2^t} \nu_{l,m} = 1$$

and

$$X^{B_1} = \sum_{k=1}^{2^s}\sum_{l=1}^{2^r}(\sum_{m=1}^{2^t} \lambda_{k,l,m})\lceil a_k, c_l \rfloor = \sum_{k=1}^{2^s}\sum_{l=1}^{2^r} \mu_{k,l}\lceil a_k, c_l \rfloor = Y \tag{6.18}$$

$$X^{B_2} = \sum_{l=1}^{2^r}\sum_{m=1}^{2^t}(\sum_{k=1}^{2^s} \lambda_{k,l,m})\lceil c_l, b_m \rfloor = \sum_{l=1}^{2^r}\sum_{m=1}^{2^t} \nu_{l,m}\lceil c_l, b_m \rfloor = Z. \tag{6.19}$$

So, we just have to show that there exists a feasible solution for (6.17). But these are the constraints of a transportation problem for each fixed $l = 1, \dots, 2^r$. A classical result is that a transportation problem is feasible if and only if the sum of the right-hand-side of the first set of constraint equals the same sum in the second set of constraints, because both of them equal the global sum $\sum_{k,m} \lambda_{k,l,m}$. But in our case, this equality is exactly (6.16), so it holds by hypothesis and hence Proposition 6.1 holds. □

*Remark* 6.12. We notice that this Lemma is also true when $r = 0$. Indeed, both formulations (6.16) and (6.17) can be defined without the index $l$ in this case.

The hypotheses of Lemma 6.2 do not hold in general, but only in some specific cases. The following Proposition shows a result, with a hypothesis on the size of the intersection.

**Proposition 6.2.** *Under the hypotheses of Statement 6.1, if in addition the dimension of the intersection block is $r \leq 2$, then the answer to statement 6.1 is positive.*

*Proof.* From constraints (6.14c)- (6.14e), due to the property expressed in Lemma 6.1, we have:

$$\sum_{k,l} \mu_{k,l} \lceil c_l \rfloor = \sum_{l,m} \nu_{l,m} \lceil c_l \rfloor. \tag{6.20}$$

If all the matrices $\lceil c_l \rfloor$ are linearly independent, equality (6.16) must hold, for each $l$. Among the matrices $\lceil c_l \rfloor$ there is always the null matrix, which is dependent on the others. However, if the nonzero matrices are linearly independent, that is, all of the matrices are affinely independent, condition (6.16) holds for all $l$ such that $\lceil c_l \rfloor$ is nonzero. And since the sum of all coefficients is always 1, then by difference it holds also for the null matrix. If $r = 1$ or $r = 2$, it is easy to see that the matrices are affinely independent, so (6.16) holds, hence Proposition 6.1 holds. If $r = 0$, the hypothesis of Lemma 6.2 hold true: we can get rid of the index $l$ and note that $\sum_{k=1}^{2^s} \mu_k = \sum_{m=1}^{2^t} \nu_m = 1$. Hence, thanks to Remark 6.12, this result is proved for the separable case as well. □

This result holds with $r \leq 2$, but $\forall r > 2$, the number of $\lceil c_l \rfloor$ matrices is $2^r$, and it is greater than the dimension of the space $r(r+1)/2$ plus 1. Hence, they are affinely dependent and condition (6.16) cannot be directly obtained. Another strategy to prove this result is by following the proof of Caratheodory's Theorem (see, for instrance, [39]). However, it cannot be applied straightforwardly, because nonnegativity of the coefficients is not guaranteed.

In order to better understand this equivalence statement, and in which conditions we could expect that it holds, we pass to the case of several blocks. In this way it will be clear that the result does not hold in general and it will seem reasonable to propose a conjecture, even if, unfortunately, we have not been able to prove it yet.

### 6.2.2 Case of several blocks

In the previous subsection we proved that, whenever we have a 2-block decomposable problem, the solution of the BQP relaxation in the original space is always a lower bound

for the solution of the corresponding block-relaxation. Moreover, in some special cases the bounds obtained by the two formulations are the same, because equivalence can be proved. Now, we are interested to study a more generic class of block structure. The following remark should be noted:

*Remark* 6.13. We are interested in th *BQP-completion* problem. However, the PSD and CPP completion results, recalled in 1.5, help us and clearly show that the specification graphs of the matrices play a significant role. Hence, we could expect that the BQP-completion problem does not hold in general for every type of structure.

From now on we consider to have a block structure $\underline{\mathcal{B}_k}$ and that our problem is decomposable with respect to it. The formulation which we are dealing with are, on the one hand (5.5), which is our original relaxation. On the other side, with several blocks the formulation has the form as in (6.1).

**Generalization of the first inclusion**

It is easy to extend the result in Proposition 6.1 to the case of several blocks. This is shown in the following Proposition, and it implies that the block-decomposed formulation always provides a valid lower bound for the 1-block relaxation and hence for the optimal value of the original problem.

**Proposition 6.3.** *Given a block structure $\underline{\mathcal{B}_k}$, suppose that problem* (5.5) *is $\underline{\mathcal{B}_k}$-decomposable. Given any feasible point for problem* (5.5)*, i.e. a feasible matrix $X$ and the corresponding coefficients $\lambda_p$, $p = 1, \dots, 2^n$, then there exists a solution of* (6.1)*, given by $Y_j$ and $\mu_l^j$ with $l = 1, \dots, 2^{d_j}$, $j = 1, \dots, k$, such that $X^{B_j} = Y_j$ $\forall j = 1, \dots, k$.*

*Proof.* We are given a matrix $X$ and coefficients $\lambda_p \geq 0$ $\forall p$, $\sum_{p=1}^{2^n} \lambda_p = 1$, such that $X = \sum_{p=1}^{2^n} \lambda_p (x_p x_p^\top)$ with $x_p \in \{0,1\}^n$. We introduce the following notation. For every $j \in \{1, \dots, k\}$, let $\underline{\bar{b}_j} = \{1, \dots, n\} \backslash \underline{b_j}$ the complement of the block $\underline{b_j}$ in $\{1, \dots, n\}$. To each $p = 1, \dots 2^n$ we can assign a couple of indices $\{l, m\}$: $l \in \{1, \dots 2^{d_j}\}$, $m \in \{1, \dots 2^{n-d_j}\}$. Let $y_l := (x_p)^{b_j}$ and $z_m := (x_p)^{\bar{b}_j}$ be the restrictions of $x_p$ to $\underline{b_j}$ and $\underline{\bar{b}_j}$. We can hence rename $x_p$ as $x_{l,m}$, $\lambda_p$ as $\lambda_{l,m}$ and write:

$$X = \sum_{p=1}^{2^n} \lambda_p (x_p x_p^\top) = \sum_{l=1}^{2^{d_j}} \sum_{m=1}^{2^{n-d_j}} \lambda_{l,m} (x_{l,m}\, x_{l,m}{}^\top) \quad \forall j = 1, \dots, k. \tag{6.21}$$

Hence, for all $j = 1, \dots, k$ we can define:

$$\mu_l^j := \sum_{m=1}^{2^{n-d_j}} \lambda_{l,m} \quad \forall l = 1, \dots, 2^{d_j}. \tag{6.22}$$

Clearly, $\mu_l^j \geq 0$ and

$$\sum_{l=1}^{2^{d_j}} \mu_l^j = \sum_{l=1}^{2^{d_j}} \sum_{m=1}^{2^{n-d_j}} \lambda_{l,m} = \sum_{p=1}^{2^n} \lambda_p = 1.$$

Moreover, for all $j = 1, \ldots, k$, it holds:

$$X^{B_j} = \sum_{l=1}^{2^{d_j}} \sum_{m=1}^{2^{n-d_j}} \lambda_{l,m} (x_{l,m}\, x_{l,m}{}^{\top})^{B_j} = \sum_{l=1}^{2^{d_j}} (\sum_{m=1}^{2^{n-d_j}} \lambda_{l,m}) (y_l y_l^{\top}) = \sum_{l=1}^{2^{d_j}} \mu_l^j (y_l y_l^{\top}). \tag{6.23}$$

So, there is a feasible point for (6.1), equivalent to the solution of (5.5), where $\mu_l^j$ are given by (6.22) and $Y_j$ are defined as:

$$Y_j := \sum_{l=1}^{2^{d_j}} \mu_l^j (y_l y_l^{\top}) \quad \forall j = 1, \ldots, k. \tag{6.24}$$

□

*Remark* 6.14. To prove this direction of the equivalence, i.e. that the block decomposed relaxation gives a valid lower bound to the original relaxation, we did not need to add further hypotheses to the block structure.

### The BQP completion problem with several blocks

For the converse direction, which can be states in terms of a BQP completion problem, we firstly show that similarly to the PSD and CPP cases, not all the specification graphs are BQP completable.

**Proposition 6.4.** *If a graph is not chordal, then it is not BQP completable.*

*Proof.* If the graph is not chordal, it contains a cycle of length $l \geq 4$ with no chords. Without loss of generality we suppose that the vertices of this cycle are the first ones. Any matrix with this specification graph, restricted to the first $l$ entries, would be specified in the following entries: $\{i, i+1\}$ $\forall i = 1, \ldots, l-1$ and $\{1, l\}$ (and the symmetric elements, of course). This means that the block structure, restricted to these entries, is made up of $l$ consecutive diagonal 2 by 2 blocks, and one 2 by 2 block connecting the first and the last entry of the cycle. Hence, we can always have the following matrix (see [6], example 1.35). The question marks correspond to unspecified elements.

$$\frac{1}{2} \begin{pmatrix} 1 & 1 & ? & \ldots & ? & 0 \\ 1 & 1 & \ddots & \ddots & \ddots & ? \\ ? & 1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & ? \\ ? & \ddots & \ddots & \ddots & 1 & 1 \\ 0 & ? & \ldots & ? & 1 & 1 \end{pmatrix}. \tag{6.25}$$

Indeed, every 2 by 2 diagonal matrix is given by

$$\frac{1}{2} \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

and the 2 by 2 matrix restricted to elements $\{1, l\}$ is given by

$$\frac{1}{2}\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix} + \frac{1}{2}\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}$$

so they are convex combinations of 2 by 2 rank-1 binary matrices. But it is known (see [87], Lemma 6) that the only PSD matrix with ones on the entries $(i, j)$ s.t. $|i-j| \leq 1$ is the all-1 matrix, hence the matrix (6.25) is not PSD-completable. Hence, it not BQP completable, since BQP is a subset of the semidefinite cone. □

However, the result holds for graphs which are more general than block-clique, due to Proposition 6.2 and the properties of chordal graphs:

**Proposition 6.5.** *If a graph $G$ is chordal and the size of the intersection of any two maximal cliques of $G$ is at most 2, then $G$ is BQP-completable.*

*Proof.* We prove the statement by induction on the number $n$ of maximal cliques. If there are only two maximal cliques, the result is given by Proposition 6.2. Now we suppose to have $n > 2$ maximal cliques and we assume by inductive hypothesis that the result holds true for $n-1$ cliques. Since the graph $G$ is chordal, there is a perfect elimination ordering (PEO) of its vertices (see Proposition 1.4). Without loss of generality, we suppose that the vertices of $G$ are sorted according to this ordering. We also sort the maximal cliques according to the order of their first vertex. We recall that, by definition of PEO, each vertex, together with its neighbours which follow it in the order, form a clique. In particular, all the neighbours of the first vertex belong to the first maximal clique $C_1$. We now consider the subgraph $G'$ of $G$ induced by the last $n-1$ maximal cliques $C_2 \cup \cdots \cup C_n$. By inductive hypothesis $G'$ is BQP-completable. Hence, $G$ is completable if $G''$ is completable, where $G''$ is obtained by adding to $G$ all the edges which complete $G'$. In this way we now have two cliques: $C_1$ and $C_2 \cup \cdots \cup C_n$. We notice that $C_1 \cap (C_2 \cup \cdots \cup C_n) = C_1 \cap C_2$, again for the PEO property. Hence, if $|C_1 \cap C_2| \leq 2$, we can apply again Proposition 6.2 and conclude the proof. □

**Corollary 6.1.** *In particular, block-clique graphs are BQP-completable.*

*Remark* 6.15. We already observed that $BQP \subset \mathcal{C}^* \subset \mathcal{S}_+$. However, the class of graphs which are BQP-completable is larger than the CPP-completable graphs, which are the block-clique graphs. We also showed in Proposition 6.4 that the BQP-completable graphs must be chordal, so the set of these graphs is included in the set of PSD-completable graphs.

As a consequence, it is natural to investigate what can happen for more general chordal graphs. A natural conjecture is the following:

**Proposition 6.6** (Conjecture)**.** *Every chordal graph is BQP-completable.*

This would imply, together with Proposition 6.4, that a graph is BQP-completable if and only if it is chordal. To the best of our knowledge, this conjecture has not been studied

yet. However, this result could be very useful, because it could allow to efficiently solve sparse problems with a technique similar to the one used in semidefinite programming, and with a stronger bound. We do not have the proof for this result, but numerical results seem suggest that it holds, as we will see in the following section.

In order to prove the conjecture, we notice that we can restrict ourselves to a simpler case, where we have only two blocks, both of them have dimension equal to $n-1$, and only the element $\{1, n\}$ is not covered by the blocks. This means that the dimension of the intersection of the blocks is $n-2$ and the dimension of both $s$ and $t$, with notation in Section 6.2.1, is 1. This result is obtained by following the proof of the PSD-completion problem in [87]: the authors strongly exploit the structure and the properties of chordal graphs, showing that they can reduce to two blocks with only one missing entry. Under these assumptions they prove their result for PSD completion and they prove that they can extend it to every chordal graph, just using the properties of the structure of these graphs. We could try to prove the same result for BQP matrices, but it is not straightforward. We also have to keep in mind that this result do not hold if the extreme matrices are general real nonnegative rank-1 matrices, because it would be the case of CPP-completion, which only holds for block-clique graphs.

Inspired by these results, we provide some tests, where we compare the numerical results obtained by our block-decomposable relaxation with a chordal block structure, and the one-block formulation. We hence expect that the block-decomposition always provides a lower bound; if in some cases its value is strictly lower that the other one, the conjecture is false.

In the following sections, we present the computational aspects of this formulation and computational results; then we will conclude, showing remarks and possible directions to continue this study.

## 6.3 Computational aspects

From the computational point of view, Formulation (6.1) has some differences with respect to Formulation (5.5). Firstly, the most evident improvement is that the number of extreme points is reduced, as well as the size of the pricing problems, which was the most time consuming part in the one-block formulation. However, this makes the master program more difficult, although still linear, because of the high number of constraints for the intersection of the blocks. The result is that if the number of blocks is relatively high and the size of the intersections between blocks is large, then the convergence can be slow. But if the blocks have little overlapping, or they are separated, then this formulation is dramatically faster than the previous one. In the following paragraphs we treat in detail how we obtained the block decomposition, and specific techniques which we used to solve master and pricing problems.

**Block decomposition** We already noticed that the block structure is not unique. One of the main issues is how to build a block structure that fits the data, and which structures are *better* than others. We shall note that the *aggregate* sparsity graph of the problem,

which has one vertex for each variable and one edge corresponding to every nonzero quadratic term in the objective function or in a constraint, is not chordal in general. Hence, we need to add edges in order to have the desired sparsity graph. This is what is called *chordal extension* of the sparsity graph of the problem. The approach used in Semidefinite programming and described in [70] searches for the *minimal* chordal extension, i.e. that which requires the least number of additional edges. Actually, computing the minimal chordal extension is a NP-hard problem, and heuristics are proposed, for instance in [70], to find a minimal extension quickly. As we will notice, however, this is generally not the best choice in our case, and the quality of the bounds does not depend on the chordal extension, of course, if the conjecture is true. We implemented an algorithm which finds the maximal cliques, and which then generates the additional edges to obtain a chordal extension.

The drawback of the minimal chordal extension criterion is the following: for any couple of blocks with nonempty intersection, we need to add a number of constraints in the master program which is polynomial on the size of the intersection. Hence, if there exist two blocks with most of the components in common, it is probably more convenient to group the two of them in a single block, even if its dimension becomes larger.

In order to verify this intuition, we considered some instances from the QPLIB library, which we tested for the one-block formulation in Chapter 5. We firstly obtained the maximal cliques and the chordal extension. Then, we grouped together cliques with a certain percentage of elements in common and verified that they are still chordal. We let the percentage vary from 0 to 100 and we collected all the different decomposition patterns. We do not report here the details of these experiments because, for the chosen instances, they show that the best option is keeping one single full-dimensional block and in most cases the block-decomposed relaxation is not solved within the time limit. This proves that if there are several linking constraints between different blocks, the decomposed formulation is not efficient. However, results on block-separate problems, which will follow, show that the improvement is very good. This means that in some cases the block formulation is actually helpful. It is also possible to improve the resolution of master and pricing problems, as we will see in the paragraphs which follow.

**Master problem** The master problem is solved with Cplex. Its initialization is made with artificial variables as in the one-block case. The elevated number of variables can be reduced by a technique of column dropping: we regularly remove some extreme columns with 0 value in the convex combination which expresses the optimal point. More specifically, in particular when there are intersections between blocks, the number of extreme points which are generated grows very fast, but most of them are discarded, because only a small number of columns is needed, due to the intersection constraints. Hence, we decided to remove some of the columns with 0 weight. However, they have reduced costs which contribute to the solution of the dual and hence the pricing problem. Our choice is to remove only 60% of the 0-weighted extreme points, every ten iterations. We tried also other percentages, and more sophisticated choices could be done: columns with the highest reduced costs, or those which have 0 weight for several iterations, would

more likely be unnecessary for the convergence of the algorithm.

**Pricing problems** The pricing problems are solved with Cplex as well. An early stopping technique can be applied with the same principle as in the previous chapter. However, in this framework more techniques can be added, since we have several independent pricing problems. We can solve all of them at each iteration, or just solve the pricing problems until we find a new column with negative reduced cost: this is sufficient for the convergence and we can directly pass to the master, without solving the remaining subproblems. We can also add several columns at each time the pricing is solved. All these parameters have been preliminarily tested. The best options are generally those which make the pricing phase faster: stopping the computations of the pricing when a column with reduced cost is found and using the early stopping technique in every pricing problem.

## 6.4 Numerical results

Here we present our computational results, which show some interesting and promising results.

We present the tests on the *sonet* instances, which complete the table presented in Chapter 5 with results given by the block-decomposed framework. Table 6.1 collects the results, with the results of BiqCrunch and the 1-block BQP-relaxation obtained in Chapter 5 for comparison. The BiqCrunch results are restraint to the only *bound* and *triangular inequalities* cases.

We notice that these instances are characterized by a separable structure, hence they are particularly suited for our algorithm. Indeed, our performances are extremely good in most instances: we obtain the same bound as before, (as expected) within a very short time, and we solve many more instances than before: we only have two failures. Since these instances are so easy to solve, in this case the techniques to speed up the pricing do not change the overall computational time, so the reported results are obtained with the default algorithm: several pricing problems are computed for every master iteration.

Similarly to what we presented in the previous chapter, in Table 6.2 we present the number of iterations, the dimension of the final master, and the partition of time between master and pricing problems. Here, the sizes of both the master and the pricing problems are very small, thus the algorithm solves both the problems really fast. The master being linear, it is easier to solve than the pricing.

Among these instances there are some of those of the QPLIB library which we tested in the previous chapter. However, our algorithm does not perform well with the block structure on the non-separable QPLIB instances: we reached the time limit of 10 hours in all the cases. This is mainly due to the difficulty of the formulation, in which the master has a considerable number of variables and constraints for the intersections of the blocks. Several improvements could be implemented to obtain a faster resolution of these instances, even if the complexity of these problems remains hard.

Finally, we generate some tests to support our conjecture. We randomly generate instances made up of two blocks, each of them of size one less than the original problem, so that only one element is not covered. We put linear random constraint and a quadratic objective function with elements on these two blocks. The size of the problems goes from 4 (the first case in which the equivalence result is not proved) to 20. For each case we run 5 different seeds and we use two different ways of generating the constraints, hence we have 10 instances for each dimension. In Table 6.3 we present the average objective function of the resolution with one or two blocks, along with the average number of iterations and the average computational time in seconds. A $*$ mark means that at least in one instance the maximum number (100000) of iterations is reached, and hence the problem is not solved to optimality.

The results shows that the optimal values are always the same (except for the last cases, in which the optimal value is not always available), so the conjecture is not disproved. Moreover, it is worth noting that the number of iterations for the two-blocks formulation increases dramatically when the size increases, and consequently the running time increases as well. This shows that, in these cases, it is more convenient to merge the two blocks into a single one.

## 6.5 Conclusions, applications and future research directions

In this chapter we introduced a block-decomposable adaptation of the column generation-based relaxation, already introduced in Chapter 5. We showed that this one is suited for problems which have a sparse structure. We carefully analyzed the relations with respect to the original relaxation, specifically we discussed the equivalence of the two formulations. We showed that the latter is always a relaxation of the previous one, while in general it is not clear whether the equivalence holds or not. We noticed the connection with the problem of matrix completion. In this way, we showed that the equivalence problem can be stated as a completion problem on the BQP polytope and we could highlight some conditions for the proof of the equivalence. More specifically, we identified classes of problems where the result is true and other classes where it is not, and we noted similarities with the results in the completely positive and semidefinite cones. We proved that the BQP completion is possible for problems with chordal sparsity graphs if the maximal dimension of the intersection of blocks is 2, and if the graph is not chordal, completion does not hold. A complete proof of the BQP completion result is still open. However, computational results seem to support the conjecture that the class of problems in which it holds is the class of problems with a chordal sparsity graph. This is a result which potentially can help solving structured binary nonconvex quadratically constrained, quadratic problems. Our results show that the proposed algorithm is efficient for the computation of the BQP bound when the matrices can be decomposed in separable blocks. For more general problems, with strongly overlapping blocks, improvements or further developments are needed.

Improvements in the implementation of master and pricing, which make the method more efficient on more general structured problems should be developed. Furthermore,

the proof for the completion problem shall be completed, and the result seems to be interesting from the theoretical point of view.

This concludes the study which has been done so far. However, other developments are planned, with the aim to continue the research on some of the question which are still not completely answered. A description of the possible research projects will follow in the next, conclusive chapter.

| Instance | BC-bound | | BC-cuts | | BQP | | BQP-blocks | |
|---|---|---|---|---|---|---|---|---|
| | T (s) | Bound | T (s) | Bound | T (s) | Bound | T (s) | Bound |
| gr17-nc-qc.lp | 38 | 15803 | 242 | 21831 | 8 | 21973 | 0.2 | 21973 |
| ins.16.v1-nc-qc.lp | 21 | -387987 | 22 | -387987 | 703 | 6636 | 0.4 | 6636 |
| ins.16.v2-nc-qc.lp | 21 | -157176 | 21 | -157176 | 169 | 75322 | 0.3 | 75322 |
| ins.16.v3-nc-qc.lp | 64 | 929 | 120 | 1571 | 277 | 1596 | 0.2 | 1596 |
| ins.16.v5-nc-qc.lp | 28 | 1872 | 169 | 3120 | 187 | 3193 | 0.2 | 3193 |
| ins.16.v6-nc-qc.lp | 44 | 2119 | 113 | 3912 | 185 | 3990 | 0.2 | 3990 |
| ins.17.v1-nc-qc.lp | 27 | -795830 | 28 | -795830 | - | - | 0.3 | 23921 |
| ins.17.v2-nc-qc.lp | 28 | -174478 | 28 | -174478 | 22 | 138612 | 0.3 | 138612 |
| ins.17.v3-nc-qc.lp | 22 | 2677 | 300 | 4026 | 15 | 4117 | 0.3 | 4117 |
| ins.17.v5-nc-qc.lp | 37 | 11000 | 227 | 16400 | 7 | 16526 | 0.2 | 16526 |
| ins.17.v6-nc-qc.lp | 26 | 12061 | 273 | 18300 | 7 | 18452 | 0.3 | 18452 |
| ins.18.v1-nc-qc.lp | 36 | -384592 | 36 | -384592 | 920 | 909 | 0.6 | 909 |
| ins.18.v2-nc-qc.lp | 39 | -117439 | 38 | -117439 | 1592 | 14816 | 0.9 | 14816 |
| ins.18.v3-nc-qc.lp | 80 | 997 | 218 | 1740 | 2544 | 1758 | 0.2 | 1758 |
| ins.18.v5-nc-qc.lp | 39 | -108 | 39 | -108 | 876 | 3516 | 0.3 | 3516 |
| ins.18.v6-nc-qc.lp | 38 | -1453 | 38 | -1453 | 1076 | 4394 | 0.3 | 4394 |
| ins.19.v1-nc-qc.lp | 52 | -368130 | 52 | -368130 | - | - | 0.5 | 21771 |
| ins.19.v2-nc-qc.lp | 52 | -1034144 | 51 | -1034144 | - | - | 0.5 | 10834 |
| ins.19.v3-nc-qc.lp | 55 | -2946 | 55 | -2946 | 701 | 1906 | 0.4 | 1906 |
| ins.19.v5-nc-qc.lp | 54 | -17124 | 54 | -17124 | 1651 | 3811 | 0.4 | 3811 |
| ins.19.v6-nc-qc.lp | 53 | -20670 | 53 | -20670 | 1724 | 4764 | 0.4 | 4764 |
| ins.20.v1-nc-qc.lp | 64 | -996229 | 64 | -996229 | 500 | 211908 | 0.7 | 211908 |
| ins.20.v2-nc-qc.lp | 67 | -571504 | 67 | -571504 | - | - | 0.8 | 118351 |
| ins.20.v3-nc-qc.lp | 69 | -1158 | 69 | -1158 | - | - | 0.2 | 592 |
| ins.20.v5-nc-qc.lp | 67 | -1990 | 68 | -1990 | - | - | 0.2 | 1183 |
| ins.20.v6-nc-qc.lp | 68 | -12886 | 69 | -12886 | - | - | 0.2 | 1478 |
| ins.21.v1-nc-qc.lp | 83 | -4910836 | 83 | -4910836 | - | - | 1 | 237976 |
| ins.21.v2-nc-qc.lp | 85 | -1539431 | 84 | -1539431 | - | - | 1.4 | 75682 |
| ins.21.v3-nc-qc.lp | 86 | -28206 | 84 | -28206 | - | - | 0.4 | 1431 |
| ins.21.v5-nc-qc.lp | 86 | -12247 | 88 | -12247 | - | - | 0.4 | 12435 |
| ins.21.v6-nc-qc.lp | 89 | 2980 | 88 | 2980 | 128 | 20279 | 0.3 | 20279 |
| ins.22.v1-nc-qc.lp | 102 | -7062597 | 103 | -7062597 | - | - | 1.6 | 323439 |
| ins.22.v2-nc-qc.lp | 105 | -2995516 | 105 | -2995516 | - | - | 1.4 | 100031 |
| ins.22.v3-nc-qc.lp | 113 | 1548 | 113 | 1548 | 187 | 7033 | 0.5 | 7033 |
| ins.22.v5-nc-qc.lp | 112 | 1821 | 112 | 1821 | 197 | 14065 | 0.5 | 14065 |
| ins.22.v6-nc-qc.lp | 112 | 4360 | 111 | 4360 | 842 | 17581 | 0.6 | 17581 |
| ins.23.v1-nc-qc.lp | 136 | 256040 | 137 | 256040 | - | - | - | - |
| ins.23.v2-nc-qc.lp | 132 | 516197 | 141 | 516197 | - | - | - | - |
| ins.23.v3-nc-qc.lp | 137 | -33822 | 137 | -33822 | 25710 | 2971 | 0.4 | 2971 |
| ins.23.v5-nc-qc.lp | 138 | -70162 | 139 | -70162 | - | - | 0.4 | 5942 |
| ins.23.v6-nc-qc.lp | 141 | -107088 | 141 | -107088 | - | - | 0.5 | 7427 |
| ins.24.v1-nc-qc.lp | 164 | -5722660 | 159 | -5722660 | - | - | 2.7 | 18946 |
| ins.24.v2-nc-qc.lp | 162 | -2777104 | 162 | -2777104 | - | - | 2.5 | 8820 |
| ins.24.v3-nc-qc.lp | 181 | -52206 | 181 | -52206 | 486 | 6289 | 0.5 | 6289 |
| ins.24.v5-nc-qc.lp | 176 | -137132 | 176 | -137132 | - | - | 0.5 | 12578 |
| ins.24.v6-nc-qc.lp | 175 | -185316 | 175 | -185316 | 2292 | 15722 | 0.5 | 15722 |
| ins.25.v1-nc-qc.lp | 205 | -3859934 | 205 | -3859934 | - | - | 3.3 | 110413 |
| ins.25.v2-nc-qc.lp | 207 | -1251807 | 207 | -1251807 | - | - | 2.7 | 53972 |
| ins.25.v3-nc-qc.lp | 217 | -9687 | 202 | -9687 | 163 | 11758 | 0.6 | 11758 |
| ins.25.v5-nc-qc.lp | 205 | -30970 | 216 | -30970 | 316 | 23516 | 0.5 | 23516 |
| ins.25.v6-nc-qc.lp | 207 | -62888 | 190 | -62888 | 231 | 29394 | 0.5 | 29394 |

Table 6.1 – Root node bound and time for QCQP instances.

| Instance | BQP bound | N. its | Final dim | Total time (s) | Time master (s) | Time pricing (s) |
|---|---|---|---|---|---|---|
| gr17-nc-qc.lp | 21973 | 3 | 31 | 0.162 | 0.002 | 0.143 |
| ins.16.v1-nc-qc.lp | 6636 | 13 | 7 | 0.439 | 0.007 | 0.382 |
| ins.16.v2-nc-qc.lp | 75322 | 13 | 7 | 0.328 | 0.005 | 0.295 |
| ins.16.v3-nc-qc.lp | 1596 | 4 | 23 | 0.171 | 0.002 | 0.153 |
| ins.16.v5-nc-qc.lp | 3193 | 4 | 23 | 0.168 | 0.002 | 0.151 |
| ins.16.v6-nc-qc.lp | 3990 | 4 | 23 | 0.163 | 0.002 | 0.147 |
| ins.17.v1-nc-qc.lp | 23921 | 8 | 9 | 0.282 | 0.003 | 0.249 |
| ins.17.v2-nc-qc.lp | 138612 | 10 | 19 | 0.302 | 0.004 | 0.264 |
| ins.17.v3-nc-qc.lp | 4117 | 6 | 23 | 0.322 | 0.002 | 0.295 |
| ins.17.v5-nc-qc.lp | 16526 | 5 | 29 | 0.193 | 0.002 | 0.167 |
| ins.17.v6-nc-qc.lp | 18452 | 6 | 29 | 0.279 | 0.003 | 0.251 |
| ins.18.v1-nc-qc.lp | 909 | 17 | 4 | 0.590 | 0.007 | 0.520 |
| ins.18.v2-nc-qc.lp | 14816 | 24 | 9 | 0.852 | 0.010 | 0.760 |
| ins.18.v3-nc-qc.lp | 1758 | 4 | 5 | 0.212 | 0.002 | 0.186 |
| ins.18.v5-nc-qc.lp | 3516 | 4 | 5 | 0.328 | 0.003 | 0.282 |
| ins.18.v6-nc-qc.lp | 4394 | 4 | 5 | 0.284 | 0.002 | 0.245 |
| ins.19.v1-nc-qc.lp | 21771 | 11 | 7 | 0.508 | 0.004 | 0.449 |
| ins.19.v2-nc-qc.lp | 10834 | 10 | 12 | 0.493 | 0.004 | 0.438 |
| ins.19.v3-nc-qc.lp | 1906 | 7 | 5 | 0.424 | 0.003 | 0.370 |
| ins.19.v5-nc-qc.lp | 3811 | 7 | 5 | 0.354 | 0.003 | 0.312 |
| ins.19.v6-nc-qc.lp | 4764 | 7 | 5 | 0.390 | 0.003 | 0.343 |
| ins.20.v1-nc-qc.lp | 211908 | 14 | 15 | 0.693 | 0.008 | 0.604 |
| ins.20.v2-nc-qc.lp | 118351 | 17 | 13 | 0.819 | 0.011 | 0.710 |
| ins.20.v3-nc-qc.lp | 592 | 3 | 17 | 0.190 | 0.002 | 0.158 |
| ins.20.v5-nc-qc.lp | 1183 | 4 | 19 | 0.238 | 0.002 | 0.199 |
| ins.20.v6-nc-qc.lp | 1478 | 4 | 20 | 0.234 | 0.002 | 0.198 |
| ins.21.v1-nc-qc.lp | 237976 | 22 | 7 | 1.001 | 0.014 | 0.848 |
| ins.21.v2-nc-qc.lp | 75682 | 27 | 7 | 1.378 | 0.018 | 1.201 |
| ins.21.v3-nc-qc.lp | 1431 | 6 | 11 | 0.385 | 0.003 | 0.333 |
| ins.21.v5-nc-qc.lp | 12435 | 5 | 11 | 0.375 | 0.002 | 0.331 |
| ins.21.v6-nc-qc.lp | 20279 | 4 | 35 | 0.326 | 0.002 | 0.282 |
| ins.22.v1-nc-qc.lp | 323439 | 27 | 11 | 1.573 | 0.017 | 1.355 |
| ins.22.v2-nc-qc.lp | 100031 | 23 | 11 | 1.409 | 0.017 | 1.213 |
| ins.22.v3-nc-qc.lp | 7033 | 6 | 35 | 0.490 | 0.002 | 0.441 |
| ins.22.v5-nc-qc.lp | 14065 | 6 | 35 | 0.494 | 0.002 | 0.445 |
| ins.22.v6-nc-qc.lp | 17581 | 6 | 35 | 0.625 | 0.003 | 0.538 |
| ins.23.v1-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.23.v2-nc-qc.lp | M.I. | - | - | - | - | - |
| ins.23.v3-nc-qc.lp | 2971 | 4 | 11 | 0.387 | 0.002 | 0.328 |
| ins.23.v5-nc-qc.lp | 5942 | 4 | 11 | 0.392 | 0.002 | 0.332 |
| ins.23.v6-nc-qc.lp | 7427 | 4 | 11 | 0.455 | 0.003 | 0.364 |
| ins.24.v1-nc-qc.lp | 18946 | 32 | 13 | 2.712 | 0.036 | 2.325 |
| ins.24.v2-nc-qc.lp | 8820 | 37 | 4 | 2.528 | 0.058 | 2.103 |
| ins.24.v3-nc-qc.lp | 6289 | 5 | 39 | 0.537 | 0.002 | 0.465 |
| ins.24.v5-nc-qc.lp | 12578 | 5 | 39 | 0.542 | 0.002 | 0.469 |
| ins.24.v6-nc-qc.lp | 15722 | 5 | 39 | 0.542 | 0.002 | 0.469 |
| ins.25.v1-nc-qc.lp | 110413 | 42 | 6 | 3.308 | 0.046 | 2.790 |
| ins.25.v2-nc-qc.lp | 53972 | 34 | 6 | 2.700 | 0.065 | 2.234 |
| ins.25.v3-nc-qc.lp | 11758 | 4 | 43 | 0.595 | 0.003 | 0.490 |
| ins.25.v5-nc-qc.lp | 23516 | 4 | 43 | 0.511 | 0.002 | 0.437 |
| ins.25.v6-nc-qc.lp | 29394 | 4 | 43 | 0.540 | 0.002 | 0.447 |

Table 6.2 – Repartition of time between master and pricing for Sonet instances.

| Size | type | Optimal value | | N iterations | | CPU time (s) | |
|---|---|---|---|---|---|---|---|
| | | Single | Blocks | Single | Blocks | Single | Blocks |
| 4 | 0 | -0.6085 | -0.6085 | 3.6 | 7.0 | 0.005 | 0.008 |
| 4 | 1 | -0.7094 | -0.7094 | 2.2 | 7.6 | 0.004 | 0.008 |
| 5 | 0 | -0.6842 | -0.6842 | 3.6 | 11.8 | 0.005 | 0.014 |
| 5 | 1 | -1.2321 | -1.2321 | 2.4 | 13.0 | 0.004 | 0.014 |
| 6 | 0 | -1.3767 | -1.3767 | 3.6 | 15.6 | 0.009 | 0.030 |
| 6 | 1 | -2.3461 | -2.3461 | 3.0 | 10.2 | 0.006 | 0.016 |
| 7 | 0 | -1.6038 | -1.6038 | 5.4 | 31.8 | 0.019 | 0.070 |
| 7 | 1 | -2.8288 | -2.8288 | 4.0 | 18.2 | 0.007 | 0.032 |
| 8 | 0 | -1.9613 | -1.9613 | 4.4 | 45.8 | 0.021 | 0.135 |
| 8 | 1 | -3.1419 | -3.1419 | 4.4 | 31.0 | 0.013 | 0.080 |
| 9 | 0 | -2.4062 | -2.4062 | 6.2 | 52.2 | 0.036 | 0.203 |
| 9 | 1 | -3.4143 | -3.4143 | 4.0 | 32.4 | 0.019 | 0.111 |
| 10 | 0 | -2.6951 | -2.6951 | 9.6 | 69.6 | 0.062 | 0.341 |
| 10 | 1 | -4.4063 | -4.4063 | 4.8 | 112.6 | 0.020 | 0.492 |
| 11 | 0 | -3.2378 | -3.2378 | 8.0 | 164.6 | 0.100 | 1.070 |
| 11 | 1 | -5.4270 | -5.4270 | 7.0 | 273.6 | 0.042 | 1.857 |
| 12 | 0 | -3.3485 | -3.3485 | 13.4 | 184.8 | 0.337 | 2.613 |
| 12 | 1 | -5.9781 | -5.9781 | 7.6 | 706.2 | 0.083 | 7.866 |
| 13 | 0 | -4.0986 | -4.0986 | 16.2 | 187.0 | 0.598 | 2.512 |
| 13 | 1 | -5.4834 | -5.4834 | 8.0 | 1992.2 | 0.089 | 23.745 |
| 14 | 0 | -4.2510 | -4.2510 | 16.4 | 342.8 | 0.846 | 6.496 |
| 14 | 1 | -5.9053 | -5.9053 | 9.4 | 2230.8 | 0.117 | 32.682 |
| 15 | 0 | -4.4630 | -4.4630 | 19.0 | 645.8 | 1.436 | 19.477 |
| 15 | 1 | -6.8074 | -6.8074 | 10.0 | * 30117.6 | 0.240 | 449.012 |
| 16 | 0 | -4.6652 | -4.6652 | 19.2 | 2997.6 | 2.266 | 113.390 |
| 16 | 1 | -6.6604 | -6.6604 | 11.6 | 18475.0 | 0.384 | 332.083 |
| 17 | 0 | -5.6246 | -5.6246 | 24.6 | 2333.0 | 4.385 | 83.688 |
| 17 | 1 | -7.2088 | -7.2088 | 11.8 | * 78874.6 | 0.516 | 3232.594 |
| 18 | 0 | -6.1250 | -6.1250 | 26.2 | 1949.2 | 4.162 | 103.928 |
| 18 | 1 | -9.3915 | -9.3915 | 9.6 | 49122.0 | 0.336 | 1158.783 |
| 19 | 0 | -7.0102 | -7.0102 | 19.6 | 4715.6 | 4.120 | 274.248 |
| 19 | 1 | -11.4662 | -11.3411 | 8.4 | * 80140.4 | 0.250 | 2393.220 |
| 20 | 0 | -7.4338 | -7.4338 | 24.0 | 3791.6 | 4.766 | 251.151 |
| 20 | 1 | -12.5117 | -11.6938 | 8.2 | * 62056.8 | 0.496 | 1878.439 |

Table 6.3 – Solution of random instances with one or two blocks.

# Chapter 7

# Conclusions and research directions

In this chapter, we summarize the main contributions of this thesis and we propose some possible short and long term research directions which can be followed in future works.

## 7.1 Main contributions

The main goal of this thesis was to propose and analyze decomposition methods to solve hard quadratic programming problems. More specifically, we were interested in exploiting the tool of column generation for both continuous and discrete quadratic problems. In particular, we showed how the classic Dantzig-Wolfe approach can be applied in different contexts: in the first part of the thesis we considered convex programs, with linear constraints. We know that the DW decomposition can be seen as a generalization of the Frank-Wolfe technique, known as Simplicial Decomposition. We proposed an algorithm for both continuous and mixed binary problems, we carefully analysed its properties and we provided extensive computational results. In the second part, we tackled general problems, which can be non convex and have quadratic constraints. We concentrated on binary problems and we showed how DW can be applied to the lifted formulation and which consequences this can have on the relaxation. We provided theoretical and computational results.

**Part I** In the first part of the thesis, we considered continuous, convex quadratic problems. We introduced a new method to solve the master problem in a Simplicial Decomposition framework, which is specifically tailored for this type of problem. We proved its convergence and we compared its performance with another master method and the state of the art solver Cplex. With respect to the pricing, we introduced some improvements, namely an early stopping technique and shrinking cuts. We provided computational results based on a large set of instances, originated from literature problems and also randomly generated ones. We analyzed in detail the performance of our method, and showed that it is particularly efficient with respect to Cplex for a specific class of problems: those which have dense objective function and linear constraints, with a

combinatorial structure or with more variables than constraints. Problems with this form arise in literature and some examples are provided.

Based on the good results obtained in the continuous case, we tackled mixed binary quadratic problems, with a similar structure: the objective function is convex and dense, the constraints are linear and are less than the variables. We embedded the continuous algorithm in a branch and bound scheme. The algorithm remained effective because we exploited the structure of the B&B tree as well, in particular the depth first search strategy, which we implemented. With respect to the continuous case, the mixed binary problems we tested have a lower number of variables and constraints. We did not tackle more general convex problems, such as those with a non quadratic objective function, or with mixed integer variables; however, in principle our framework can be modified to solve such problems.

**Part II** In the second part of our work, we considered more general problems: the objective function and the constraints are quadratic, but can be non convex. We tackled purely binary problems. We applied the Dantzig-Wolfe decomposition to the lifted space of variables, we showed that the relaxation that we get lies in the Boolean Quadric Polytope and hence we proved that the bound obtained is better than classic bounds, as the SDP bound. We provided some preliminary results. It is worth noting that, for linearly constrained QPs, this technique could be enhanced with some standard techniques and provide not only a relaxation, but also a reformulation.

Then, we restricted to the case of block-decomposable problems, which represent a wide class of literature problems, since in most cases the data are sparse. We proposed another formulation with separate subproblems for every block of the structure, which are considerably smaller than the original problem. We provided results on the relation between this relaxation and the previous one. We proved that in one sense the relation is easy, but in the other direction it is an interesting problem related to the concept of matrix completion. We provided theoretical results for this problem and proposed a conjecture for the generic case. We provided computational results also for this algorithm, which show that this technique is really efficient for block-separable problems, while, if the problems have large overlaps, it shows some drawbacks and it should be improved.

In conclusion, our work shows that, in some classes of problems, decomposition algorithms, even based on classical ones, can be efficiently employed to solve quadratic problems and lead to good results. Moreover, the study on the BQP relaxation is interesting because it potentially links combinatorial, quadratic and conic optimization. We showed how BQP is related to CPP optimization, which is a continuously expanding and very active branch of research.

## 7.2 Future research

In this thesis, we presented theoretical and computational results. However, our research is not concluded and several results can be further improved and extended. Here, we present some possible research topics, for both the parts of the thesis, which are the

natural continuation of this discussion, and we also draw some lines which might be developed in future investigations.

**Part I** Regarding the SD-based algorithm, a subject of further improvement is surely the B&B for mixed binary problems. As already pointed out at the end of Chapter 4, several developments could be tested for this algorithm, whose efficiency has only been verified on some specific types of problem. We already explained some ideas, among which the first one to be tested is probably a clever preprocessing in which we compute a fixed set of conjugate directions; doing so, an option which has been very useful in [24], [25] and [26], is to fix the variable branching. In this way, it might be possible to reuse, in different nodes, the same directions and optimal points already available: hence, we could improve the computational results. Techniques to raise the lower bound at each node, exploiting the ellipsoids used in [25], could be introduced in the algorithm as well. In a long-term perspective, the algorithm could be used for nonquadratic but convex objective functions, for continuous problems. An example is given by the *perspective formulations* of quadratic or non quadratic problems (as studied in [66], [67], [68], [88]). The drawback for generic nonlinear functions is that the change of variables in the master, which in the quadratic case lead to a small-dimensional problem, in general cannot be done. Hence, probably the use of *restricted* or *nonlinear* simplicial decomposition, as described in [93] and [100] could be helpful to accelerate the convergence. As another future work, the B&B, here introduced for the mixed binary case, could be adapted to deal with more general mixed integer problems. In this case, also the projection of infeasible directions and extreme points shall be further studied and can lead to interesting results.

**Part II** Regarding the second part of the thesis, both theoretical and computational results shall be improved, in particular for the block-decomposable case. We shall first prove which is the specific class of graphs which are BQP-completable, and if the conjecture is actually true. This would be an important result, since it would prove that structured problems can be solved in a decomposed way. We think to start from the smallest intersection size in which the result is not known, that is dimension 3. We stated the problem as a linear programming feasibility problem, and we are trying to show that its dual problem is bounded.

In parallel, also several improvements on the computational side could be done, both for the master and the pricing subproblems. In particular, the master often contains a large number of coefficients which have a 0 weight in the expression of the optimal point. These variables are regularly deleted, but they have an impact on the dual coefficients and on the convergence. Hence, they should be deleted in a more efficient way. Moreover, in the master we have an explicit formulation, which consists of both the original $Y_j$ variables and the coefficients $\mu_j^l$. This formulation can be simplified, obtaining a master problem which contains the $Y_j$ variables only implicitly; it would have much less constraints and variables, so its optimization would become much easier. Furthermore, whenever a column is obtained by one pricing problem, if some coordinates of this column also belong to other blocks, it can be used to generate feasible columns for other blocks. This could

lead to a faster convergence. Tests on other sets of instances, with these improvements in the implementation, can be carried out. Moreover, in both the block-decomposition and the one-block formulation, we always solved master and pricing with Cplex. However, there are ad-hoc solvers for the pricing problems, since they are in max-cut form. An example of solver is BiqMac, an SDP-based B&B algorithm, described in [122]. Finally, the computation of the different pricing problems can be easily done in parallel.

As future research, we have several other ideas to work on: for instance, we could integrate our algorithm in a B&B approach, to obtain the solution of the problem, not only a relaxation. We also could study how relaxations of problems with inequalities, or mixed binary variables, can be modified to obtain a reformulation, if it is possible; this would extend the results provided in Section 5.4. One big complication is given by the fact that in the mixed binary case, the number of extreme points is not only infinite, but also uncountable. Nevertheless, we could test if good bounds can be obtained in a reasonable time. We can also extend the results to integer and maybe mixed integer problems. Adding the quadratization of linear constraints could also be done in the block-decomposable form, to improve the relaxation. However, if the constraints are dense, this destroys the sparse structure, so it could only be done if the constraints are sparse: otherwise to improve the value, a B&B is preferable.

In a longer-term perspective, we could address nonlinear non quadratic problems with quadratic approximations. Results obtained by these relaxations could let us achieve strong bounds in relatively short time, thus accelerating the convergence of the method. Finding a suitable sparse quadratic function which approximates a sparse nonlinear problem, and hence exploit the block properties, might be object of future research.

# Bibliography


[1] Farid Alizadeh and Donald Goldfarb. Second-order cone programming. *Mathematical programming*, 95(1):3–51, 2003.

[2] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

[3] Mokhtar S Bazaraa, Hanif D Sherali, and Chitharanjan M Shetty. *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2013.

[4] John E. Beasley. Portfolio optimization data, 2016.

[5] Abraham Berman. *Cones, matrices and mathematical programming*, volume 79. Springer Science & Business Media, 2012.

[6] Abraham Berman and Naomi Shaked-Monderer. *Completely positive matrices*. World Scientific, 2003.

[7] Abraham Berman and Changqing Xu. {0, 1} completely positive matrices. *Linear algebra and its applications*, 399:35–51, 2005.

[8] Avi Berman, Mirjam Dur, and Naomi Shaked-Monderer. Open problems in the theory of completely positive and copositive matrices. *Electronic Journal of Linear Algebra*, 29(1):46–58, 2015.

[9] Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

[10] Dimitri P Bertsekas and Athena Scientific. *Convex optimization algorithms*. Athena Scientific Belmont, 2015.

[11] Dimitri P Bertsekas and Huizhen Yu. A unifying polyhedral approximation framework for convex optimization. *SIAM Journal on Optimization*, 21(1):333–360, 2011.

[12] Enrico Bettiol, Lucas Létocart, Francesco Rinaldi, and Emiliano Traversi. A simplicial decomposition framework for large scale convex quadratic programming. *arXiv preprint arXiv:1705.09210*, 2017.

[13] Alain Billionnet, Sourour Elloumi, and Amélie Lambert. Extending the qcr method to general mixed-integer programs. *Mathematical programming*, 131(1-2):381–401, 2012.

[14] Alain Billionnet, Sourour Elloumi, and Marie-Christine 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.

[15] Ernesto G Birgin, José Mario Martínez, and Marcos Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000.

[16] Immanuel M Bomze. Copositive optimization–recent developments and applications. *European Journal of Operational Research*, 216(3):509–520, 2012.

[17] Immanuel M Bomze and Etienne De Klerk. Solving standard quadratic optimization problems via linear, semidefinite and copositive programming. *Journal of Global Optimization*, 24(2):163–185, 2002.

[18] Immanuel M Bomze, Mirjam Dür, Etienne De Klerk, Cornelis Roos, Arie J Quist, and Tamás Terlaky. On copositive programming and standard quadratic optimization problems. *Journal of Global Optimization*, 18(4):301–320, 2000.

[19] Immanuel M Bomze, Florian Jarre, and Franz Rendl. Quadratic factorization heuristics for copositive programming. *Mathematical Programming Computation*, 3(1):37–57, 2011.

[20] Immanuel M Bomze, Werner Schachinger, and Gabriele Uchida. Think co (mpletely) positive! matrix properties, examples and a clustered bibliography on copositive optimization. *Journal of Global Optimization*, 52(3):423–445, 2012.

[21] Pierre Bonami, Viet Hung Nguyen, Michel Klein, and Michel Minoux. On the solution of a graph partitioning problem under capacity constraints. In *International Symposium on Combinatorial Optimization*, pages 285–296. Springer, 2012.

[22] Brian Borchers. Csdp, ac library for semidefinite programming. *Optimization methods and Software*, 11(1-4):613–623, 1999.

[23] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[24] C. Buchheim and E. Traversi. Quadratic combinatorial optimization using separable underestimators. *INFORMS Journal on Computing*, 30(3):424–437, 2018.

[25] Christoph Buchheim, Alberto Caprara, and Andrea Lodi. An effective branch-and-bound algorithm for convex quadratic integer programming. *Mathematical programming*, 135(1-2):369–395, 2012.

[26] Christoph Buchheim, Marianna De Santis, Laura Palagi, and Mauro Piacentini. An exact algorithm for nonconvex quadratic integer minimization using ellipsoidal relaxations. *SIAM Journal on Optimization*, 23(3):1867–1889, 2013.

[27] Stefan Bundfuss and Mirjam Dür. An adaptive linear approximation algorithm for copositive programs. *SIAM Journal on Optimization*, 20(1):30–53, 2009.

[28] Samuel Burer. On the copositive representation of binary and continuous nonconvex quadratic programs. *Mathematical Programming*, 120(2):479–495, 2009.

[29] Samuel Burer. Optimizing a polyhedral-semidefinite relaxation of completely positive programs. *Mathematical Programming Computation*, 2(1):1–19, 2010.

[30] Samuel Burer. Copositive programming. In *Handbook on semidefinite, conic and polynomial optimization*, pages 201–218. Springer, 2012.

[31] Samuel Burer and Adam N Letchford. On nonconvex quadratic programming with box constraints. *SIAM Journal on Optimization*, 20(2):1073–1089, 2009.

[32] Samuel Burer and Adam N Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012.

[33] Rainer E Burkard, Stefan E Karisch, and Franz Rendl. Qaplib–a quadratic assignment problem library. *Journal of Global optimization*, 10(4):391–403, 1997.

[34] Michael R Bussieck, Arne Stolbjerg Drud, and Alexander Meeraus. Minlplib—a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003.

[35] Francesco Cesarone and Fabio Tardella. Portfolio datasets, 2010.

[36] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.

[37] Paul C Chu and John E Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.

[38] Laurent Condat. Fast projection onto the simplex and the l1-ball. *Mathematical Programming*, 158(1):575–585, 2016.

[39] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer programming*, volume 271. Springer, 2014.

[40] Andrea Cristofari. An almost cyclic 2-coordinate descent method for singly linearly constrained problems. *Computational Optimization and Applications*, 73(2):411–452, 2019.

[41] Andrea Cristofari, Marianna De Santis, Stefano Lucidi, and Francesco Rinaldi. A two-stage active-set algorithm for bound-constrained optimization. *J. Optim. Theory Appl.*, 172(2):369–401, 2017.

[42] Frank E Curtis, Zheng Han, and Daniel P Robinson. A globally convergent primal-dual active-set framework for large-scale convex quadratic optimization. *Computational Optimization and Applications*, 60(2):311–341, 2015.

[43] Geir Dahl and Torkel Andreas Haufmann. Zero-one completely positive matrices and the a (r, s) classes. *Special Matrices*, 4(1), 2016.

[44] George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[45] Etienne De Klerk and Dmitrii V Pasechnik. Approximation of the stability number of a graph via copositive programming. *SIAM Journal on Optimization*, 12(4):875–892, 2002.

[46] Marianna De Santis, Gianni Di Pillo, and Stefano Lucidi. An active set feasible method for large-scale minimization problems with bound constraints. *Computational Optimization and Applications*, 53(2):395–423, 2012.

[47] G Delaporte, S Jouteau, and F Roupin. Sdp s: A tool to formulate and solve semidefinite relaxations for bivalent quadratic problems (2002).

[48] Camil Demetrescu, Andrew V Goldberg, and David S Johnson. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74. American Mathematical Soc., 2009.

[49] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[50] Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

[51] Michel Marie Deza and Monique Laurent. *Geometry of cuts and metrics*, volume 15. Springer, 2009.

[52] Peter James Clair Dickinson. *The copositive cone, the completely positive cone and their generalisations.* Citeseer, 2013.

[53] Peter JC Dickinson. An improved characterisation of the interior of the completely positive cone. *Electron. J. Linear Algebra*, 20:723–729, 2010.

[54] Peter JC Dickinson. Geometry of the copositive and completely positive cones. *Journal of Mathematical Analysis and Applications*, 380(1):377–395, 2011.

[55] M. Djerdjour, K. Mathur, and H.M. Salkin. A surrogate relaxation based algorithm for a general quadratic multi-dimensional knapsack problem. *Operations Research Letters*, 7(5):253–258, 1988.

[56] Elizabeth D. Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[57] John Drake. Benchmark instances for the multidimensional knapsack problem, 01 2015.

[58] John H Drew and Charles R Johnson. The completely positive and doubly nonnegative completion problems. *Linear and Multilinear Algebra*, 44(1):85–92, 1998.

[59] Mirjam Dür. Copositive programming–a survey. In *Recent advances in optimization and its applications in engineering*, pages 3–20. Springer, 2010.

[60] Mirjam Dür and Georg Still. Interior points of the completely positive cone. *Electron. J. Linear Algebra*, 17:48–53, 2008.

[61] Jack Elzinga and Thomas G Moore. A central cutting plane algorithm for the convex programming problem. *Mathematical Programming*, 8(1):134–145, 1975.

[62] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpoases: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[63] Reeves Fletcher and Colin M Reeves. Function minimization by conjugate gradients. *The computer journal*, 7(2):149–154, 1964.

[64] Antonio Frangioni. Solving semidefinite quadratic problems within nonsmooth optimization algorithms. *Computers & Operations Research*, 23(11):1099–1118, 1996.

[65] Antonio Frangioni. Standard bundle methods: Untrusted models and duality. Technical report, Technical reports, Department of Informatics, University of Pisa, Italy . . . , 2018.

[66] Antonio Frangioni, Fabio Furini, and Claudio Gentile. Approximated perspective relaxations: a project and lift approach. *Computational Optimization and Applications*, 63(3):705–735, 2016.

[67] Antonio Frangioni and Claudio Gentile. Perspective cuts for a class of convex 0–1 mixed integer programs. *Mathematical Programming*, 106(2):225–236, 2006.

[68] Antonio Frangioni and Claudio Gentile. Sdp diagonalizations and perspective cuts for a class of nonseparable miqp. *Operations Research Letters*, 35(2):181–185, 2007.

[69] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics (NRL)*, 3(1-2):95–110, 1956.

[70] Mituhiro Fukuda, Masakazu Kojima, Kazuo Murota, and Kazuhide Nakata. Exploiting sparsity in semidefinite programming via matrix completion i: General framework. *SIAM Journal on Optimization*, 11(3):647–674, 2001.

[71] Fabio Furini, Emiliano Traversi, Pietro Belotti, Antonio Frangioni, Ambros Gleixner, Nick Gould, Leo Liberti, Andrea Lodi, Ruth Misener, Hans Mittelmann, et al. Qplib: A library of quadratic programming instances. *Optimization Online*, 5846, 2017.

[72] F. Glover, G.A. Kochenberger, B. Alidaee, and M. Amini. Solving quadratic knapsack problems by reformulation and tabu search: Single constraint case. In *Combinatorial and global optimization*, pages 111–121. World Scientific, 2002.

[73] Fred Glover and Gary A Kochenberger. Critical event tabu search for multidimensional knapsack problems. In *Meta-Heuristics*, pages 407–427. Springer, 1996.

[74] J-L Goffin, Jacek Gondzio, Robert Sarkissian, and J-P Vial. Solving nonlinear multicommodity flow problems by the analytic center cutting plane method. *Mathematical Programming*, 76(1):131–154, 1997.

[75] Jean-Louis Goffin and Jean-Philippe Vial. Cutting planes and column generation techniques with the projective algorithm. *Journal of optimization theory and applications*, 65(3):409–429, 1990.

[76] Jean-Louis Goffin and Jean-Philippe Vial. On the computation of weighted analytic centers and dual ellipsoids with the projective algorithm. *Mathematical Programming*, 60(1):81–92, 1993.

[77] J. Gondzio, O. du Merle, R. Sarkissian, and J.-P. Vial. Accpm — a library for convex optimization based on an analytic center cutting plane method. *European Journal of Operational Research*, 94(1):206 – 211, 1996.

[78] Jacek Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3):587–601, 2012.

[79] Jacek Gondzio and Pablo González-Brevis. A new warmstarting strategy for the primal-dual column generation method. *Mathematical Programming*, 152(1-2):113–146, 2015.

[80] Jacek Gondzio, Pablo González-Brevis, and Pedro Munari. New developments in the primal–dual column generation technique. *European Journal of Operational Research*, 224(1):41–51, 2013.

[81] Jacek Gondzio, Pablo González-Brevis, and Pedro Munari. Large-scale optimization with the primal-dual column generation method. *Mathematical Programming Computation*, 8(1):47–82, Mar 2016.

[82] Jacek Gondzio and Roy Kouwenberg. High-performance computing for asset-liability management. *Operations Research*, 49(6):879–891, 2001.

[83] Jacek Gondzio, Robert Sarkissian, and J-P Vial. Using an interior point method for the master problem in a decomposition approach. *European Journal of Operational Research*, 101(3):577–587, 1997.

[84] Jacek Gondzio, Jean-Philippe Vial, et al. Warm start and -subgradients in a cutting plane scheme for block-angular linear programs. *Computational Optimization and Applications*, 14:17–36, 1999.

[85] Nicholas IM Gould and Philippe L Toint. A quadratic programming bibliography. *Numerical Analysis Group Internal Report*, 1:32, 2000.

[86] Luigi Grippo, Francesco Lampariello, and Stefano Lucidi. A nonmonotone line search technique for newton's method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986.

[87] Robert Grone, Charles R Johnson, Eduardo M Sá, and Henry Wolkowicz. Positive definite completions of partial hermitian matrices. *Linear algebra and its applications*, 58:109–124, 1984.

[88] Oktay Günlük and Jeff Linderoth. Perspective reformulations of mixed integer nonlinear programs with indicator variables. *Mathematical programming*, 124(1-2):183–205, 2010.

[89] Nebojša Gvozdenović and Monique Laurent. The operator $\psi$ for the chromatic number of a graph. *SIAM Journal on Optimization*, 19(2):572–591, 2008.

[90] William W Hager and Hongchao Zhang. A new active set algorithm for box constrained optimization. *SIAM J. Optim.*, 17(2):526–557, 2006.

[91] Marshall Hall and Morris Newman. Copositive and completely positive quadratic forms. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 59, pages 329–339. Cambridge University Press, 1963.

[92] Peter L Hammer and Abraham A Rubin. Some remarks on quadratic programming with 0-1 variables. *Revue française d'informatique et de recherche opérationnelle. Série verte*, 4(V3):67–79, 1970.

[93] Donald W Hearn, S Lawphongpanich, and Jose A Ventura. Restricted simplicial decomposition: Computation and extensions. *Computation Mathematical Programming*, pages 99–118, 1987.

[94] Charles A Holloway. An extension of the frank and wolfe method of feasible directions. *Mathematical Programming*, 6(1):14–27, 1974.

[95] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[96] Dennis Huisman, Raf Jans, Marc Peeters, and Albert PM Wagelmans. Combining column generation and lagrangian relaxation. In *Column generation*, pages 247–270. Springer, 2005.

[97] IBM. Cplex (version 12.6.3), 2016.

[98] Krzysztof C Kiwiel. *Methods of descent for nondifferentiable optimization*, volume 1133. Springer, 2006.

[99] Nathan Krislock, Jérôme Malick, and Frédéric Roupin. Biqcrunch: A semidefinite branch-and-bound method for solving binary quadratic problems. *ACM Transactions on Mathematical Software (TOMS)*, 43(4):32, 2017.

[100] Torbjörn Larsson, Michael Patriksson, and Clas Rydergren. Applications of simplicial decomposition with nonlinear column generation to nonlinear network flows. In *Network optimization*, pages 346–373. Springer, 1997.

[101] Claude Lemaréchal. Chapter vii nondifferentiable optimization. *Handbooks in operations research and management science*, 1:529–572, 1989.

[102] Claude Lemaréchal. The omnipresence of lagrange. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(1):7–25, 2003.

[103] Adam N Letchford and Michael M Sørensen. Binary positive semidefinite matrices and associated integer polytopes. *Mathematical programming*, 131(1-2):253–271, 2012.

[104] A Yu Levin. On an algorithm for the minimization of convex functions. In *Soviet Mathematics Doklady*, volume 160, pages 1244–1247, 1965.

[105] Marco E Lubbecke and Jacques Desrosiers. Selected topics in column generation. *OPERATIONS RESEARCH-BALTIMORE THEN LINTHICUM-*, 53(6):1007, 2005.

[106] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[107] John E Maxfield and Henryk Minc. On the matrix equation $X'X = A$. *Proceedings of the Edinburgh Mathematical Society*, 13(2):125–129, 1962.

[108] Christian Michelot. A finite algorithm for finding the projection of a point onto the canonical simplex of $\mathbb{R}^n$. *Journal of Optimization Theory and Applications*, 50(1):195–200, 1986.

[109] Pedro Munari and Jacek Gondzio. Using the primal-dual interior point algorithm within the branch-price-and-cut method. *Computers & Operations Research*, 40(8):2026–2036, 2013.

[110] Karthik Natarajan, Chung Piaw Teo, and Zhichao Zheng. Mixed 0-1 linear programs under objective uncertainty: A completely positive representation. *Operations research*, 59(3):713–728, 2011.

[111] George L Nemhauser. Column generation for linear and integer programming. *Optimization Stories*, 20:64, 2012.

[112] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[113] Donald J Newman. Location of the maximum on unimodal surfaces. *Journal of the ACM (JACM)*, 12(3):395–398, 1965.

[114] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[115] Jorge Nocedal and Stephen J Wright. *Sequential quadratic programming*. Springer, 2006.

[116] Ivo Nowak. *Relaxation and decomposition methods for mixed integer nonlinear programming*, volume 152. Springer Science & Business Media, 2005.

[117] Manfred Padberg. The boolean quadric polytope: some characteristics, facets and relatives. *Mathematical programming*, 45(1-3):139–172, 1989.

[118] Michael Patriksson. *The traffic assignment problem: models and methods*. Courier Dover Publications, 2015.

[119] Itamar Pitowsky. Correlation polytopes: their geometry and complexity. *Mathematical Programming*, 50(1-3):395–414, 1991.

[120] Florian A Potra and Stephen J Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1-2):281–302, 2000.

[121] Janez Povh and Franz Rendl. Copositive and semidefinite relaxations of the quadratic assignment problem. *Discrete Optimization*, 6(3):231–241, 2009.

[122] Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming*, 121(2):307, 2010.

[123] Donald J Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(3):597–609, 1970.

[124] Jo Bo Rosen. The gradient projection method for nonlinear programming. part i. linear constraints. *Journal of the society for industrial and applied mathematics*, 8(1):181–217, 1960.

[125] B. Rostami, F. Malucelli, D. Frey, and C. Buchheim. On the quadratic shortest path problem. In *International Symposium on Experimental Algorithms*, pages 379–390. Springer, 2015.

[126] Borzou Rostami, André Chassein, Michael Hopf, Davide Frey, Christoph Buchheim, Federico Malucelli, and Marc Goerigk. The quadratic shortest path problem: complexity, approximability, and solution methods. *European Journal of Operational Research*, 268(2):473–485, 2018.

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

[128] Jos F Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625–653, 1999.

[129] Siddhartha S. Syam. A dual ascent method for the portfolio selection problem with multiple constraints and linked proposals. *European Journal of Operational Research*, 108(1):196 – 207, 1998.

[130] S.P. Tarasov, L.G. Khachiian, and I.I. Erlikh. The method of inscribed ellipsoids. *Doklady Akademii Nauk SSSR*, 298(5):1081–1085, 1988.

[131] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[132] Reha H Tütüncü, Kim-Chuan Toh, and Michael J Todd. Solving semidefinite-quadratic-linear programs using sdpt3. *Mathematical programming*, 95(2):189–217, 2003.

[133] Wim Van Ackooij and Antonio Frangioni. Incremental bundle methods using upper models. *SIAM Journal on Optimization*, 28(1):379–410, 2018.

[134] Wim van Ackooij, Antonio Frangioni, and Welington de Oliveira. Inexact stabilized benders' decomposition approaches with application to chance-constrained problems with finite support. *Computational Optimization and Applications*, 65(3):637–669, 2016.

[135] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM review*, 38(1):49–95, 1996.

[136] François Vanderbeck. On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48(1):111–128, 2000.

[137] François Vanderbeck. Implementing mixed integer column generation. In *Column generation*, pages 331–358. Springer, 2005.

[138] François Vanderbeck and Laurence A Wolsey. An exact algorithm for ip column generation. *Operations research letters*, 19(4):151–159, 1996.

[139] Jose A Ventura and Donald W Hearn. Restricted simplicial decomposition for convex constrained problems. *Mathematical Programming*, 59(1):71–85, 1993.

[140] Balder Von Hohenbalken. Simplicial decomposition in nonlinear programming algorithms. *Mathematical Programming*, 13(1):49–68, 1977.

[141] WolframAlpha. Mathematica (version 11.3), 2016.

[142] Margaret Wright. The interior-point revolution in optimization: history, recent developments, and lasting consequences. *Bulletin of the American mathematical society*, 42(1):39–56, 2005.

[143] Stephen J Wright. *Primal-dual interior-point methods*. SIAM, 1997.

[144] Yinyu Ye. *Interior point algorithms: theory and analysis*, volume 44. John Wiley & Sons, 2011.

[145] Günter M Ziegler. Lectures on 0/1-polytopes. In *Polytopes—combinatorics and computation*, pages 1–41. Springer, 2000.