Continuous Optimization Methods for Convex Mixed-Integer Nonlinear Programming

Dissertation

zur
Erlangung des Grades
eines
Doktors der Naturwissenschaften

Der Fakultät für Mathematik
der Technischen Universität Dortmund
vorgelegt von

Long Trieu

aus Wuppertal

Dortmund 2015

Die vorliegende Arbeit wurde von der Fakultät für Mathematik der Technischen Universität Dortmund als Dissertation zur Erlangung des Grades eines Doktors der Naturwissenschaften genehmigt.

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Ben Schweizer
Erster Gutachter:	Prof. Dr. Christoph Buchheim
Zweiter Gutachter:	Prof. Dr. Christian Meyer
Dritter Prüfer:	Prof. Dr. Peter Recht

Tag der Einreichung:	07.Mai 2015
Tag der Disputation:	26.August 2015

Abstract

The topic of this dissertation is the design of fast branch-and-bound algorithms that use intelligently adapted approaches from continuous optimization for solving convex mixed-integer nonlinear programming problems. This class of optimization problems is NP-hard and polynomial-time algorithms for these problems are therefore unlikely to exist (unless P=NP). The importance of this class is highlighted by the fact that many real-world applications can be modeled as a (convex) mixed-integer nonlinear programming problem. Currently, there are several standard techniques such as outer approximation that are used within recent state-of-the-art software. Although all these algorithms include sophisticated improvements such as primal heuristics and effective preprocessing, they do not take into account the large gap between the algorithmic performance of NLP and IP solvers. While NLP solvers are well-engineered for large-scale problems, MIP problems of similar sizes are by far harder to solve in practice. Therefore, when using NLP techniques within MIP solvers, these NLP algorithms have to be adjusted to handle small-size instances effectively.

Taking this problem into account, we present three branch-and-bound algorithms, based on a former work by Buchheim et al. [43] on unconstrained convex quadratic integer programming problems. The main strategies used within this branch-and-bound framework include extensive preprocessing and fast incremental computations, aiming at a very fast enumeration of the nodes. The first algorithm we present is designed to solve convex quadratic mixed-integer programming problems with linear inequality constraints and is based on a new feasible active set algorithm applied to the dual of the continuous relaxation. This active set algorithm is tailored for the continuous problem and fully exploits its structure. Furthermore, a warmstarting procedure is used to reduce the number of active set iterations per node. The second algorithm we introduce is an approach called quadratic outer approximation for solving box-constrained convex mixed-integer nonlinear programming problems. It extends the classical outer approximation by using quadratic underestimators leading to a faster convergence in practice. Finally, the last algorithm we devise is aimed at a class of mean-risk portfolio optimization problems that can be modeled as convex mixed-integer programming problems with a single linear budget constraint. For this application we propose a branch-and-bound scheme using a modified Frank-Wolfe type algorithm to solve the node relaxations. Similarly to the branch-and-bound algorithms mentionded above we exploit the simplicity of the relaxations to enumerate the nodes as quickly as possible rather than focussing on strong dual bounds.

We implemented all three algorithms and compared their performance with several state-of-the art approaches. Our extensive computational studies show that all new approaches presented in this thesis are able to effectively solve large classes of real-world instances.

Zusammenfassung

Diese Arbeit befasst sich mit dem Entwurf von schnellen branch-and-bound Algorithmen zur Lösung von konvexen gemischt-ganzzahligen nichtlinearen Optimierungsproblemen. Solche Probleme sind NP-schwer, so dass es dafür unter der Annahme P≠NP keine effizienten exakten Algorithmen gibt. Aktuell gibt es etablierte Standardverfahren, wie z.B. outer approximation, welche in den modernsten Software-Paketen implementiert sind. Obwohl diese Verfahren ausgereifte algorithmische Techniken benutzen, berücksichtigen sie dennoch nicht, dass die verwendeten Algorithmen der nichtlinearen und der gemischt-ganzzahligen Optimierung ursprünglich für Probleme verschiedener Größe konzipiert wurden. Während NLP Löser darauf spezialisiert sind Instanzen mit einer großen Anzahl an Variablen und Nebenbedingungen zu lösen, können in der aktuellen Forschung keine vergleichbar großen Instanzen für gemischt-ganzzahlige Probleme gelöst werden. Da die Effektivität der Löser für gemischt-ganzzahlige Probleme in der Regel auch von denen der NLP Löser abhängt, führt dies dazu, dass letztere auch speziell für kleine Instanzen angepasst werden müssen.

Ausgehend von dieser Herausforderung, stellen wir insgesamt drei branch-and-bound Verfahren vor, basierend auf einem exakten Algorithmus von Buchheim et al. [43] für unrestringierte konvexe quadratische ganzzahlige Probleme. Die Hauptkomponenten des Verfahrens sind ein ausgereiftes Preprocessing sowie eine schnelle inkrementelle Berechnung der dualen Schranken. Dies führt zu einer sehr effektiven Enumerierung des Suchbaumes. Der erste branch-and-bound Algorithmus, den wir vorstellen, löst konvexe quadratische gemischt-ganzzahlige Probleme mit linearen Ungleichungen und basiert auf einer Aktive-Mengen Strategie, welche auf das duale Problem der kontinuierlichen Relaxierung angewendet wird. Dieser Algorithmus ist besonders effektiv, da er die spezielle Struktur der quadratischen Teilprobleme ausnutzt. Ferner präsentieren wir einen Ansatz um konvexe gemischt-ganzzahlige nichtlineare Probleme mit Variablenschranken zu lösen. Dieser Ansatz erweitert die klassische outer approximation Methode und nutzt anstelle von Linearisierungen quadratische Unterschätzer. Durch die bessere Approximation der Zielfunktion kann dies bei bestimmten Instanzklassen zu einer schnelleren Konvergenz in der Praxis führen. Schließlich entwerfen wir einen branch-and-bound Algorithmus zur Lösung von speziellen Problemen der Portfolio-Optimierung, die ebenfalls als konvexe gemischt-ganzzahlige Probleme modelliert werden können. Für diese Anwendung entwerfen wir einen modifizierten Frank-Wolfe Algorithmus, der erneut gezielt die Struktur der zulässigen Menge ausnutzt.

Alle drei Algorithmen wurden implementiert und in einer experimentellen Studie mit ausgewählten aktuellen Lösern verglichen. Die Ergebnisse zeigen, dass alle vorgestellten Algorithmen dieser Arbeit in der Lage sind eine große Anzahl an praxisorientierten Instanzen effektiv zu lösen.

Partial Publications and Collaboration Partners

All research results presented in this dissertation have been worked out under the supervision of Christoph Buchheim. Partial results on the active set based branch-and-bound algorithm in Chapter 5 have been published in [42]. Further extensions of this chapter were developed in a productive cooperation with Marianna De Santis (TU Dortmund), Francesco Rinaldi (University of Padua) and Stefano Lucidi (Sapienza University of Rome). The results on the quadratic outer approximation scheme in Chapter 6 have been published in [41]. Furthermore, the results on the Frank-Wolfe based branch-and-bound algorithm in Chapter 7 were elaborated together with Marianna De Santis and Francesco Rinaldi.

Acknowledgements

„Was wäre ich denn, wenn ich nicht immer mit klugen Leuten umgegangen wäre und von ihnen gelernt hätte?“
Johann Wolfgang von Goethe, German writer and statesman

First of all, I would like to thank my supervisor Prof. Christoph Buchheim for his constant scientific support throughout the years at Dortmund. I am very thankful for being part of his fantastic working group and I fully enjoyed the casual but at the same time motivational atmosphere created by the group.

It was a pleasure for me to work with every single one of my colleagues during this period. I did not only profit from their professional expertise, but I also really appreciated their non-scientific skills that enriched my time outside working hours. I am really honored to have been part of LSV. Especially, I thank Frank Baumann for teaching me many very useful tricks in coding; Viktor Bindewald for showing me the best activities whenever I needed some timeout from work; Marianna De Santis for perfectly answering any of my questions concerning nonlinear optimization; Anna Ilyina for the good times during the conferences; Laura Klein for sharing all the good but also hard moments during almost my complete time at the chair; Jannis Kurtz for taking care of the constant coffee production and Maribel Montenegro for helping me whenever it was important. Special thanks are also devoted to Sabine Willrich for absolutely helping me with all administrative issues and always conveying positive mood to the working group.

Moreover, many thanks go to my co-authors Stefano Lucidi, Francesco Rinaldi and Marianna De Santis who were significantly involved in the results of Chapter 5 and 7 of this thesis.

Furthermore, I would like to thank Frank Baumann, Marianna De Santis and Daniel Heubes for carefully proofreading my thesis.

Finally, I would like to thank my family and Laura for always being there for me.

Contents

List of Frequently Abbreviated Problems

CBIP	Convex Box-Constrained Integer Programming
CBMIQP	Convex Box-Constrained Mixed-Integer Quadratic Programming
CMINLP	Convex Mixed-Integer Nonlinear Programming
CMIQP	Convex Mixed-Integer Quadratic Programming
CP	Convex Programming
EQP	Equality-Constrained Quadratic Programming
GCBP	Generalized Capital Budgeting Problem
INLP	Integer Nonlinear Programming
MILP	Mixed-Inter Linear Programming
MINLP	Mixed-Integer Nonlinear Programming
MIQCP	Mixed-Integer Quadratically-Constrained Programming
MIQP	Mixed-Integer Quadratic Programming
NLP	Nonlinear Programming
NLP-R	Nonlinear Programming Relaxation
QP	Quadratic Programming
UCQIP	Unconstrained Convex Quadratic Integer Programming
UNLP	Unconstrained Nonlinear Programming
UQP	Unconstrained Quadratic Programming

Chapter 1

Introduction

In this thesis we are dealing with *Mixed-Integer Nonlinear Programming* (MINLP) problems involving both integer and continuous variables. In addition to the difficulty caused by the integrality, the need to handle the nonlinearity of the functions increases their complexity even more. One reason why in particular optimization over integers is more difficult than continuous optimization is that, except for a very few special cases, there are no optimality conditions for integer programming problems. Proving optimality is therefore much more difficult than in the continuous case. One possible approach would be to enumerate all finitely many feasible points, but this is not feasible due to the combinatorial explosion of the feasible set. Another reason is that integrality automatically implies non-convexity of the feasible region. Thus, although we are concerned with convex objective and constraint functions in this thesis, we are dealing implicitly with non-convex feasible regions. While convexity in continuous optimization implies that every local minimum is also a global minimum, this is no longer true if we require integrality of the solution. A famous quote that reflects the role of non-convexity in optimization is given by the renowned mathematician Ralph Tyrrell Rockafellar. In a SIAM review survey paper from 1993, he argued that

> *"the great watershed in optimization isn't between linearity and nonlinearity, but convexity and non-convexity".*

The general Mixed-Integer Nonlinear Programming problem has the following formulation:

$$\begin{aligned} \min_{x,y}\ & f(x,y) \\ \text{s.t. } & g(x,y) \le 0 \\ & x \in X \cap \mathbb{Z}^n,\ y \in Y, \end{aligned} \tag{MINLP}$$

where $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}$ is the nonlinear objective function, $g : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ models the nonlinear constraints and $X \subseteq \mathbb{R}^n$, $Y \subseteq \mathbb{R}^p$. We assume f and g to be twice continuously differentiable and X, Y to be bounded. In case the feasible region is bounded, MINLP belongs to the class of NP-hard problems, since it contains *Mixed-Integer Linear Programming* (MILP) as a special case, which is known to be NP-hard itself [118]. If the feasible region is unbounded, MINLP is even undecidable [116]. Nevertheless, in practice, the feasible region is usually bounded. Therefore MINLP is an extremely difficult class of optimization problems. A comprehensive study on the computational complexity of optimization problems can be found in [89]. If both f and g are convex functions and X, Y are convex sets, MINLP is called convex, otherwise non-convex. While solving convex MINLPs is already very challenging, non-convex MINLPs are much more difficult to solve since even their continuous relaxation can be NP-hard itself (see, e.g., [180, 182]), due to the possibility of local minima. If $p = 0$ MINLP becomes a *Pure Integer Nonlinear Programming* problem:

$$\min\{f(x) \mid g(x) \leq 0,\ x \in X \cap \mathbb{Z}^n\}. \tag{INLP}$$

INLP itself can be classified into different subclasses according to the special structure of the objective and constraint functions. We mention three special cases.

- *Unconstrained Integer Programming:* All inequality constraints are absent ($m = 0$) and $X = \mathbb{R}^n$.
- *Quadratic Integer Programming:* The objective function f is quadratic and the constraint function g is linear.
- *Convex Integer Programming:* Both functions f and g are convex.

Combining all three subclasses above, we get an *Unconstrained Convex Quadratic Integer Programming* problem

$$\min\{x^\top Qx + c^\top x + d \mid x \in \mathbb{Z}^n\}, \tag{UCQIP}$$

where $Q \in \mathbb{R}^{n\times n}$ is symmetric and positive semidefinite, $c \in \mathbb{R}^n$ and $d \in \mathbb{R}$. Note that, in spite of all restrictions to INLP, UCQIP is already NP-hard, since it is equivalent to the *Closest Vector Problem*, see [184]. In Chapter 4, we will recall a tailored branch-and-bound algorithm for solving a box-constrained version of UCQIP, developed by Buchheim et al. [43], which will be the fundamental branch-and-bound scheme for all the other algorithms in this thesis.

There is an endless number of real-world problems which can be formulated as mixed-integer nonlinear programming problems. They include portfolio optimization [29], block layout design in the manufacturing and service sectors [50], network design with queuing delay constraints [38], integrated design and control of chemical processes [83], drinking water distribution systems security [131], minimizing the environmental impact of utility plants [76], and multiperiod supply chain problems subject to probabilistic constraints [136]. A very detailed overview of applications in the field of engineering is given by Grossmann and Sahinidis [102, 103]. MINLP problems are therefore of great interest in mathematical optimization. In 2008, Jon Lee used the term *"the mother of all deterministic optimization problems"* to describe the importance of MINLP. A comprehensive survey on applications, models and solution methods, both for non-convex and convex MINLP is given by Belotti et al. [19].

Contributions

In this thesis, we present new branch-and-bound algorithms for solving convex mixed-integer nonlinear programming problems that combine well known methods from nonlinear optimization with a special branch-and-bound scheme. Algorithms for convex MINLP often strongly rely on the effectiveness of nonlinear programming (NLP) solvers, since the original mixed-integer problem is usually tackled by solving a sequence of NLP subproblems that occur as relaxations. Currently, theory and algorithms for nonlinear programming are still constantly improving. Yet, recent advances already allow state-of-the art NLP solvers to handle large-scale problems with several thousands of variables and/or constraints. In contrast, unlike in nonlinear programming the algorithmic advances in integer programming are still lagging behind such that solving general integer problems of about 100 variables can be already very challenging. Therefore a key task in designing fast MINLP solvers consists also in the reengineering of existing NLP solvers to accelerate the process of finding solutions to small-size problems. In particular, it is necessary to fully exploit the given nonlinear structure by appropriate methods instead of typically just linearizing the nonlinearities. Another important concept is the use of warmstarts within the branch-and-bound scheme, leading to a further reduction of iterations within each node and finally to a faster enumeration in the search tree. The main concept and contribution of this thesis is the design of new, but also the reengineering of existing nonlinear programming algorithms. The effective embedding of the tailored NLP solvers into a general branch-and-bound framework is done by the use of a sophisticated preprocessing phase, yielding a very effective overall enumeration process in the search tree. Following this philosophy, we devise three branch-and-bound algorithms for convex quadratic mixed-integer programming problems (Chapter 5), box-constrained convex nonlinear mixed-integer programming problems (Chapter 6) and, as an application, one class of mixed-integer mean-risk portfolio optimization problems (Chapter 7). All three algorithms have in common that they are based on the same quick branch-and-bound scheme combined with finely tuned NLP solvers.

Outline

In Chapter 2, we introduce the basic concepts of nonlinear programming. We give optimality conditions for the relevant problems to be discussed, summarize known results from Lagrangian duality theory, convex and quadratic programming. We furthermore give a brief overview of the nonlinear programming methods we will use in order to solve the nonlinear programming relaxations in our branch-and-bound algorithms.

Chapter 3 deals with the basic theory of convex mixed-integer nonlinear programming. First, we discuss common approaches for mixed-integer linear programming and show how they have been used to design algorithms for their nonlinear counterpart. We present the fundamental methodology that is needed and shortly survey the existing state-of-the-art algorithms. At the end a short presentation of the most common software packages is given.

In Chapter 4, a branch-and-bound scheme proposed by Buchheim et al. for box-constrained convex quadratic mixed-integer programming problems is formulated, on which the upcoming algorithms are based. The most important components, namely the preprocessing phase and the fast incremental computation technique for the dual bounds, are presented. Finally, results on achieving linear running time per node are given.

Chapter 5 presents a novel generalization of the latter scheme to mixed-integer quadratic programming problems with linear inequalities. We show that the theoretical complexity of computing dual bounds for this generalized class of problems does not increase if the number of linear inequalities is considered as fixed. An active set based algorithm is devised to compute the dual bounds effectively. Additionally, the crucial concepts of using duality and warmstarts within Branch-and-Bound are investigated.

In Chapter 6 we formulate an extension of the standard outer approximation approach for solving box-constrained convex mixed-integer nonlinear programming problems by using quadratic underestimators instead of linearizations. We give a sufficient condition for the existence of a global quadratic underestimator and illustrate the potential of the scheme by studying a special class of exponential objective functions for which underestimators can be derived explicitly.

Chapter 7 presents a branch-and-bound algorithm for solving a generalization of the risk-averse capital budgeting problem where the investor can adjust his own risk profile by choosing an appropriate risk function. This application is modeled as a convex mixed-integer optimization problem. We design a modified Frank-Wolfe type algorithm to compute dual bounds within our branch-and-bound framework. Again, duality and warmstart properties are exploited to speed up the algorithm.

Finally, we give summary and a short outlook about future research perspectives.

Chapter 2

Nonlinear Programming

A general nonlinear optimization problem is given by

$$\min\{f(x) \mid x \in \mathcal{F}\} \tag{NLP}$$

where $\emptyset \neq \mathcal{F} \subseteq \mathbb{R}^n$ is the feasible region and $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function we want to minimize. In the further course of this section, unless explicitly indicated, we assume f to be continuously differentiable. If $\mathcal{F} = \mathbb{R}^n$, the problem NLP is called *unconstrained*, otherwise *constrained.*

We call a vector $x \in \mathbb{R}^n$ *feasible* for NLP if $x \in \mathcal{F}$. If $x^\star$ is feasible and

$$f(x^\star) \leq f(x) \; \forall \; x \in F, \tag{2.1}$$

$x^\star$ is called a *(global) minimizer* of NLP.
If there exists some $r > 0$, such that for all $x \in F \cap B_r(x^\star)$

$$f(x^\star) \leq f(x), \tag{2.2}$$

$x^\star$ is called a *(local) minimizer* of NLP. Here, $B_r(x^\star)$ denotes the closed ball of radius r centered in $x^\star$.

If in (2.1) and (2.2) strict inequality holds for all $x \neq x^\star$, $x^\star$ is called a *strict (global/local) minimizer.* Each global minimizer is by definition also a local minimizer. The corresponding objective function value $f(x^\star)$ is called a *(global/local) minimum.*

2.1 Existence of Global Minima

A well-known result from calculus, the Extreme Value Theorem of Weierstrass, states that, for any given compact set $\mathcal{F}$, a continuous function $f : \mathcal{F} \to \mathbb{R}$ attains its global minimum on $\mathcal{F}$. Although we are going to deal only with smooth functions in this thesis, we note that this result can be extended and generalized also to non-continuous functions and non-compact sets, using the following definitions: any real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ is called *lower semicontinuous* at $x \in \mathbb{R}^n$ if $\liminf_{k\to\infty} f(x_k) \geq f(x)$ for every sequence $\{x_k\} \subset \mathbb{R}^n$ that converges to x. It is called *coercive* if for every sequence $\{x_k\} \subset \mathbb{R}^n$ such that $||x_k|| \to \infty$ we have $\lim_{k\to\infty} f(x_k) = \infty$.

Theorem 2.1 (Weierstrass' Theorem [26]). *Let $\mathcal{F} \subseteq \mathbb{R}^n$ be a nonempty set and $f : \mathcal{F} \to \mathbb{R}$ be lower semicontinuous on $\mathcal{F}$. Assume that one of the following conditions holds:*

(i) $\mathcal{F}$ is compact.

(ii) $\mathcal{F}$ is closed and f is coercive.

(iii) There exists $z \in \mathcal{F}$ such that the level set

$$\mathcal{N}_f(z) = \{x \in \mathcal{F} \mid f(x) \leq f(z)\}$$

is nonempty and compact.

Then the set of minimizers of f over $\mathcal{F}$ is nonempty and compact.

Example 2.1. We consider the minimization of the strictly convex quadratic function $f(x) := \frac{1}{2}x^\top Qx + c^\top x + d$, where $Q \in \mathbb{R}^{n\times n}$ is symmetric and positive definite, $c \in \mathbb{R}^n$ and $d \in \mathbb{R}$. It is easy to see that from the coercivity of f there exists some $z \in \mathbb{R}^n$ sufficiently large such that level set $\mathcal{N}_f(z)$ is nonempty and compact. Hence, every strictly convex quadratic function attains its global minimum on $\mathbb{R}^n$.

2.2 Optimality Conditions

In this section we introduce necessary and sufficient optimality conditions for nonlinear optimization problems. These conditions are used to identify local minima. Necessary conditions are satisfied in every local minimizer, while sufficient conditions imply local optimality, if they are satisfied. Characterizations of minimizers are both important in theory and from a practical point of view for designing effective optimization algorithms. We first take a look at optimality conditions for unconstrained nonlinear optimization problems and extend the theory to the presence of nonlinear constraints.

2.2.1 Unconstrained Nonlinear Optimization Problems

An unconstrained nonlinear optimization problem is of the form

$$\min\{f(x) \mid x \in \mathbb{R}^n\}. \tag{UNLP}$$

Note that even if the problem is unconstrained, the minimization of an arbitrary nonlinear function may be hard, since it might have several local minima. Nevertheless, every local minimizer of UNLP satisfies the following first-order necessary condition.

Theorem 2.2 (First-Order Necessary Condition [26]). *If x^* is a local minimizer of UNLP, then*

$$\nabla f(x^\star) = 0. \tag{2.3}$$

Solutions of (2.3) are called *stationary points* of f. This means when searching for minima of f, we can reduce the search space to all its stationary points. However, in general this is equivalent to solving a system of nonlinear equations which can be done analytically only for special classes of functions, for example if f is quadratic. In this case (2.3) reduces to a system of linear equations. If we assume f to be additionally twice continuously differentiable, we can use second-order information to obtain a necessary optimality condition.

Theorem 2.3 (Second-Order Necessary Condition [26]). *If $x^\star$ is a local minimizer of UNLP, then $\nabla^2 f(x^\star)$ is positive semidefinite.*

If f is convex, (2.3) is also sufficient for optimality, see Section 2.3.2. For a general function f that is not necessarily convex, any stationary point at which the Hessian of f is positive definite is a strict local minimizer.

Theorem 2.4 (Second-Order Sufficient Conditions [26]). *If $\nabla f(x^\star) = 0$ and $\nabla^2 f(x^\star)$ is positive definite, then $x^\star$ is a strict local minimizer of UNLP.*

Note that the second-order necessary and sufficient conditions differ only slightly, but the second-order sufficient conditions even imply strict local minimality.

2.2.2 Constrained Nonlinear Optimization Problems

In this section, we consider general nonlinear optimization problems NLP. We assume that their feasible region can be described by a set of m inequalities and p equations, i.e.,

$$\mathcal{F} = \{x \in \mathbb{R}^n \mid g(x) \leq 0,\ h(x) = 0\}, \tag{2.4}$$

where $g : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^p$ are assumed to be continuously differentiable. We point out that there exist many optimality conditions for generally constrained nonlinear programming problems in the literature. Instead of giving a complete overview, we focus on optimality conditions that will be exploited in the following to design practical algorithms. We start with some basic definitions.

Definition 2.1 (Lagrangian)**.** The *Lagrangian* of NLP is defined as the function $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$,

$$L(x, \lambda, \mu) := f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{p} \mu_j h_j(x). \tag{2.5}$$

The vectors $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^p$ are called *Lagrangian Multipliers.*

Definition 2.2 (KKT-Conditions)**.** The system

$$\begin{aligned} \nabla_x L(x, \lambda, \mu) &= 0 && \text{(Stationarity)} \\ g(x) \leq 0, \ h(x) &= 0 && \text{(Primal Feasibility)} \\ \lambda \geq 0, \ \lambda^\top g(x) &= 0 && \text{(Complementary Slackness)} \end{aligned}$$

is called *Karush-Kuhn-Tucker (KKT)-system* of NLP. Any solution of the KKT-system is called *KKT-point.*

Definition 2.3 (Active and Non-active Set)**.** For any $x \in \mathcal{F}$ the *active set* at x is the set $\mathcal{A}(x) := \{i \in \{1, \dots, m\} \mid g_i(x) = 0\}$ and the *non-active set* at x is the set $\mathcal{N}(x) := \{i \in \{1, \dots, m\} \mid g_i(x) < 0\}$.

Definition 2.4 (Linear Independence Constraint Qualification)**.** $x \in \mathcal{F}$ satisfies the *Linear Independence Constraint Qualification (LICQ)* if the vectors

$$\nabla h_i(x), \ i = 1, \dots, p \text{ and } \nabla g_j(x), \ j \in \mathcal{A}(x) \tag{2.6}$$

are linearly independent.

We can finally state a very practical necessary optimality condition for NLP, based on the KKT-system. It is the basis for many optimization algorithms.

Theorem 2.5 (First-order Necessary Condition [26])**.** *If $x^\star \in \mathcal{F}$ is a local minimizer of NLP and satisfies LICQ, then there exists unique (λ^*, μ^*), such that $(x^\star, \lambda^*, \mu^*)$ is a KKT-point.*

For the validity of the theorem it is important that (2.6) holds. There are several other constraint qualifications that can be used instead of (2.6) like the *Abadie Constraint Qualification (ACQ)* or the *Mangasarian-Fromovitz Constraint Qualification (MFCQ)* [91]. Nevertheless LICQ implies ACQ and MFCQ. Although

ACQ and MFCQ are also sufficient they just ensure the existence of a KKT-point, while LICQ even ensures the existence of a unique KKT-point. We also mention the *Slater Constraint Qualification*, which is probably the most well-known constraint qualification in literature. It is also sufficient for the existence of a KKT-point and implies ACQ. To formulate second-order optimality conditions for NLP, we introduce the *critical cone* and require f, g and h to be twice continuously differentiable.

Definition 2.5 (Critical Cone)**.** For any $x \in \mathcal{F}, \lambda \in \mathbb{R}^m, \lambda \geq 0$ the *critical cone* is defined as

$$K(\mathcal{F}; x, \lambda) := \left\{ s \in \mathbb{R}^n \,\middle|\, \begin{array}{lll} \nabla h(x)^\top s & = 0 & \\ g_i(x)^\top s & = 0, & i \in \mathcal{A}(x), \lambda_i > 0 \\ g_i(x)^\top s & \leq 0, & i \in \mathcal{A}(x), \lambda_i = 0 \end{array} \right\}.$$

We can finally state the following second-order sufficient condition.

Theorem 2.6 (Second-Order Sufficient Condition [26])**.** *Let $(x^\star, \lambda^*, \mu^*)$ be a KKT-point. If for all $s \in K(\mathcal{F}; x^\star, \lambda^*)$ we have*

$$s^\top \nabla_x^2 L(x^\star, \lambda^*, \mu^*) s > 0,$$

then $x^\star$ is a strict local mimimum of NLP. Here, $\nabla_x^2 L$ denotes the Hessian of L with respect to x.

As we will see in Section 2.3, the KKT-conditions can be simplified in the convex case and therefore play an important role in convex optimization.

2.3 Lagrangian Duality and Convexity

In this section we deal with the concept of duality in nonlinear optimization. We will see that the duality theory for linear programming can be classified within this concept and that some results are valid analogously to the linear case.

2.3.1 Weak and Strong Duality

Recall a general nonlinear optimization problem NLP given in the form

$$\min\{f(x) \mid g(x) \leq 0,\ h(x) = 0\},$$

where $f : \mathbb{R}^n \to \mathbb{R},\ g : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^p$.

Studying the dual problem of NLP, also denoted as the *Lagrangian Dual*, is motivated by the following result.

Definition 2.6 (Saddle Point)**.** A vector $(x^\star, \lambda^*, \mu^*) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p$ is called *saddle point* of its Lagrangian L, if the following inequalities hold

$$L(x^\star, \lambda, \mu) \leq L(x^\star, \lambda^*, \mu^*) \leq L(x, \lambda^*, \mu^*), \tag{2.7}$$

for all $(x, \lambda, \mu) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p,\ \lambda \geq 0$.

Theorem 2.7 (Saddle Point Theorem [170])**.** *If $(x^\star, \lambda^*, \mu^*)$ is a saddle point of L, then $x^\star$ is a minimizer of NLP.*

This means the saddle point property (2.7) implies optimality. The idea of finding a saddle point of L is to first find a minimizer $x(\bar{\lambda}, \bar{\mu})$ of $L(\cdot, \bar{\lambda}, \bar{\mu})$ for fixed $(\bar{\lambda}, \bar{\mu}),\ \bar{\lambda} \geq 0$, and then to find a maximizer of $L(x(\bar{\lambda}, \bar{\mu}), \lambda, \mu)$. We motivate this idea by the simple Example 2.2.

Example 2.2. Consider the problem

$$\min\{f(x) := x^2 \mid g(x) := 1 - x \leq 0,\ x \in \mathbb{R}\}.$$

L is given by $L(x, \lambda) = x^2 + \lambda(1 - x)$. For fixed $\lambda \geq 0$, we have $x(\lambda) = \frac{\lambda}{2}$. Maximizing $L(x(\lambda), \lambda) = \lambda - \frac{\lambda^2}{4}$ yields $\lambda^* = 2$. Thus, $(x^\star, \lambda^*) = (1, 2)$ is a saddle point of L and hence a minimizer by Theorem 2.7.

To follow this idea, it is convenient to consider the inner minimization problem and the outer maximization problem independently.

Definition 2.7 (Dual Function)**.** The function $q : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R} \cup \{\infty\}$,

$$q(\lambda, \mu) := \inf_x L(x, \lambda, \mu)$$

is called the *dual function* of NLP. In general the dual function is not differentiable. The problem

$$\sup\{q(\lambda, \mu) \mid \lambda \geq 0,\ \mu \in \mathbb{R}^p\} \tag{NLD}$$

is called the *dual problem* of NLP. The corresponding *primal problem* of NLP is naturally defined as $\inf_x \sup_{\lambda \geq 0,\, \mu} L(x, \lambda, \mu)$. In fact, it can be shown that NLP is equivalent to its primal problem in the sense that every minimizer of the problem is also a minimizer of NLP and vice versa.

As we can see, the dual problem has simpler constraints, while its objective function is more complicated, so dualizing a problem moves the difficulty from the constraints to the objective function. Nevertheless, we will see that considering the dual problem can be useful. The following results explain the connection between the optimal objective function values of the primal and dual problem. It is natural to analyze if and under which conditions we achieve equality. In this case it might be easier to solve the dual problem instead. The Weak Duality Theorem states that for every feasible pair of points for the primal and dual problem the dual objective function value is a lower bound on the primal objective function value.

Lemma 2.8 (Weak Duality Theorem [91]). *Let $x \in \mathbb{R}^n$ be feasible to NLP and $(\lambda, \mu) \in \mathbb{R}^m \times \mathbb{R}^p$ be feasible to NLD, then*

$$\sup(NLD) \leq \inf(NLP).$$

The dual function q and its feasible domain

$$\operatorname{dom}(q) := \{(\lambda, \mu) \in \mathbb{R}^m \times \mathbb{R}^p \mid \lambda \geq 0,\, q(\lambda, \mu) > -\infty\}$$

have the properties that $\operatorname{dom}(q)$ is convex and $q : \operatorname{dom}(q) \to \mathbb{R}$ is concave. This means the dual problem can be transformed into a convex optimization problem by writing it as a minimization problem. Note that any dual problem is always convex, independently of the primal problem.

A *duality gap* between the primal and the dual solutions p and d can occur. This means that the lower bounds obtained by the dual problem might be weak in the sense that the gap is huge. The following theorem shows that under certain assumptions on the primal problem, we can obtain strong duality, i.e., $\sup(NLD) = \inf(NLP)$ and the duality gap is thus zero.

Theorem 2.9 (Strong Duality Theorem [91]). *If in NLP f, g are convex, h is affine-linear, furthermore, $\inf(NLP)$ is finite and there exists $\hat{x} \in \mathbb{R}^n$, such that $g(\hat{x}) < 0$ and $h(\hat{x}) = 0$, then the dual problem has an optimal solution and*

$$\sup(NLD) = \inf(NLP).$$

For example, for any convex quadratic optimization problem that has a strictly feasible point strong duality holds. We will make use of this useful result in particular in Chapter 5.

2.3.2 Optimality Conditions for Convex Programming

In this section we consider convex optimization problems, i.e., NLP, where the objective function as well as the feasible region are both convex:

$$\min\{f(x) \mid x \in C\}, \tag{CP}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $C \subseteq \mathbb{R}^n$. Convex optimization problems have some useful properties and are in general easier to solve in practice than non-convex problems. We start with some basic definitions and give important results on convex optimization problems, including optimality conditions.

Definition 2.8 (Convex Set). A set $C \subseteq \mathbb{R}^n$ is called *convex*, if for all $x, y \in C$ and for all $\lambda \in [0, 1]$:

$$(1 - \lambda)x + \lambda y \in C.$$

Definition 2.9 (Convex, Strictly Convex and Uniformly Convex Function)**.** Let $C \subseteq \mathbb{R}^n$ be convex. A function $f : C \to \mathbb{R}$ is called

(i) *convex*, if for all $x, y \in C$ and for all $\lambda \in [0, 1]$:

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

(ii) *strictly convex*, if in (i) strict inequality holds for all $x \neq y$.

(iii) *uniformly convex*, if there exists some $\mu > 0$, such that for all $x, y \in C$ and for all $\lambda \in [0, 1]$:

$$f(\lambda x + (1 - \lambda)y) + \mu\lambda(1 - \lambda)||x - y||^2 \leq \lambda f(x) + (1 - \lambda)f(y).$$

Using the following theorem, we can characterize convex, strongly convex and uniformly convex functions by first-order information.

Theorem 2.10 ([26])**.** *Let $C \subseteq \mathbb{R}^n$ be convex, open and $f : C \to \mathbb{R}^n$ be continuously differentiable. Then,*

(i) f is convex if and only if for all $x, y \in C$:

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

(ii) f is strictly convex, if and only if for all $x, y \in C$, $x \neq y$:

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

(iii) f is uniformly convex, if and only if there exists some $\mu > 0$, such that

$$f(y) - f(x) \geq \nabla f(x)^\top (y - x) + \mu||y - x||^2$$

for all $x, y \in C$.

To characterize convexity of twice continuously differentiable functions, the following observation is useful.

Theorem 2.11 ([91])**.** *Let $C \subseteq \mathbb{R}^n$ be convex, open and $f : C \to \mathbb{R}$ be twice continuously differentiable. Then*

(i) f is convex, if and only if $\nabla^2 f(x)$ is positive semidefinite for all $x \in C$.

(ii) If $\nabla^2 f(x)$ is positive definite for all $x \in C$, then f is strictly convex.

(iii) *f is uniformly convex, if and only if there exists some $\mu > 0$, such that*

$$s^\top \nabla^2 f(x) s \geq \mu ||s||^2$$

for all $x \in C$ and for all $s \in \mathbb{R}^n$.

Some useful features of convex optimization problems are summarized in the following theorem.

Theorem 2.12 ([91])**.** *Let $C \subseteq \mathbb{R}^n$ be convex and $f : C \to \mathbb{R}$ be convex. Consider the problem of minimizing f over C. Then we have that*

(i) *each local minimizer is also a global minimizer.*

(ii) *the set of optimal solutions is convex.*

(iii) *if f is strictly convex, at most one global minimizer exists.*

(iv) *if f is uniformly convex and C is nonempty and closed, there exists a unique global minimizer.*

For any general convex set $C = \{x \in \mathbb{R}^n \mid h(x) = 0,\ g(x) \leq 0\} \subseteq \mathbb{R}^n$ given by equations and inequalities, we have the following sufficient optimality condition which is a consequence of Theorem 2.7, since KKT-points coincide with saddle points if the optimization problem is convex.

Theorem 2.13 (Sufficient Optimality Condition [91])**.** *If $(x^\star, \lambda^*, \mu^*)$ is a KKT-point, $x^\star$ is a global minimizer of CP.*

In case f is convex and $C = \mathbb{R}^n$, i.e., we have an unconstrained convex optimization problem, any stationary point is a global minimizer, since the KKT-system reduces to (2.3).

We already know that the KKT-conditions are sufficient for optimality. If the following constraint qualification holds we can also guarantee that the KKT-conditions are necessary for optimality.

Definition 2.10 (Slater Condition)**.** A feasible point $\hat{x} \in \mathcal{F}$ is called *Slater Point*, if $g(\hat{x}) < 0$.

Note that this condition was already used in the Strong Duality Theorem. The KKT-conditions are sufficient for optimality for CP. They are also necessary only if a constraint qualification like Slater's condition holds.

Theorem 2.14 (Necessary Optimality Condition [91])**.** *If $x^\star$ is a global minimizer of CP and in addition a Slater Point exists, then there exists a KKT-point $(x^\star, \lambda^*, \mu^*)$.*

Assume that we have a convex optimization problem of the form CP, where the feasible region $\mathcal{F}$ is given by general inequalities $g(x) \leq 0$ and equations $h(x) = 0$, i.e., $\mathcal{F} = \{x \in \mathbb{R}^n \mid g(x) \leq 0,\, h(x) = 0\}$, where $g : \mathbb{R}^n \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^p$. In some special cases, e.g., when h and g are linear, the constraint qualification simplifies to require the existence of any feasible point $x \in \mathcal{F}$. In this case there exists a global minimizer of CP, if and only if there exists a KKT-point. For example this is true for any convex quadratic programming problem. We will study this type of problems in the next section.

2.4 Quadratic Programming

Quadratic Programming (QP) plays an important role in mathematical programming. Optimization problems involving a quadratic objective function and linear constraints are often used as subproblems to solve more general nonlinear optimization problems. A representative of this class of algorithms is the iterative *Sequential Quadratic Programming* algorithm [32, 190], one of the most effective methods for nonlinearly constrained optimization problems. Concerning the complexity it was shown that convex quadratic programming can be solved in polynomial time by using either the ellipsoid method [128] or the interior point method [119, 193]. In contrast, non-convex quadratic programming was shown to be NP-hard by Sahni [173]. Pardalos and Vavasis [166] showed that quadratic programming in which the quadratic matrix has a single negative eigenvalue is already NP-hard. In this section we are mainly interested in conditions which ensure the existence of an optimal solution.

2.4.1 Unconstrained Quadratic Programming

We start with unconstrained quadratic programming problems, i.e., problems of the form

$$\min\{f(x) := x^\top Qx + c^\top x + d \mid x \in \mathbb{R}^n\}, \tag{UQP}$$

where $Q \in \mathbb{R}^{n\times n}$, $c \in \mathbb{R}^n$ and $d \in \mathbb{R}$. Note that without loss of generality Q can be assumed to be symmetric, since $x^\top Qx = x^\top \bar{Q}x$, where $\bar{Q} = \frac{1}{2}(Q + Q^\top)$ is symmetric.

According to Theorem 2.11, we have that

- Q is positive definite $\Leftrightarrow$ $f(x)$ is strictly convex, and
- Q is positive semidefinite $\Leftrightarrow$ $f(x)$ is convex.

The implication $f(x)$ is strictly convex $\Rightarrow$ Q is positive definite does not follow immediately but can be shown as follows: Let $t \in \mathbb{R}^n, t \neq 0$. For any $x \in \mathbb{R}^n$ we

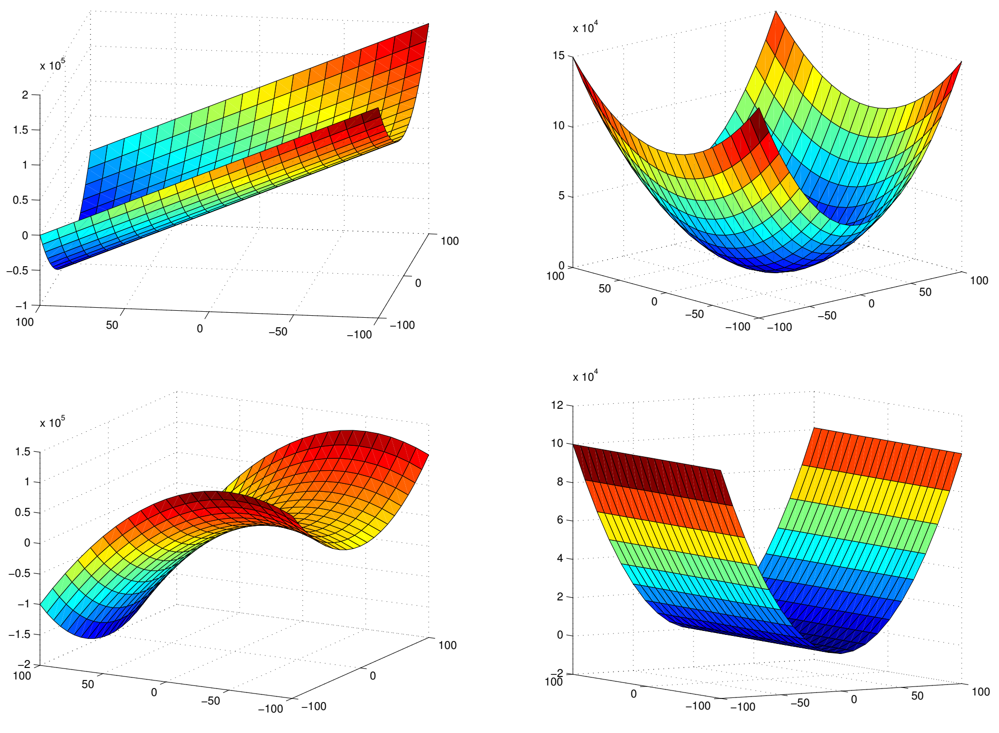

Figure 2.1: The top left picture shows an unbounded convex function without minima, the top right one shows a strictly convex function with a unique minimum, the bottom left one is a non-convex function and the bottom right one is a convex function with infinitely many minima.

define $y := x - t$. We have that $(x - y)^\top(\nabla f(x) - \nabla f(y)) > 0$ since f is strictly convex. By inserting the gradients we see that this is equivalent to $t^\top Qt > 0$.

Furthermore, for a quadratic function we have the equivalence between strictly and uniformly convex.

Lemma 2.15. *A quadratic function f is strictly convex if and only if f is uniformly convex.*

Proof. Let λ_{min} be the smallest eigenvalue of Q. We have $\lambda_{min} > 0$, such that $s^\top \nabla^2 f(x)s = 2s^\top Qs \geq 2\lambda_{min}||s||^2$ for all $s \in \mathbb{R}^n$. $\square$

In particular Theorem 2.12 implies that any strictly convex quadratic program has a unique minimizer. The following theorem shows that any unconstrained non-convex quadratic optimization problem is unbounded, while any strictly convex quadratic optimization problem has a unique minimizer that can be computed by solving a system of linear equations. A convex but not strictly convex quadratic optimization problem might have either infinitely many or no minima. In Figure 2.1, we illustrate the different cases for convex functions.

Theorem 2.16 ([70])**.** *For UQP, it holds*

(i) *$x^\star$ is a local minimizer of UQP, if and only if Q is positive semidefinite and $\nabla f(x^\star) = 2Qx^\star + c = 0$.*

(ii) *UQP has a unique solution, if and only if Q is positive definite.*

(iii) *If Q is not positive semidefinite, then UQP is unbounded from below.*

2.4.2 Quadratic Programming with Linear Constraints

We consider equality constrained quadratic programming problems, i.e., problems of the form

$$\begin{aligned} \min\ & x^\top Qx + c^\top x + d \\ \text{s.t. } & Ax = b \\ & x \in \mathbb{R}^n, \end{aligned} \tag{EQP}$$

where $Q \in \mathbb{R}^{n\times n},\ Q = Q^\top,\ c \in \mathbb{R}^n,\ d \in \mathbb{R},\ A \in \mathbb{R}^{p\times n}$ and $b \in \mathbb{R}^p$. Equality constrained quadratic programming problems are easy to solve if Q is assumed to be positive semidefinite since this implies f to be convex, and hence the KKT-conditions are necessary and sufficient for optimality. Thus, if there exists a solution $(x^\star, \mu^*) \in \mathbb{R}^n \times \mathbb{R}^p$ of the KKT-system, $x^\star$ is also a minimizer of EQP, and vice versa. In particular, in this case the KKT-system is a system of linear equations:

$$\begin{pmatrix} 2Q & A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \mu \end{pmatrix} = \begin{pmatrix} -c \\ b \end{pmatrix}. \tag{2.8}$$

From a practical point of view we have the following theorem.

Theorem 2.17 ([91])**.** *Let $x^k \in \mathbb{R}^n$ be feasible for EQP. A pair $(x^\star, \mu^*) \in \mathbb{R}^n \times \mathbb{R}^p$ is a KKT-point of EQP if and only if $x^\star = x^k + \Delta x^\star$ and $(\Delta x^\star, \mu^*)$ is a solution of the following system of linear equations*

$$\begin{pmatrix} 2Q & A^\top \\ A & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \mu \end{pmatrix} = \begin{pmatrix} -\nabla f(x^k) \\ 0 \end{pmatrix}.$$

This result will be used in standard active set methods, see Section 2.5.3. Concerning the existence and uniqueness of minima for EQP, we can state the following result.

Theorem 2.18. *If Q is positive definite and $rk(A) = p \leq n$, there exists a unique solution $(x^\star, \mu^*)$ of (2.8) and $x^\star$ is the unique minimizer of EQP.*

Proof. Let f be the quadratic objective function of EQP. For any arbitrary vector $z \in \mathbb{R}^n$ the level sets $N_f(z) = \{x \in \mathbb{R}^n \mid f(x) \leq f(z)\}$ of f are compact. Since $rk(A) = p$, we have that the feasible region is nonempty: let $A = (A_0, A_1)$, where $A_0 \in \mathbb{R}^{p \times p}$, $A_1 \in \mathbb{R}^{p \times (n-p)}$ and $rk(A_0) = p$ (if necessary after permutation of the columns of A). Thus,

$$z_0 = (A_0^{-1} b, 0)^\top$$

is a feasible point. According to Theorem 2.1 there exists a minimizer of EQP. Since f is strictly convex, it is also unique using Theorem 2.12. □

Again, like in the unconstrained case, it is sufficient to assume the objective function to be strictly convex and additionally the constrained matrix to be full-ranked to have a unique solution. Solving the KKT-system (2.8) can be done in different ways. Besides solving it directly, it can be done by using *Range-Space Methods* or *Null-Space Methods*. We refer to [162] for details.

If we allow inequality constraints, the problem becomes

$$\begin{aligned} \min\ & f(x) := x^\top Q x + c^\top x + d \\ \text{s.t. } & Ax \leq b \\ & x \in \mathbb{R}^n, \end{aligned} \tag{QP}$$

where $Q \in \mathbb{R}^{n \times n}$, $Q = Q^\top$, $c \in \mathbb{R}^n$, $d \in \mathbb{R}$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

By using the necessary optimality conditions (2.5) for NLP and taking into account that no constraint qualification is needed since all constraints are affine-linear [91], we get the following first-order necessary condition for QP.

Theorem 2.19 ([162])**.** *If $x^\star$ is local minimizer of QP, then there exists $(x^\star, \lambda^*)$ that satisfies the KKT-system:*

$$\begin{aligned}
& 2Qx^\star + c + \sum_{i \in \mathcal{A}(x^\star)} \lambda_i^* a_i = 0 && \text{(Stationarity)} \\
& g_i(x) = a_i^\top x^\star \leq b_i,\ i \in \mathcal{N}(x^\star) && \text{(Primal Feasibility)} \\
& g_i(x) = a_i^\top x^\star = b_i,\ i \in \mathcal{A}(x^\star) && \text{(Primal Feasibility)} \\
& \lambda_i^* \geq 0,\ i \in \mathcal{A}(x^\star) && \text{(Dual Feasibility, Complementarity)} \\
& \lambda_i^* = 0,\ i \in \mathcal{N}(x^\star) && \text{(Dual Feasibility, Complementarity)}
\end{aligned}$$

Again, Theorem 2.12 and Lemma 2.15 imply the existence of a unique solution if Q is positive definite.

Corollary 1. *If Q is positive definite and the feasible region is nonempty, then there exists a unique solution for QP.*

In this case, since the KKT-conditions are necessary and sufficient for optimality, we can conclude that the unique minimizer can be computed by the unique solution of the KKT-system.

For general Q, we can at least give sufficient conditions under which there exists a minimizer. The result is the well-known theorem of Frank and Wolfe.

Theorem 2.20 (Frank-Wolfe Theorem [70]). *Let $X \subseteq \mathbb{R}^n$ be a nonempty polyhedron, then the optimization problem $\min\{f(x) = x^\top Qx + c^\top x + d \mid x \in X\}$ attains its global minimum on X, if and only if f is bounded from below.*

This theorem is not valid for arbitrary nonlinear functions. For example $f(x) = e^x$ does not have any minimizer although bounded from below. Using the Frank-Wolfe Theorem, it is possible to derive a sufficient condition for the existence of minima for QP, if Q is positive semidefinite but not definite.

For any convex set $C \subseteq \mathbb{R}^n$ its *recession cone* is defined as the following cone $\text{rec}(C) := \{y \in \mathbb{R}^n \mid x + \lambda y \in C \text{ for all } x \in C,\ \lambda \geq 0\}$.

Theorem 2.21 ([58]). *Let $C \subseteq \mathbb{R}^n$ be the nonempty feasible polyhedral region of QP. Then it holds:*

(i) If QP has a solution, then $x^\top Qx \geq 0$ for all $x \in rec(C)$ and

$$c^\top x \geq 0 \text{ for all } x \in rec(C) \cap ker(Q). \tag{2.9}$$

(ii) If (2.9) holds and f is convex, QP has a solution.

Without loss of generality assume the polyhedral set contains only inequalities. Otherwise the equations can be written as inequalities. It is interesting that the condition (2.9) can be checked in polynomial time by solving the following linear program:

$$\min\{c^\top x \mid Ax \leq 0,\ Qx = 0,\ x \in \mathbb{R}^n\}, \tag{2.10}$$

since the recession cone of a polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ is the set $\{x \in \mathbb{R}^n \mid Ax \leq 0\}$. Thus, if there exists an optimal solution $x^\star$ of (2.10), such that $c^\top x^\star < 0$, QP is unbounded from below and there is no solution. On the other hand, if $c^\top x^\star \geq 0$, the objective function is bounded from below and hence there exists at least one solution to the problem if f is convex. This shows that if Q is positive semidefinite (but not positive definite), we have a necessary and sufficient condition for the existence of an optimal solution for QP, which can be checked in polynomial time.

In general, there are two common approaches for solving QP, namely interior point methods and active set methods. In interior point methods, a sequence of parameterized barrier functions is (approximately) minimized using Newton's method. The main computational effort consists in solving the Newton system to get the search direction. In active set methods, at each iteration, a working set that estimates the set of active constraints at the solution is iteratively updated. They will be briefly described in the next Section.

2.5 Methods for Nonlinear Programming

This chapter is based on the books of Nocedal and Wright [162], Bazaraa et al. [17] and Geiger and Kanzow [91] which give a detailed insight into theory and practice of nonlinear optimization. For an overview of NLP software we refer to the work of Leyffer and Majahan [139]. We start with unconstrained optimization problems. They are of great interest since they are often used to solve subproblems in approaches for constrained optimization problems. After giving a brief description of the common approaches, we take a closer look at line search methods. After that, we concentrate on two quadratic programming algorithms, namely the conjugate gradient method for unconstrained convex quadratic programming problems and the active set method for linearly constrained convex quadratic programming problems. All three types of algorithms, line search method, conjugate gradient method and active set method will play an important role in the design of our fast active set algorithm for convex mixed-integer programming in Chapter 5. Finally we take a look at the Frank-Wolfe Algorithm for solving constrained convex optimization problems. We use an algorithm tailored to our branch-and-bound algorithm for a class of mixed-integer portfolio optimization problems in Chapter 7.

2.5.1 Unconstrained Nonlinear Optimization

All algorithms for unconstrained nonlinear optimization problems follow the idea of generating a sequence of iterates $\{x^k\}_{k=0}^{\infty}$ such that for some iterate we can either verify optimality (up to a certain accuracy) or no progress can be made. The design of these algorithms differs in the way of deciding how to move from one iterate x^k to the next x^{k+1}. There are mainly two categories of approaches of choosing the next iterate: *line search methods* and *trust region methods*.

Line search methods are moving from the current iterate x^k to a new one x^{k+1} with a better objective function value along some descent direction s^k. To decide how far the step size along the descent direction should be they solve an one-dimensional optimization problem: $\min_{\alpha_k>0} f(x^k + \alpha_k x^k)$. This is often done approximately, since solving this subproblem can already be expensive. If we solve the problem to optimality we call it an *exact line search*, otherwise an *inexact line search*.

Trust region methods are different, as in each iteration k they first approximate $f(x^k + s)$ by some (simpler) auxiliary function $\bar{f}_k(s)$ and then minimize it locally in a trust region Δ_k in which both are similar:

$$\min\{\bar{f}_k(s) \mid ||s|| \leq \Delta_k\}. \tag{2.11}$$

Typically trust regions are chosen as either balls $\Delta_k = \{s \in \mathbb{R}^n \mid ||s|| \leq \Delta\}$, where $\Delta > 0$, or boxes/ellipsoids. The auxiliary function is usually chosen as a quadratic function of the form $\bar{f}_k(s) = f(x^k) + \nabla f(x^k)^\top s + \frac{1}{2}s^\top B_k s$, where B_k is either the Hessian or an approximation of it. Depending on the ratio of actual and expected reduction of the objective function $\rho_k = \frac{f(x^k)-f(x^k+s^k)}{\bar{f}_k(0)-\bar{f}_k(s^k)}$, either the solution s^k is accepted and the new iterate is set to $x^{k+1} = x^k + s^k$ or the trust region is modified and the problem is solved again. It is common to reject the step and decrease the trust region if $\rho_k < 0$ since it implies the objective function value to become worse in the current iteration. On the other hand if $\rho_k \approx 1$, we can increase the trust region, and if $\rho_k \gg 1$, the current trust region is kept. The fundamental question is how to solve the trust region problems (2.11). Well-known algorithms that solve the problem approximately are for example the *dogleg method* for positive definite B_k, and the *two-dimensional subspace minimization* for indefinite B_k, a method by *Steinhaug*, based on the CG Method, if B_k is chosen as the exact Hessian of f. In this section we are not going into further details about trust region methods since they are not needed in our approaches. A survey on trust region methods is given by Dennis and Schnabel [67].

Line Search Methods

We start with the definition of a *descent direction*, i.e., directions in which we can improve our objective function. In fact, moving along any descent direction of f, we can guarantee an improvement of the objective function value.

Definition 2.11 (Descent Direction)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable in $x \in \mathbb{R}^n$. A vector $s \in \mathbb{R}^n$ is called a descent direction of f in x if we have

$$\nabla f(x)^\top s < 0.$$

Lemma 2.22 ([9])**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable in $x \in \mathbb{R}^n$ and $s \in \mathbb{R}^n$ be a descent direction in x. Then there exists some $\alpha > 0$, such that $f(x + \alpha s) < f(x)$.*

This leads to the following iterative procedure, sketched in Algorithm 1. The algorithm starts with an initial guess x^0 and determines a feasible descent direction s^0 as well as a feasible step size α_0 to compute the new iterate $x^1 := x^0 + \alpha_0 s^0$ with a reduced objective function value. This is repeated until a stopping criterion is satisfied. In practice the stopping criterion is $||\nabla f(x^k)|| < \varepsilon$, where $0 < \varepsilon \ll 1$.

Algorithm 1: Line Search

```
input  : differentiable function $f : \mathbb{R}^n \to \mathbb{R}$
output: stationary point $x^\star$ of $f$

Choose starting point $x^0 \in \mathbb{R}$. Set $k = 0$.
while $\nabla f(x^k) \neq 0$ do
    Compute descent direction $s^k$.
    Compute step size $\alpha_k > 0$, such that
        $f(x^k + \alpha_k s^k) < f(x^k).$
    Set $x^{k+1} = x^k + \alpha_k s^k$, $k = k + 1$.
```

The obvious questions are how to determine a suitable descent direction s and an appropriate step size α. Concerning the descent direction it is common to choose

$$s = -B^{-1}\nabla f$$

for some symmetric and regular matrix B. Two specific choices for B are for example $B = I$ or $B = \nabla^2 f(x)$, if the Hessian is positive definite. The resulting descent directions are called *Anti-Gradient* and *Newton-Direction*, respectively.

Assuming we are interested in an inexact line search, some common conditions on the step sizes then are:

- Wolfe conditions:

$$f(x^k + \alpha_k s^k) \leq f(x^k) + c_1 \alpha_k \nabla f(x^k)^\top s^k \tag{2.12}$$
$$\nabla f(x^k + \alpha_k s^k)^\top s^k \geq c_2 \nabla f(x^k)^\top s^k, \tag{2.13}$$

  where $0 < c_1 < c_2 < 1$.

  Condition (2.12) is also known as the *Armijo condition* and guarantees a sufficient decrease of the objective function. The second condition (2.13) is the *curvature condition* that ensures the line search to make reasonable progress since a sufficiently small step size would always satisfy the Armijo condition. Furthermore, it may be the case that some step size that satisfies the Wolfe conditions is not close to a minimizer of $\phi(\alpha) = f(x^k + \alpha s^k)$. To ensure this, we can require the derivative $\phi'(\alpha)$ to be non-negative such that points far away from the minimizer are excluded by the following additional requirement (2.14).

- Strong Wolfe conditions:

$$f(x^k + \alpha_k s^k) \leq f(x^k) + c_1 \alpha_k \nabla f(x^k)^\top s^k$$
$$|\nabla f(x^k + \alpha_k s^k)^\top s^k| \leq c_2 |\nabla f(x^k)^\top s^k|, \tag{2.14}$$

where $0 < c_1 < c_2 < 1$. A third pair of conditions are the

- Goldstein conditions:

$$f(x^k) + (1-c)\alpha_k \nabla f(x^k)^\top s^k \leq f(x^k + \alpha_k s^k) \tag{2.15}$$
$$\leq f(x^k) + c\alpha_k \nabla f(x^k)^\top s^k, \tag{2.16}$$

  where $0 < c < \frac{1}{2}$.

  Again, the first inequality prevents to have a step size too small and the second inequality ensures the sufficient decrease.

Often the sufficient decrease condition (2.12) is combined with a backtracking such that we do not need to consider (2.13). It starts with an initial step size and reduces it step by step until the sufficient decrease condition holds, see Algorithm 2.

Algorithm 2: Backtracking Line Search

```
input  : iterate $x^k$, search direction $s^k$, step size $\alpha_k$
output: feasible step size $\alpha_k$
Choose $\alpha^k > 0$, $\rho, c \in (0,1)$.
while $f(x^k + \alpha_k s^k) > f(x^k) + c\alpha_k \nabla f(x^k)^\top s^k$ do
    Set $\alpha^k = \rho\alpha^k$.
```

If the descent directions and step sizes are chosen appropriately the line search algorithm converges to a stationary point. We therefore define *feasible search directions* and *feasible step sizes.*

Definition 2.12 (Feasible Search Direction, Feasible Step Size)**.** Let $\{s^k\}_k$, $\{x^k\}_k$ and $\{\alpha_k\}_k$ be the the sequences of search directions, iterates and step sizes generated by Algorithm 1. The sequence $\{s^k\}_k$ is called *feasible*, if for all $\{x^k\}_k$ we have the implication:

$$\lim_{k\to\infty} \frac{\nabla f(x^k)^\top s^k}{||s^k||} = 0 \Rightarrow \lim_{k\to\infty} \nabla f(x^k) = 0. \tag{2.17}$$

The sequence $\{\alpha_k\}_k$ is called *feasible*, if for all $\{x^k\}_k$ and $\{s^k\}_k$ we have the implication:

$$f(x^k + \alpha_k s^k) < f(x^k) \text{ for all } k \in \mathbb{N} \text{ and}$$
$$\lim_{k\to\infty} (f(x^k) - f(x^k + \alpha_k s^k)) = 0 \Rightarrow \lim_{k\to\infty} \frac{\nabla f(x^k)^\top s^k}{||s^k||} = 0.$$

The condition (2.17) ensures that the descent direction does not tend to become orthogonal to the gradient. Both, the Anti-Gradient as well as the Newton-Direction are feasible search directions. On the other hand, it can be shown that under standard assumptions the Wolfe conditions as well as the Goldstein conditions are choosing step sizes that are both feasible.

Under additional requirements, we can assure the convergence of the line search method to a minimizer.

Theorem 2.23 ([9]). *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable, x^0 be the starting point, the level set $N_f(x^0)$ be convex and compact and f strictly convex on $N_f(x^0)$. Furthermore, let the sequence of descent directions $\{s^k\}_k$ and step sizes $\{\alpha_k\}_k$ be feasible. Then there exists a unique global minimizer $x^\star$ of f and Algorithm 1 produces a sequence $\{x^k\}_k$ that converges to $x^\star$.*

The assumptions made are quite restrictive since we require the objective function in a neighborhood of $x^\star$ to be strictly convex and additionally that the starting point x^0 has to be in that neighbourhood.

One of the most important representatives of line search algorithms is the *Newton Method* that uses in its most general form the Newton Direction as a descent direction and step size 1. Assuming the starting point x^0 is sufficiently close to a minimizer $x^\star$, it can be shown that if f is twice continuously differentiable and the Hessian of f is Lipschitz continuous in a neighbourhood of $x^\star$ at which $\nabla f(x^\star) = 0$ and $\nabla^2 f(x^\star)$ is positive definite, then $\{x^k\}_k$ converges to $x^\star$ and $\{||\nabla f(x^k)||\}_k$ converges to 0 quadratically. Several modifications can be made to get a practical global convergent version of the Newton Method. For example in the *Modified Newton Method*, whenever necessary, the Hessian is modified to be positive definite throughout the whole algorithm to guarantee descent directions. To save computational expenses, the computation of the Newton Direction can be done approximately, yielding an *Inexact Newton Method*. In particular, *Quasi-Newton* type methods aim at reducing the computational effort by approximating the Hessian with some matrix that is easier to invert.

2.5.2 Unconstrained Quadratic Optimization

Conjugate Gradient Method

The conjugate gradient method (CG) was introduced by Hestenes and Stiefel [110] in the 1950s for solving systems of linear equations $Qx = b$ for any positive definite coefficient matrix $Q \in \mathbb{R}^{n\times n}$ and right hand side vector $b \in \mathbb{R}$. It is obvious that solving $Qx = b$ is equivalent to minimizing the unconstrained quadratic programming problem

$$\min\{f(x) = \frac{1}{2}x^\top Qx - b^\top x \mid x \in \mathbb{R}^n\}. \tag{2.18}$$

Looking at this, we can consider unconstrained quadratic programming as an alternative way to solve systems of linear equations, if Q is positive definite. Theorems 2.12 and 2.13 guarantee the existence of a unique minimizer of the quadratic programming problem.

CG Methods are specific line search algorithms, i.e., they iteratively create a sequence of points $x^{k+1} = x^k + \alpha_k s^k$ by choosing appropriate step sizes α_k and descent directions s^k in each iteration. The method starts with the anti-gradient as an initial descent direction but continues with modified Anti-Gradients such that a certain property, called *conjugacy* holds for the set of search directions.

Definition 2.13 (Q-Conjugacy)**.** A set of nonzero vectors $\{p^0, p^1, \dots, p^\ell\}$ is called *Q-conjugate* for a symmetric positive definite matrix $Q \in \mathbb{R}^{n\times n}$, if ${p^i}^\top Q p^j = 0$ for all $i \neq j$.

This property ensures that given an arbitrary starting point $x^0 \in \mathbb{R}^n$ and a set of n nonzero Q-conjugate vectors $p_i \in \mathbb{R}^n,\ i = 0, \dots, n-1$, the iterations

$$x^{k+1} = x^k + \alpha_k p^k, \tag{2.19}$$

where $\alpha_k = -\frac{\nabla f(x^k)^\top p^k}{{p^k}^\top Q p^k}$, give a minimizer of (2.18) after at most n iterations. Note that any two eigenvectors of Q corresponding to distinct eigenvalues are Q-conjugate. But computing all the eigenvectors would be too expensive. Instead of computing all the Q-conjugate vectors a priori, it is possible to compute them iteratively. This leads to the question of how to determine a set of Q-conjugate vectors. The answer is given by the following constructive Lemma.

Lemma 2.24 ([90])**.** *Let $p^0 \in \mathbb{R}^n,\ p^0 \neq 0$ be arbitrary and $p^k,\ k = 1, \dots, n-1$, be defined by*

$$p^k = -\nabla f(x^k) + \beta_{k-1} p^{k-1}, \text{ where } \beta_{k-1} = \frac{\nabla f(x^k)^\top Q p^{k-1}}{{p^{k-1}}^\top Q p^{k-1}}$$

and x^k defined as in (2.19)*. Then $p^0, \dots, p^{n-1}$ are Q-conjugate.*

Note that in the above construction of the set of Q-conjugate vectors, each vector x^k is constructed by using the current iterate x^k and the preceding conjugate vector p^{k-1}. In practice, the computation of α_k and β_k can be simplified by the following formulas

$$\alpha_k = \frac{||\nabla f(x^k)||^2}{{p^k}^\top Q p^k}, \quad \beta_k = \frac{||\nabla f(x^{k+1})||^2}{||\nabla f(x^k)||^2}.$$

In particular $\nabla f(x^k)^\top p^k = -||\nabla f(x^k)||^2 < 0$ for all $\nabla f(x^k) \neq 0$ implies that the CG Method in fact belongs to the class of line search methods. The CG Method in its most general form is given by Algorithm 3. Putting these results together, we can state the following convergence theorem.

Algorithm 3: Conjugate Gradient Method

```
input  : symmetric, positive definite matrix $Q \in \mathbb{R}^{n \times n}$, vector $b \in \mathbb{R}^n$
output: minimizer of $f(x) := \frac{1}{2}x^\top Qx - L^\top x$
Choose $x^0 \in \mathbb{R}^n$, $p^0 = -\nabla f(x^0)$, set $k = 0$.
while $\nabla f(x^k) \neq 0$ do
    Set $\alpha_k = \frac{||\nabla f(x^k)||^2}{{p^k}^\top Qp^k}$.
    Compute
        $x^{k+1} = x^k + \alpha_k p^k$
        $\nabla f(x^{k+1}) = \nabla f(x^k) + \alpha_k Qp^k$
        $\beta_k = \frac{||\nabla f(x^{k+1})||^2}{||\nabla f(x^k)||^2}$
        $p^{k+1} = -\nabla f(x^{k+1}) + \beta_k p^k$
    Set $k = k + 1$.
```

Theorem 2.25 ([90])**.** *If Q is symmetric and positive definite, Algorithm 3 terminates after at most n iterations with the unique minimizer of* (2.18).

The convergence rate of the CG Method is linear, since we have the following error estimation:

$$||x^k - x^\star|| \leq \left(\frac{\sqrt{\kappa(Q)} - 1}{\sqrt{\kappa(Q)} + 1} \right)^{2k} ||x_0 - x^\star||,$$

where $\kappa(Q) = \frac{\lambda_{max}(Q)}{\lambda_{min}(Q)}$ is the condition of the matrix Q. We can see that the convergence highly depends on the distribution of the eigenvalues of Q. Therefore, in practice CG Methods are often accelerated via *Preconditioning*: Instead of using the CG Method for the original problem we consider the transformation $y = Cx$ and the resulting problem $C^{-\top}QC^{-1}y = C^{-1}b$, where $C \in \mathbb{R}^{n \times n}$ is non-singular and $\kappa(C^{-\top}QC^{-1})$ is smaller than $\kappa(Q)$. Some common choice is the *Incomplete Cholesky Factorization* $C = \tilde{L}^\top$, where $Q \approx \tilde{L}\tilde{L}^\top$ such that $\tilde{L}$ is sparser than the the matrix in the exact Cholesky factorization. The idea of the CG Method is extendible to general unconstrained nonlinear optimization problems of the form $\min\{f(x) \mid x \in \mathbb{R}^n\}$ with an arbitrary continuously differentiable objective function $f : \mathbb{R}^n \to \mathbb{R}$. Popular versions are for example the *Fletcher-Reeves Method* and the *Polak-Ribiere Method.* For details, we refer to [162].

2.5.3 Constrained Quadratic Optimization

Essentially two classes of methods have been developed for solving linearly constrained quadratic programs of the form QP, namely *active set methods* (ASM) (see, e.g., [94, 168]) and *interior point methods* (IPM) (see, e.g., [95, 152]). For a detailed comparison, we refer to [162]. In this thesis, we will focus on inequality constrained convex problems, i.e., we assume Q to be positive semidefinite. Equality constrained quadratic programs of the form EQP do not require special algorithms to be solved, since solving its KKT-system, which is a system of linear equations, is sufficient (see Section 2.4). We therefore concentrate on quadratic programs with inequalities and describe a solution technique based on the reduction to equality constrained problems.

Active Set Methods

The main idea of active set methods is to solve a sequence of equality constrained quadratic programs by considering only the active constraints at the current iterate. For sake of simplicity, assume we have an optimization problem of the form QP without equations, i.e., $m = 0$. Recall that by Definition 2.3 the active and non-active sets at some point x^k are denoted by

$$\mathcal{A}(x^k) := \{i \mid a_i^\top x^k = b_i\} \quad \text{and} \quad \mathcal{N}(x^k) := \{1, \dots, m\} \setminus \mathcal{A}(x^k).$$

It is well known that active set methods are preferable to interior point methods in practice, if the number of constraints is small or medium.

The basic idea of general active set methods for convex quadratic programming problems relies on the following observation. If $x^\star$ is the minimizer of QP, we have

$$\begin{aligned} x^\star &= \text{ argmin } f(x) \text{ s.t. } Ax \leq b \\ &= \text{ argmin } f(x) \text{ s.t. } a_i^\top x = b_i \text{ for all } i \in \mathcal{A}(x^\star). \end{aligned}$$

Knowing the optimal active set of $x^\star$ would solve the problem immediately by finding the solution to (2.8). Since this is not the case, the approach starts by choosing a *working set* $\mathcal{W}_k \subseteq \{1, \dots, m\}$ and computing

$$x^{k+1} = \text{ argmin } f(x) \text{ s.t. } a_i^\top x = b_i \text{ for all } i \in \mathcal{W}_k.$$

If the new iterate x^{k+1} is not the minimizer of QP, adjust $\mathcal{W}_k$ to $\mathcal{W}_{k+1}$ in a clever way and solve the updated equality constrained quadratic programming problem. It is common to distinguish between primal and dual active set methods. While a primal active set method ensures primal feasibility of all iterates, a dual active set methods ensures dual feasibility.

Let $\mathcal{W}_k$ denote the current working set and A_k the matrix that consists of all rows of A belonging to $\mathcal{W}_k$. We assume that the row vectors $a_i,\ i = 1, \dots, m$, are linearly independent. Algorithm 4 gives a basic version of a primal active set method for quadratic optimization problems with linear inequalities.

Theorem 2.26 ([91])**.** *Consider Algorithm 4 for Problem QP.*

(i) If Q is positive definite and a_i $(i \in \mathcal{W}_k)$ are linear independent, (2.20) *has a unique solution.*

(ii) If a_i $(i \in \mathcal{W}_k)$ are linear independent at any iteration k and the algorithm does not terminate after solving (2.20)*, this also holds for a_i $(i \in \mathcal{W}_{k+1})$.*

(c) If Q is positive definite and $\Delta x^k \neq 0$, we have $\nabla f(x^k)^\top \Delta x^k < 0$, i.e., Δx^k is a descent direction.

In each iteration k, the algorithm considers the feasible iterate x^k and checks if it minimizes the quadratic objective function in the subspace defined by the current working set $\mathcal{W}_k$. If not, it computes a descent direction Δx^k by solving (2.20). This corresponds to solving the equality constrained quadratic program with constraints $a_i^\top x = b_i$ for all $i \in \mathcal{W}_k$ while all other constraints are relaxed. From Theorem 2.17 we know that $\Delta x^k \neq 0$, since otherwise the iterate would already be optimal in that subspace. On the other hand $(x^k + \Delta x^k, \lambda^{k+1}_{\mathcal{W}_k})$ is a KKT-point of that equality constrained problem. Since $A_k \Delta x^k = 0$, every constraint in the working set will still be satisfied with equality by the new iterate $x^k + \Delta x^k$. If (c) holds, i.e., $x^k + \Delta x^k$ is feasible for all constraints, we update our iterate and the working set is not changed. If there are constraints $i \notin \mathcal{W}_k$ that are violated by the new iterate, we choose the step size α_k as large as possible until the first constraint that is violated becomes active and any other constraints stays strictly feasible. Therefore we choose the step size as $\alpha_k := \min_{i \notin \mathcal{W}_k,\ a_i^\top \Delta x^k > 0} \frac{b_i - a_i^\top x^k}{a_i^\top \Delta x^k}$, update our iterate by $x^k + \alpha_k \Delta x^k$ and add this *blocking constraint* to the working set. This iterative process is repeated until a point is reached that is optimal for the current working set, i.e., when $\Delta x^k = 0$. Then, either all of the multipliers in the working set $\mathcal{W}_k$ are non-negative and we can stop since (x^{k+1}, λ^{k+1}) is already a KKT-point of our original problem, or if there is a negative multiplier in the working set $\mathcal{W}_k$, we remove this constraint. It can be shown that the solution of the resulting subproblem (2.20) gives a feasible direction for the constraint that has just been dropped.

Assuming that it is always possible to take a nonzero step along a nonzero descent direction guarantees that the algorithm does not cycle. It can also be shown that no working set is considered twice. Since the number of different working sets is finite, the algorithm stops after a finite number of iterations with a KKT-point of the problem. Nevertheless active set methods can have exponential running time

Algorithm 4: Active Set Method

```
input  : symmetric, positive definite matrix $Q \in \mathbb{R}^{n\times n}$, matrix $A \in \mathbb{R}^{m\times n}$,
         vectors $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$
output: minimizer of $q(x) := x^\top Qx + c^\top x$ such that $Ax \le b$
Choose feasible $x^0 \in \mathbb{R}^n$ for QP and $\lambda^0 \in \mathbb{R}^m$. Set $\mathcal{W}_0 := \{i \mid a_i^\top x^0 = b_i\}$.
while $(x^k, \lambda^k)$ is not a KKT-point do
    Set $\lambda_i^{k+1} := 0$ for all $i \notin \mathcal{W}_k$. Solve
        $$\begin{pmatrix} 2Q & A_k^\top \\ A_k^\top & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \lambda_{\mathcal{A}} \end{pmatrix} = \begin{pmatrix} -\nabla f(x^k) \\ 0 \end{pmatrix}. \qquad (2.20)$$
    Distinguish between:
    (a) If $\Delta x^k = 0$ and $\lambda_i^{k+1} \ge 0$ for all $i \in \mathcal{W}_k$: STOP
    (b) If $\Delta x^k = 0$ and $\min\{\lambda_i^{k+1} \mid i \in \mathcal{W}_k\} < 0$:
        choose
            $$q := \arg\min\{\lambda_i^{k+1} \mid i \in \mathcal{W}_k\}$$
        and set
            $$x^{k+1} := x^k, \quad \mathcal{W}_{k+1} := \mathcal{W}_k \setminus \{q\}.$$
    (c) If $\Delta x^k \ne 0$ and $x^k + \Delta x^k$ is feasible for QP:
        set
            $$x^{k+1} := x^k + \Delta x^k, \quad \mathcal{W}_{k+1} := \mathcal{W}_k.$$
    (d) If $\Delta x^k \ne 0$ and $x^k + \Delta x^k$ is infeasible for QP:
        choose
            $$r := \arg\min\left\{\frac{b_i - a_i^\top x^k}{a_i^\top \Delta x^k} \mid i \notin \mathcal{W}_k \text{ s.t. } a_i^\top \Delta x^k > 0\right\}$$
        and set
            $$\alpha_k := \frac{b_r - a_r^\top x^k}{a_r^\top \Delta x^k}, \quad x^{k+1} := x^k + \alpha_k \Delta x^k, \quad \mathcal{W}_{k+1} := \mathcal{W}_k \cup \{r\}.$$
    Set $k := k + 1$.
```

in the dimension of the problem [125], due to the existence of up to 2^n possible working sets, but their practical performance is usually much better. A significant advantage of active set methods is that they can be easily warmstarted. We will profit from this fact in the design of our branch-and-bound algorithms.

2.5.4 Constrained Nonlinear Optimization

There are plenty of classical and well-studied approaches for solving constrained, but not necessarily convex, nonlinear optimization problems. We give a short overview without going into detail. At the end we focus on the Frank-Wolfe method that will be use in the last chapter of this thesis.

Penalty Methods follow the idea of replacing the constrained problem by a sequence of unconstrained optimization problems $P_k := \min\{f(x) + \tau_k P(x)\}$, where $P(x)$ is a non-negative penalty function that is zero if and only if x is feasible and $\tau_k > 0$ is a positive penalty parameter. The use of penalty function and parameter ensures that the objective function is penalized by a perturbation if x is infeasible. The magnitude of perturbation for each unconstrained problem P_k is controlled by the parameter τ_k. For a strictly monotonically increasing sequence of penalty parameters $\{\tau_k\}_k$ it can be shown that for suitable penalty functions every limit point of the sequence of minimizers $\{x^k\}_k$ solves the original constrained problem. Typically P_k is chosen as $P_k := \min\{f(x) + \frac{\tau_k}{2}||h(x)||^2 + \frac{\tau}{2}\sum_{i=1}^m \max^2\{0, g_i(x)\}\}$. Penalty Methods usually suffer from the fact that P_k becomes ill-conditioned for growing τ_k.

Exact Penalty Methods overcome this disadvantage by considering a bounded sequence of penalty parameters. Exact penalty functions have the property that there exists $\tau_{\bar{k}} > 0$, such that if $x^\star$ is a minimizer of f, it is also a minimizer of $P_{\bar{k}}$ for every $k \geq \bar{k}$. This makes it possible to tackle the problem by solving a single (or at least a small number of) unconstrained problem(s) instead of a whole sequence. The penalty functions $P_q(x) := f(x) + \tau||(h(x), \max\{0, g(x)\})||_q$, $q \in [1, \infty)$, are standard choices in practice. It is known that P_q, $q \in [1, \infty)$, is exact for convex optimization problems while they are even exact for general nonlinear optimization problems if in addition $x^\star$ satisfies a constraint qualification. The disadvantage of exact penalty functions is that they are necessarily non-differentiable and methods of non-differentiable optimization, like the bundle method, are needed. A popular reference on non-differentiable methods, especially on bundle methods, is given by [124].

Barrier Methods for inequality constrained problems use a similar strategy. Again a sequence of unconstrained problems is considered. In contrast to Penalty Methods they compute a sequence of feasible instead of infeasible iterates by using barrier terms that keep the iterate in the interior of the feasible region. The barrier term is chosen in a way that it penalizes the function only if the iterate is moving towards the boundary of the feasible region. For example logarithmic barrier terms are often used, resulting in unconstrained problems of the form $B_k := \min\{f(x) - \tau_k \sum_{i=1}^m \ln(-g_i(x))\}$. In this case, we solve B_k for a strictly monotonically decreasing sequence of barrier parameters $\tau_k > 0$ and every limit point of a sequence $\{x^k\}_k$ generated by a Barrier Method is a global minimizer of the original constrained problem. Unfortunately, Barrier Methods also suffer from ill-conditioned problems when τ_k is converges to zero from above.

Augmented Lagrangian Methods, also known as *Multiplier-Penalty Methods*, combine Lagrangian multipliers with penalty functions. For constrained nonlinear programming problems the augmented Lagrangian function can be chosen as $L_a(x, \lambda, \mu, \tau) := f(x) + \frac{\tau}{2}||h(x))||^2 + \mu^\top h(x) + \frac{1}{2\tau}\sum_{i=1}^m(\max^2\{0, \lambda_i + \tau g_i(x)\} - \lambda_i^2)$, which is a continuously differentiable function. It can be shown that there exists $\bar{\tau} > 0$ such that any strict local minimizer $x^\star$ of f is also a strict local minimizer of $L_a(x, \lambda^*, \mu^*, \tau)$ for all $\tau \geq \bar{\tau}$, where (λ^*, μ^*) are the optimal Lagrangian multipliers. Since neither $\bar{\tau}$ nor (λ^*, μ^*) are known a priori, one usually uses iterative procedures to approximate (λ^*, μ^*) and $\bar{\tau}$ in each iteration. In general, Augmented Lagrangian Methods tend to be less vulnerable to ill-conditioning of the generated subproblems.

Sequential Quadratic Programming (SQP) Methods are among the most popular methods for constrained nonlinear programming problems. Their essential idea is to successively model the problem at each iterate x^k by a quadratic programming problem and use its solution to define the new iterate x^{k+1}. SQP Methods can be seen as an application of the Newton Method to the KKT-system and are therefore sometimes also called *Newton-Lagrange Methods.* To be more precise each iteration k consists of solving the quadratic program

$$\begin{aligned} \min\ & \frac{1}{2}s^\top \nabla^2_{xx} L(x^k, \lambda, \mu)s + \nabla f(x^k)s \\ \text{s.t. } & \nabla h_i(x^k)^\top s + h_i(x^k) = 0,\ i = 1, \dots, p \\ & \nabla g_j(x^k)^\top s + g_j(x^k) = 0,\ j = 1, \dots, m, \end{aligned}$$

and setting $x^{k+1} = x^k + s$. This quadratic model consists of a quadratic approximation of the Lagrangian and a linear approximation of the constraints. It can be solved by using standard methods of constrained quadratic programming. Additional strategies inspired by line search and trust-region methods can be used to ensure global convergence from an arbitrary starting point, yielding a very effective instrument to tackle constrained nonlinear programs.

In Chapter 7, we will deal with convex but not necessarily quadratic mixed-integer problems. To solve their continuous relaxations effectively we rely on the following classical method from convex optimization.

Frank-Wolfe Algorithm

The *Frank-Wolfe Algorithm*, also known as the *Conditional Gradient Method* [85] was proposed in 1956 by Marguerite Frank and Philip Wolfe to originally solve convex quadratic optimization problems over a polygonal set and can be extended to convex problems of the more general form CP. This method was in fact one of the breakthroughs in convex optimization. It assumes that the problem:

$$\min\{c^\top x \mid x \in C\}$$

can be solved effectively for any linear objective function $c^\top x$. We assume C to be compact, nonempty and f to be continuously differentiable. Again, the Frank-Wolfe Algorithm can be categorized as a line search method. In each iteration, the search direction is computed by solving a convex problem with a linear objective function. In case C is a polygonal region this is just a linear programming problem. The main idea is to replace the general convex objective function f by its first-order Taylor approximation. More precisely, given an iterate $x^k \in C$, we consider the Taylor expansion of f about x^k:

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

and minimize it over the feasible region C. Since $f(x^k)$ and $\nabla f(x^k)^\top x^k$ are constant, we can replace the objective function by

$$l_k(y) = \nabla f(x^k)^\top (y - x^k),$$

yielding the following subproblem that has to be solved in each iteration k:

$$\min\{\nabla f(x^k)^\top y \mid y \in C\}. \qquad (FW_k)$$

Let y^k denote the minimizer of FW_k. Then $y^k - x^k$ is a feasible direction. Moreover it is a descent direction. To see this, note that the variational inequality $\nabla f(y^k)^\top (y - y^k) \geq 0$ for all $y \in C$ is a sufficient condition for y^k to be a minimizer of f over C. Since y^k is the minimizer of FW_k, we have

$$l_k(y^k) \leq l_k(x^k) = \nabla f(x^k)^\top (x^k - x^k) = 0.$$

This means that $l_k(y^k)$ is either zero or negative. In the first case, we have

$$0 = l_k(y^k) \leq l_k(y) = \nabla f(x^k)^\top (y - x^k) \text{ for all } y \in C,$$

meaning that x^k is already stationary and we can stop. In the second case we have

$$0 > l_k(y^k) = \nabla f(x^k)^\top (y^k - x^k),$$

implying $s^k := y^k - x^k$ to be a descent direction. We can improve our iterate by moving along s^k with any step size $\alpha_k \leq 1$, by the convexity of f.

Some possible choices for the step sizes are:

- an exact step size $\alpha_k := \arg\min_\alpha f(x^k + \alpha d^k)$
- an Armijo Rule, compare (2.12)
- a constant step size $\alpha_k = 1$.
- the simple step size $\alpha_k = \frac{2}{k+2}$.

Assuming C to be compact, the existence of a limit point $\bar{x} \in C$ is guaranteed and we can derive the following convergence theorem:

Theorem 2.27 ([26])**.** *Let $\{x^k\}$ be as sequence generated by the Frank-Wolfe Algorithm:*

$$x^{k+1} = x^k + \alpha_k s^k.$$

If all step sizes α_k satisfy

(a) $f(x^{k+1}) < f(x^k)$, if $\nabla f(x^k) \neq 0$,

(b) If $\nabla f(x^k) \neq 0$ for all k, then

$$\lim_{k\to\infty} \frac{\nabla f(x^k)^\top s^k}{||s^k||} = 0.$$

Then every limit point $\bar{x}$ of $\{x^k\}$ is a stationary point.

In Algorithm 5 we sketch the basic scheme of the Frank-Wolfe Algorithm. The

Algorithm 5: Frank-Wolfe Algorithm

```
input  : convex function $f : \mathbb{R}^n \to \mathbb{R}$ and convex set $C \subseteq \mathbb{R}^n$
output: minimizer of $f(x)$ such that $x \in C$
Set $k = 0$, choose $x^k \in C$.
repeat
    Solve
        $z^k := \min\{\nabla f(x^k)^\top y \mid y \in C\}$.
    Let $y^k$ be the minimizer.
    if ($z^k < 0$) then
        Set $s^k := y^k - x^k$.
        Choose $\alpha_k \leq 1$ using any step size rule.
        Set $x^{k+1} = x^k + \alpha_k s^k$ and $k = k + 1$.
until $z^k = 0$
STOP: $x^k$ is a stationary point.
```

Frank-Wolfe algorithm is widely used for sparse greedy optimization in machine learning and signal processing problems. An overview about the history and recent variants of this algorithm is given in [115].

Chapter 3

Convex Mixed-Integer Nonlinear Programming

In this chapter we are concerned with the solution of convex integer nonlinear optimization problems, i.e., problems of the form INLP, where f and g are assumed to be convex. The reason for considering convex mixed-integer optimization problems is that their continuous relaxation is a pure continuous convex problem, that can be handled effectively because of its good properties, see Section 2.3.2. Indeed, considered as a breakthrough in convex optimization, Nemirovski and Nesterov [161] showed that interior point methods can be used to solve convex optimization problems to a given accuracy with a number of operations that does not grow faster than a polynomial of the problem dimension. Special classes include linear programming (LP), second order cone programming (SOCP) and, under reasonable assumptions, also semidefinite programming (SDP). We note that many convex optimization problems can be cast as one of these. For modern interior point methods in convex optimization, we refer to the book of Ben-Tal and Nemirovski [20]. On the contrary, for many non-convex optimization problems, in the worst case, all known algorithms require a number of operations that is exponential in the problem dimension, e.g., global quadratic optimization is already NP-hard [173]. In fact all algorithms for solving mixed-integer nonlinear programming problems are based on the solution of subproblems which are either integer linear programming (ILP) problems or continuous nonlinear programming (NLP) problems, as we will see in Section 3.2. In both cases the underlying subproblem should be easier to solve than the original problem. Therefore convexity is crucial to tackle the NLP problems.

It is also worth to mention that convexity is often used to deal with non-convex problems, especially for computing dual bounds, e.g., in the context of branch-and-bound. This is often done by convexifying the constraints or using Lagrangian relaxation that is always a convex optimization problem. A detailed survey about non-convex mixed-integer nonlinear programming is given by Burer and Letchford [45]. We will not discuss any problems with non-convexity besides integrality in this work.

To motivate integer programming, let us consider the following two dimensional integer linear optimization problem, taken from [191].

Example 3.1.

$$\begin{aligned} \max\ & 1.00x_1 + 0.64x_2 \\ \text{s.t. } & 50x_1 + 31x_2 \leq 250 \\ & 3x_1 - 2x_2 \geq -4 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned}$$

This integer linear optimization problem has the optimal solution $\bar{x}^\star = (5,0)^\top$. If we relax the integrality restriction on x, i.e., to $x \in \mathbb{R}^2$, we get $\bar{x} = (\frac{376}{193}, \frac{950}{193})^\top$ as the optimal solution of the LP-relaxation. Intuitively, one might think of rounding the continuous minimizer to obtain the optimal integer solution. In our small example this does not help since rounding the components of $\bar{x}$ to the nearest integer gives $\hat{x} = (2,5)^\top$ which is first of all far away from the optimal integer solution but also infeasible, while rounding $\bar{x}$ to the nearest feasible point gives $\tilde{x} = (2,4)$. Figure 3.1 illustrates this problem. Unless the underlying optimization problem has a special structure, rounding does not give an optimal integer solution. Although in general it is not true, there exist special classes of optimization problems where clever rounding of the LP-solution gives the optimal integer solution [112].

This small motivating example gives rise to the question how we can design algorithms to handle integrality and solve integer optimization problems to optimality.

We start this chapter with a section about *Mixed-Integer Linear Programming* problems, i.e., we assume f and g to be linear. After introducing the machinery to deal with such kind of problems, we continue with convex but nonlinear functions f and g and explain how theory and algorithms for the linear case can be adapted and used for convex *Mixed-Integer Nonlinear Programming.*

3.1 Basic Methods for Mixed-Integer Linear Programming

If in MINLP, we require both the objective function f and the constraints g to be linear, it reduces to a Mixed-Integer Linear Program:

$$\min \{c^\top x + d^\top y \mid Ax + By \leq b,\ x \in X \cap \mathbb{Z}^n,\ y \in Y\}, \tag{MILP}$$

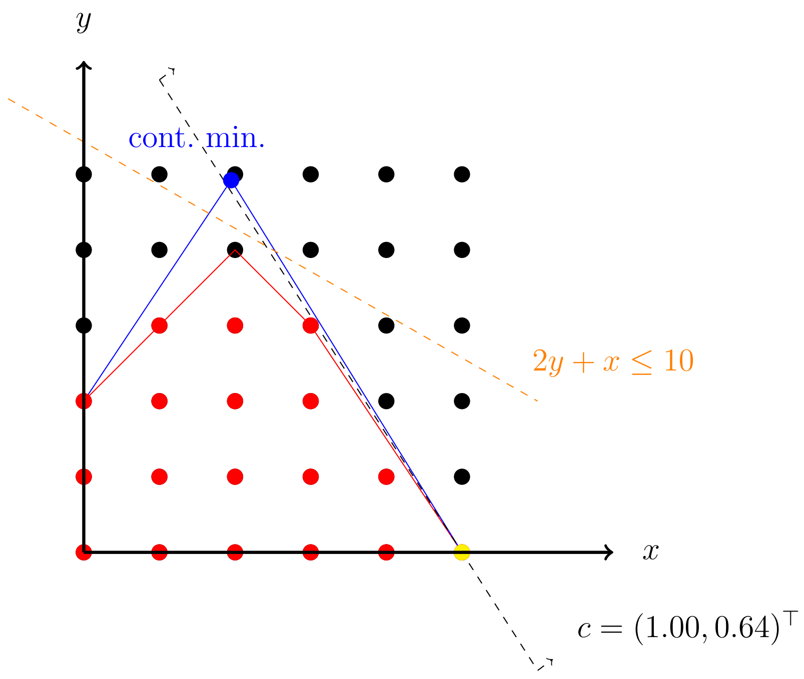

Figure 3.1: The feasible region of the LP-relaxation (blue lines), the integer feasible points (red dots) and its convex hull (red lines), the continuous minimizer $\bar{x} = (376/193, 950/193)^\top$ (blue dot), the integer minimizer $x^\star = (5, 0)^\top$ (yellow dot), a cutting plane (orange dashed line) and the level set $\{x \in \mathbb{R}^2 \mid c^\top x = 5\}$ (black dashed line).

where $c \in \mathbb{R}^n,\ d \in \mathbb{R}^p,\ A \in \mathbb{R}^{m\times n},\ B \in \mathbb{R}^{m\times p},\ b \in \mathbb{R}^m$ and $X \subseteq \mathbb{R}^n, Y \subseteq \mathbb{R}^p$ are polyhedra.

Integer linear programming is known to be NP-hard. In the last decades mainly two algorithmic approaches have been developed to handle integer linear and also nonlinear programming problems: *branch-and-bound* [59, 132, 144], *cutting planes* [63, 96, 97] and *branch-and-cut* [164] as a mixture of both. In nearly every state-of-the-art software at least one of these approaches is implemented. We refer to [159, 175, 191] for a comprehensive overview in the theory of mixed-integer linear programming. Surveys about software for solving mixed-integer linear programming problems are given by [141, 142, 145].

3.1.1 Branch-and-Bound Algorithm

Mixed-Integer Linear Programming problems are generally solved using a linear programming based branch-and-bound algorithm. This is due to the fact that linear programming problems can be solved efficiently in theory by the ellipsoid method [123] as well as by the interior point method [120]. In practice linear programming problems are solved effectively, since the *Simplex Algorithm* was introduced by Dantzig in late sixties [62]. Basic LP-based branch-and-bound is nowadays used in nearly every state-of-the-art MIP software. The main idea of branch-and-bound is based on the concept of *relaxations*. Instead of solving the original mixed-integer programming problem, we remove all integrality restrictions, resulting in a linear programming problem, called the *LP-relaxation*, usually also known as the *continuous relaxation* of the original mixed-integer problem. More generally, relaxing the integrality condition $x \in \mathbb{Z}^n$ in an integer nonlinearoptimization problem INLP results in a problem of the form NLP, i.e., a continuous optimization problem that can be handled by algorithms for nonlinear optimization problems. In particular if the underlying problem is convex it is likely that we can find an efficient algorithm to solve the subproblem.

In this way, we obtain a lower bound on the optimal objective function value of the mixed-integer problem. More generally, there are many ways to compute lower bounds for mixed-integer nonlinear programming problems by using the concept of relaxation. To be more precise, a relaxation is defined as follows.

Definition 3.1 (Relaxation)**.** For a given minimization problem

$$f^* := \min\{f(x) \mid x \in \mathcal{F}\} \tag{P}$$

we call the optimization problem

$$\bar{f}^* := \min\{\bar{f}(x) \mid x \in \bar{\mathcal{F}}\} \tag{R}$$

a *relaxation* for P, if the following two conditions hold:

(i) $\mathcal{F} \subseteq \bar{\mathcal{F}}$ and

(ii) $\bar{f}(x) \leq f(x)$, for all $x \in \mathcal{F}$.

There are two possibilities to relax an optimization problem. The first one is to enlarge the feasible set $\mathcal{F}$, the second one is to find an underestimator for the objective function, such that it always has a value less or equal than the original one. Of course, one is interested in finding relaxations which are easy to solve, but are also close to the original problem. Since there is no need to simplify f, the first possibility is more promising in our case. It follows directly that solving a relaxation always gives a lower bound on $f^* = f(x^\star)$. During the branch-and-bound process, the following properties hold:

(P1) For any relaxation R of P, we have $\bar{f}^* \leq f^*$.

(P2) If any relaxation R of P is infeasible, P is also infeasible.

(P3) If for any relaxation R of P we have $\bar{f}^* = f^*$, then $x^\star$ is also optimal for P.

On the other hand, we observe that for any feasible $x \in \mathcal{F}$, $f(x)$ gives an upper bound on f^*. The goal in branch-and-bound will be to find good lower and upper bounds on f^*.

Algorithm 6: LP-based branch-and-bound for MILP

```
input  : Polyhedron $P = \{(x, y) \in X \times Y \mid Ax + By \leq b\}$, $c \in \mathbb{R}^n$
output: min $c^\top x$ s.t. $(x, y) \in P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$
Set
    $P = \{(x, y) \in X \times Y \mid Ax + By \leq b\}$,
    $L = \{P\}$,
    $U = \infty$.
while $L \neq \emptyset$ do
    Choose $P \in L$ and set $L := L \setminus \{P\}$.
    Solve
        $\min\{c^\top x + d^\top y \mid (x, y) \in P\}$.        $(\star)$
    if $(\star)$ is infeasible then
        Set $f^* = \infty$.
    else
        Let $(x^\star, y^*)$ denote the optimal solution of $(\star)$ and set
        $f^* = c^\top x^\star + d^\top y^*$.
    if $f^* \geq U$ then
        Continue
    if $(x^\star, y^*) \in P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$ then
        if $f^* < U$ then
            Set $(\bar{x}, \bar{y}) = (x^\star, y^*)$ and $U = f^*$.
            Continue
    Choose $i \in \{1, \ldots, n\}$ such that $x_i^\star \notin \mathbb{Z}$ and let
        $P_1 := \{(x, y) \in P \mid x_i \geq \lceil x_i^* \rceil\}$ and $P_2 := \{(x, y) \in P \mid x_i \leq \lfloor x_i^* \rfloor\}$.
    Set $L = L \cup \{P_1, P_2\}$.
if $U = \infty$ then
    STOP: The problem is infeasible.
else
    STOP: $(\bar{x}, \bar{y})$ is optimal with objective function value $U$.
```

Reduced Enumeration by Branching and Bounding Since complete enumeration in combinatorial optimization leads to a combinatorial explosion, meaning that for increasing problem dimension, the number of potential optimal solutions is increasing exponentially in the dimension, we are seeking for some more intelligent algorithms for finding the optimal solution. The basic principle of branch-and-bound is to create a reduced enumeration tree of all feasible candidates by excluding those in advance for which it is known that they cannot be optimal. This can be achieved by using the divide and conquer strategy combined with a *Bounding* procedure: the original problem P is divided into k more restrictive subproblems $P_1, \dots, P_k$ such that

$$\mathcal{F} = \bigcup_{i=1}^{k} \mathcal{F}_i,$$

where $\mathcal{F}, \mathcal{F}_1, \dots, \mathcal{F}_k$ are the feasible sets of P, $P_1, \dots, P_k$, respectively. Usually the union is chosen to be disjunct. It is clear that

$$z = \min\{z_i, \dots, z_k\},$$

where z is the optimal objective function value of P and z_i is the optimal objective function values of P_i for all $i = 1, \dots, k$. In other words the sets of feasible solutions form a (disjunctive) union of the original set of feasible solutions. Since this leads to k different branches in the tree, we call this step *Branching.* Every subproblem created in this way is called a *node* in the branch-and-bound tree. The original mixed-integer problem is therefore called the *root node.* In Algorithm 6, we describe an LP-based branch-and-bound scheme, where the number of subproblems generated at each node is two. This formulation was proposed by Dakin [59] in 1965 and that way of branching is known as *dichotomy branching.* Every time we find a feasible mixed-integer solution this is a valid *upper bound* for the optimal objective function value z. On the other hand every relaxation yields a *lower bound.* We have the following observation. If $\bar{z}_k$ and $\underline{z}_k$ denote the upper and lower bound on z_k, respectively, we can conclude that $\bar{z} = \min_k \bar{z}_k$ is an upper bound on z. Therefore we can successively compare upper and lower bounds to potentially exclude branches of the tree. If no upper bound is known at the root node, it is set to infinity. Usually the expressions *primal bound* and *dual bound* are used instead of upper and lower bounds to be consistent if switching from minimization to maximization problems.

For each subproblem i we solve an appropriate relaxation. Three different cases may occur:

(i) the solution is feasible for P.

(ii) the solution is feasible for P_i but not feasible for P.

(iii) P_i is infeasible.

Everytime case (i) occurs, we have found a feasible solution to the original mixed-integer problem and according to property (P2) it is not necessary to continue branching on this node. This is due to the fact that in every consecutive node, originating from the current one, the objective function value is non-decreasing since we restrict the set of feasible solutions. We further check if the solution provides an improved solution, i.e., it has a better objective function value than the best one known so far. If so, it yields an improved primal bound and we update our best solution. For any upcoming node we know that we will never have to accept a feasible mixed-integer solution of value worse than our best primal bound. Otherwise, if we do not get an improved solution, we simply proceed with the tree search.

If case (ii) occurs, we use the fact that every relaxation yields a dual bound. We can compare it to the best primal bound found so far. Now, if the dual bound is greater or equal to the best known primal bound, we do not need to consider that branch of the tree and therefore cut it off from the search space. This process of comparing dual and primal bounds is known as *Bounding*. Cutting off a branch is denoted also as *Pruning*. In the other case, i.e., the dual bound is strictly less than the primal bound, that branch may still contain a better solution and the current subproblems are again divided into k more restrictive ones. This procedure is continued until we either reach a point at which we have solved or pruned all of those subproblems. The best solution at the end is proven to be optimal for the original mixed-integer problem, or if no feasible solution has been found, the problem is infeasible.

Finally, if case (iii) occurs, we can prune the node due to property (P3), since there will be for sure no feasible solution for the original problem in that branch. A typical branch-and-bound tree is illustrated in Figure 3.2.

To sum up, there are mainly three crucial components that can improve the performance of a generic branch-and-bound algorithm, namely the *Branching Strategy*, the *Node Selection* and the *Bound Computation*. We will shortly discuss the possibilities.

Branching Strategies Besides a *random branching strategy*, where a random fractional variable is chosen, *Most-Fractional-Branching* is the most intuitive branching strategy, picking some fractional variable in the LP-solution whose distance to the next integer is maximal. In *Strong Branching* [10] one aims at maximizing the expected increase of the lower bound to prune as early as possible. This is done by solving all nodes originating from branching on each fractional variable. Looking at the optimal objective function value of the current node and comparing it to the solution values of the nodes generated by branching on one of the fractional variables, the branching rule chooses a variable that is promising.

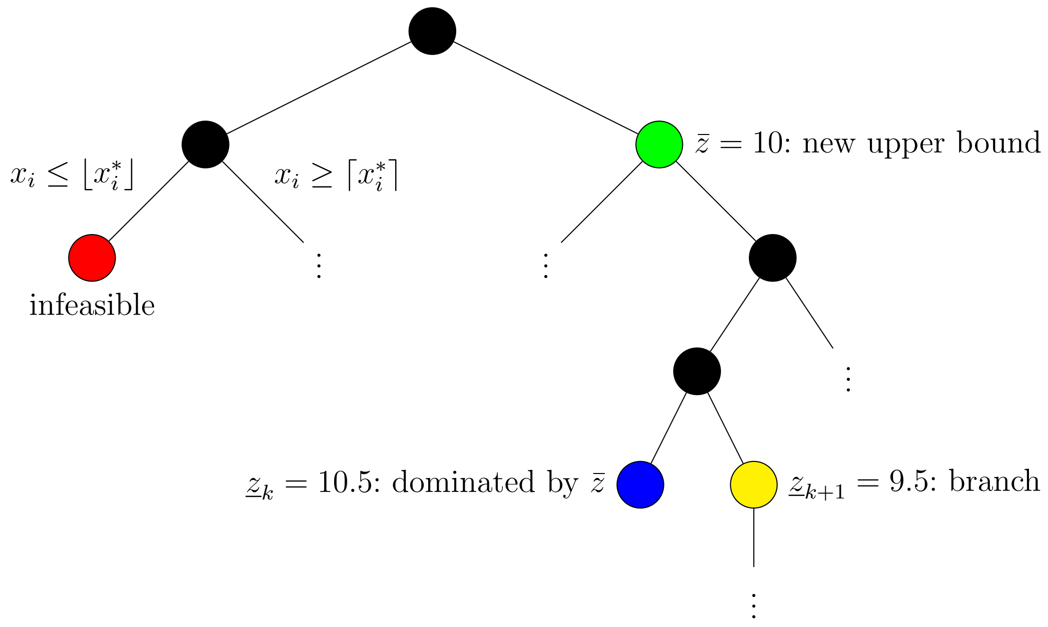
$x_i \leq \lfloor x_i^* \rfloor$
$x_i \geq \lceil x_i^* \rceil$
$\bar{z} = 10$: new upper bound
infeasible
$\underline{z}_k = 10.5$: dominated by $\bar{z}$
$\underline{z}_{k+1} = 9.5$: branch

Figure 3.2: Illustration of a branch-and-bound tree using dichotomy branching: the red node indicates the infeasibility of the node relaxation, thus it can be pruned. The green node indicates that a feasible (mixed-)integer solution has been found. In the upper green node a new primal bound of value 10 has been found. In the blue node the dual bound of value 10.5 exceeds the current primal bound and we can also prune, while in the yellow node we need to continue branching.

This strategy requires the exact solution of a lot of nodes at every level of the branching tree. The computational effort can be reduced by solving the nodes only approximately. A similar strategy is devised in [21], called *Pseudo-Cost Branching* and can be seen as a computationally cheaper version of strong branching. The idea is to replace the exact solution values at the nodes by an estimation of these values, gained from the branchings on that variable done so far in the search tree. Every time a variable has been chosen for branching, the change in the objective function is stored. Then a variable that is predicted to have the biggest change in the objective function based on stored values is chosen. *Reliability Branching*, introduced by Achterberg et al. [3], combines strong branching and pseudo-cost branching. Since in pseudo-cost branching no data is available for initializing the estimated costs, it is useful to perform strong branching at the beginning, until every variable has been branched on k times, and then switch to pseudo-cost branching. Again the computational effort can be controlled by limiting the number of variables for which strong branching will be executed, if $k \geq 2$. For example this can be done by ranking the variables by some score number and taking the best ones.

Node Selection Strategies After processing the current node, we have to decide which open node will be processed next. Two common node selection strategies are *depth-first search* and *best-first search*. For a fixed branching strategy, depth-first search selects the last node put in the list, which corresponds to the deepest node of the enumeration tree. Best-first search selects the node with the smallest lower bound since it might be more promising to search for the optimal solution in that branch. While a depth-first search leads to a quick initial primal bound, a best-first search might reduce the number of nodes in the enumeration tree. Clearly, on the other hand, depth-first may result in the enumeration of many nodes if no good primal bound can be found quickly while a best-first search might take longer to find any feasible solution before reaching the bottom of the enumeration tree. Both strategies can also be combined in *Two-Phase-Methods*, e.g., *Diving Methods*, as proposed in [143]. After starting with a depth-first search, until one or a few feasible solutions have been found, it continues with a best-first search aiming at proving local optimality.

Bound Computation Strategies We have to choose which appropriate relaxation for the nodes we want to use to compute the dual bounds. It is intuitive that tighter bounds are more costly to compute but can lead to a smaller overall search tree. In case of an integer linear programming problem it is obvious to use the continuous relaxation. The same trade-off holds for the primal bound. The usage of *Primal Heuristics*, i.e., algorithms that compute feasible solutions quickly without assuring optimality, can lead to good approximative solutions that can be used as primal bounds to reduce the gap between dual and primal bounds but can be time consuming.

3.1.2 Cutting Plane Algorithm

The motivation of the Cutting Plane Algorithm for mixed-integer linear programming problems comes from a well-known result in integer programming, namely that minimizing a linear function over an arbitrary set is the same when replacing the set by its convex hull:

Theorem 3.1 ([191]). *Let $X \subseteq \mathbb{R}^n$ and $c \in \mathbb{R}^n$. Then we have*

$$\min\{c^\top x \mid x \in X\} = \min\{c^\top x \mid x \in \mathit{conv}(X)\}.$$

In particular, if $X = P \cap \mathbb{Z}^n$, where P is a polyhedron, $P' := \operatorname{conv}(X)$ is also a polyhedron, i.e., there exists a matrix $\tilde{A}$ and a vector $\tilde{b}$ of appropriate size such that $P' = \{x \in \mathbb{R}^n \mid \tilde{A}x \leq \tilde{b}\}$. Knowing $\tilde{A}$ and $\tilde{b}$ in advance, would give a complete description of the convex hull and in theory such an integer program could be reformulated as a linear programming problem and solved by using any algorithm for linear programming. But in general P' might need an exponentially large number of linear inequalities to be described. In practice it is often hard to find an explicit description of P' for a set X due to numerical issues, so one is interested in approximating P'. The above result is also true for any mixed-integer linear program MILP. The Cutting Plane Algorithm aims at approximating P' by the usage of cutting planes.

Definition 3.2 (Valid Inequality, Cutting Plane)**.** Let P be a polyhedron and $P' := \mathrm{conv}(P \cap \mathbb{Z}^n)$. A linear inequality $\pi^\top x \leq \pi_0$ is a *valid inequality* for P if it holds for all $x \in P$. A linear inequality is a *Cutting Plane* for P, if it is valid for P' but not for P.

In Figure 3.1 a possible cutting plane is shown. There are mainly two reasons why adding cutting planes to P from the beginning is not always reasonable. First, there is generally an exponential number of such additional constraints. Second, by adding additional constraints the LP-relaxations become successively harder to solve. We only want to add these constraints if they are violated by any fractional solution not contained in P'. Therefore it is important to study ways of generating cutting planes for P whenever needed.

Definition 3.3 (Separation Problem)**.** For a polyhedron P a *separation problem* is the following problem:
Given any $x^\star \in \mathbb{R}^n$, find a valid inequality $\pi^\top x \leq \pi_0$ for P' such that $\pi^\top x^\star > \pi_0$ or decide that none exists.

For any $x^\star \notin P'$, the separation algorithm gives a cutting plane for P. In general, efficient optimization and efficient separation are equivalent, i.e., the following theorem holds:

Theorem 3.2 ([191])**.** *For a given class of optimization problems with a linear objective function* $\min\{c^\top x \mid x \in X \subseteq \mathbb{R}^n\}$*, there exists an efficient (polynomial) algorithm, if and only if there exists an efficient algorithm for the separation problem associated with the problem class.*

The main idea of the cutting plane algorithm is to tighten the original formulation of the LP-relaxation by removing undesired fractional solutions during the solution process without the undesired side effect of creating additional subproblems, such as in a branching process. Given the fractional LP solution $x^\star \in P$ found by solving the LP-relaxation, it is tested for being feasible in the integer components. If it is not, the separation problem is solved, i.e., a valid linear inequality for P' is determined which cuts off the non-integer point $x^\star$. Once such a cutting plane is found, it is added to the LP-relaxation to tighten the feasible set and the relaxation is solved once again. In an iterative fashion, we alternate between adding cutting planes and solving the LP-relaxation with the aim of finding an integer solution. Each time a cutting plane is added, the current feasible set gets closer to the desired convex hull.

In Algorithm 7, we sketch a general cutting plane algorithm for problems of the form MILP. It might be the case that Algorithm 7 terminates without finding an integer solution because no further cutting plane can be found. In this case the improved formulation can still be passed to any branch-and-bound algorithm.

Algorithm 7: Generic Cutting-Plane Algorithm

```
input  : Polyhedron $P = \{(x, y) \in X \times Y \mid Ax + By \leq b\}$, $c \in \mathbb{R}^n$
output: min $c^\top x$ s.t. $(x, y) \in P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$.

repeat
    Minimize $c^\top x + d^\top y$ over $P$.
    Let $(x^\star, y^*) \in P$ denote its optimal solution.
    if $x^\star \in \mathbb{Z}^n$ then
        STOP: $x^\star$ is optimal.
    else
        Solve the separation problem for $P$.
    if a cutting plane $\pi^\top(x, y) \leq \pi_0$ is found then
        Set $P := \{(x, y) \in P \mid \pi^\top(x, y) \leq \pi_0\}$.
until no cutting plane was found
```

It remains the question of how to find a cutting plane in Step 3 of Algorithm 7. In the 1950s Ralph Gomory and Vaclav Chvátal independently found out that the following observation holds:

Lemma 3.3 ([191]). *Given $X = P \cap \mathbb{Z}^n$, where $P = \{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$ and $u \in \mathbb{R}^m_+$, the linear inequality $\sum_{j=1}^n \lfloor u^\top a_j \rfloor x_j \leq \lfloor u^\top b \rfloor$ is valid for X, where $a_1, \ldots, a_n \in \mathbb{R}^m$ denote the columns of $A \in \mathbb{R}^{m \times n}$.*

These linear inequalities are known as *Chvátal-Gomory Cuts*. Adding one of them to P is called the *Chvátal-Gomory Procedure*. It can be shown that these cuts are sufficient to generate all possible valid inequalities for X.

Theorem 3.4 ([191]). *Every valid inequality for $P \cap \mathbb{Z}^n$ can be obtained by applying the Chvátal-Gomory procedure a finite number of times.*

The first cutting plane algorithm was introduced by Dantzig, Fulkerson and Johnson in 1954 [63], to solve a 54-city instance of the *Traveling Salesman Problem*. In 1958 Ralph Gomory [96] formulated an algorithm, based on the Chvátal-Gomory Procedure, for successively generating Chvátal-Gomory Cuts directly out of the optimal simplex tableau for any fractional LP solution found in Step 1, cutting it off. It was the first finite cutting plane algorithm presented for solving general pure integer linear programming problems of the form $\min\{c^\top x \mid x \in P \cap \mathbb{Z}^n\}$, where P is defined by linear inequalities $Ax \leq b$, and A, b are required to be integer valued. Later, in 1960 Gomory [97] introduced the *Gomory Mixed-Integer Cuts* and extended the algorithm to general mixed-integer linear programming problems. These cuts are based on a disjunctive argument. However, iteratively adding Gomory Mixed-Integer Cuts will only guarantee finite convergence if also the objective function is required to be integer valued.

In the last decades, different types of cutting planes have been studied besides the *Chvátal-Gomory Cuts* and *Gomory Mixed-Integer Cuts*, e.g., *Mixed-Integer Rounding Cuts* [150], *Lift-and-Project (Disjunctive) Cuts* [13, 14], *Split Cuts* [54], *Intersection Cuts* [12], $\{0, \frac{1}{2}\}$-*Cuts* [49]. A detailed survey on cutting planes for mixed-integer linear programming is given by Cornuéjols [55]. Their effectiveness depends on the underlying problem and usually different types of cuts are combined. The theory of cutting planes is deep and extensive. It is also generally accepted to be the most important contributor to the computational advances that have been made in integer programming over the last several years. An interesting study by Bixby [31] shows the improvements in solving mixed-integer problems using all innovations over the last decades.

3.1.3 Branch-and-Cut Algorithm

Combining branch-and-bound with the cutting plane algorithm results in the so-called *branch-and-cut* algorithm and merges the advantages of both approaches. In the root node the LP-relaxation is solved. If the solution is not integer feasible, the separation problem is solved and cutting planes are added to the problem if possible and we repeat as in a cutting plane algorithm. If no cutting plane can be found we switch to a branching procedure as in branch-and-bound. In commonly used optimization software for integer linear programming problems branch-and-cut algorithms can be considered as state-of-the-art, see Section 3.3.

3.2 Methods for Convex Mixed-Integer Nonlinear Programming

We are now interested in convex mixed-integer nonlinear programming problems, i.e., problems of the form MINLP:

$$\begin{aligned} z_{CMINLP} := \min_{x,y}\ & f(x,y) \\ \text{s.t. } & g(x,y) \leq 0 \qquad\qquad \text{(CMINLP)} \\ & x \in X \cap \mathbb{Z}^n,\ y \in Y, \end{aligned}$$

where additionally the objective function f, the constraint functions g and the sets X, Y are assumed to be convex.

Many optimization problems arising in real world applications can be formulated as convex mixed-integer nonlinear programs of the form CMINLP. Allowing both integrality and nonlinearity makes this class of problems extremely hard. In fact, MINLP comprises the NP-hard subclasses of mixed-integer linear programming (MILP) and general nonlinear programming (NLP). The restriction to convex MINLP preserves NP-hardness, as MILP is still contained as a special case.

In literature, there are mainly five approaches for this kind of problems, which we will explain in the following subsections: *NLP-based branch-and-bound* [105], *outer approximation* [75, 80], *general benders cecomposition* [92], *extended cutting planes* [188] and *LP/NLP-based branch-and-bound* [169]. A detailed survey of algorithms and software for solving convex MINLP is given by Bonami, Kilinç and Linderoth [36]. Especially branch-and-bound and outer approximation will build the main components for our quadratic outer approximation algorithm, presented in Chapter 6. The idea is always to reduce the original problem to a sequence of ILP and/or NLP problems and use the methodology of Chapter 2.

3.2.1 Branch-and-Bound

NLP-based branch-and-bound works analogously to Algorithm 6. Instead of considering its linear programming relaxation, the following nonlinear programming relaxation is used to to compute the dual bounds in Step 4:

$$\begin{aligned} \min_{x,y} \ & f(x,y) \\ \text{s.t.} \ & g(x,y) \leq 0 \\ & x \in X, \ y \in Y. \end{aligned} \tag{NLP-R}$$

This relaxation is also known as the *NLP-relaxation* of a mixed-integer programming problem. Therefore one may think of choosing any algorithm from nonlinear programming that can solve the continuous relaxation. For this purpose, all algorithms presented in the following chapters make use of one or more algorithms presented in Section 2.5.

3.2.2 Outer Approximation

Outer approximation was introduced by Duran and Grossmann in 1986 [75] and extended to general convex mixed-integer nonlinear programming problems by Fletcher and Leyffer in 1994 [80]. The basic concept is to successively build up a mixed-integer linear programming problem that is equivalent to the original problem by linearizing both the objective function as well as the constraints at

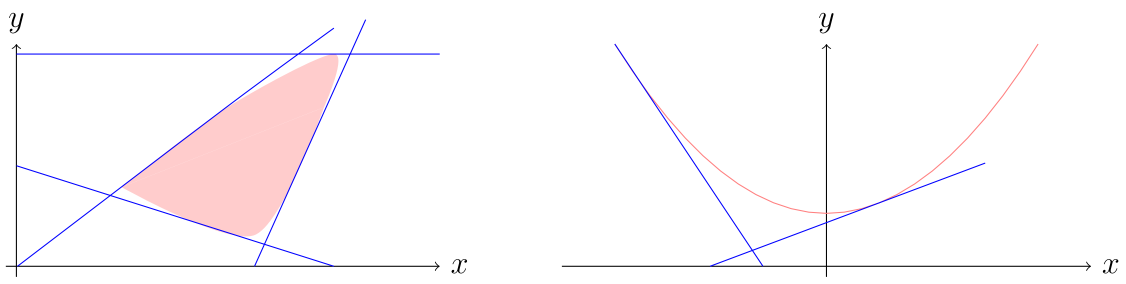

Figure 3.3: Polyhedral outer approximation of the feasible region (left) and objective function (right).

certain points. Figure 3.3 illustrates an outer approximation of feasible region and objective function. The approach requires the following assumptions:

A1 $X \neq \emptyset$ is compact.

A2 f, g are convex and continuously differentiable.

A3 for any fixed integer $\bar{x} \in X \cap \mathbb{Z}^n$, a constraint qualification, e.g., Slater's condition, holds at the solution of the resulting NLP-relaxation:

$$\begin{aligned} z_{NLP(\bar{x})} := \min\ & f(\bar{x}, y) \\ \text{s.t. } & g(\bar{x}, y) \leq 0 \\ & y \in Y, \end{aligned} \tag{NLP($\bar{x}$)}$$

or, in case NLP($\bar{x}$) is infeasible, at the solution of the following *feasibility problem*, that minimizes the violation of the constraints g:

$$\begin{aligned} z_{NLPF(\bar{x})} := \min\ & \sum_{j=1}^{m} g_j(\bar{x}, y)^+ \\ \text{s.t. } & y \in Y, \end{aligned} \tag{NLPF($\bar{x}$)}$$

with $g_j(\bar{x}, y)^+ := \max\{0, g_j(\bar{x}, y)\}$.

Introducing a real variable η and using the convexity of f and g, we end up with a mixed-integer linear programming problem for which it can be shown that it is equivalent to CMINLP:.

$$\begin{aligned}
z_{OA} := \min\ & \eta \\
\text{s.t. } & f(\bar{x}, y(\bar{x})) + \nabla f(\bar{x}, y(\bar{x}))^\top \begin{pmatrix} x - \bar{x} \\ y - y(\bar{x}) \end{pmatrix} \leq \eta,\ \forall \bar{x} \in T^* \\
& g_j(\bar{x}, y(\bar{x})) + \nabla g_j(\bar{x}, y(\bar{x}))^\top \begin{pmatrix} x - \bar{x} \\ y - y(\bar{x}) \end{pmatrix} \leq 0,\ \forall j = 1, \dots, m,\ \forall \bar{x} \in T^* \\
& g_j(\tilde{x}, y(\tilde{x})) + \nabla g_j(\tilde{x}, y(\tilde{x}))^\top \begin{pmatrix} x - \tilde{x} \\ y - y(\tilde{x}) \end{pmatrix} \leq 0,\ \forall j = 1, \dots, m,\ \forall \tilde{x} \in S^* \\
& x \in X \cap \mathbb{Z}^n, y \in Y, \eta \in \mathbb{R}
\end{aligned} \tag{MILP-OA}$$

where $T^* = \{x \in X \cap \mathbb{Z}^n \mid \text{NLP}(x) \text{ is feasible}\}$ consists of all integer points in X for which the subproblem is feasible, $S^* = \{x \in X \cap \mathbb{Z}^n \mid \text{NLP}(x) \text{ is infeasible}\}$ consists of those for which it is infeasible and $y(x)$ denotes an optimal solution of NLP(x) or NLPF(x), respectively.

Theorem 3.5 ([36, 75, 80]). *Assuming A1-A3 hold, then $z_{OA} = z_{CMINLP}$. All optimal solutions of CMINLP are optimal solutions of MILP-OA.*

From a practical point of view formulating MILP-OA requires the a priori knowledge of T^* and S^*, thus also the knowledge of the solutions of all the NLP subproblems for each $x^k \in X \cap \mathbb{Z}^n$ at some iteration k of the outer approximation scheme. Algorithmically, it is therefore convenient to build up the mixed-integer linear problem step by step, alternating between solving the nonlinear programming problem NLP(x^k), yielding a point (x^k, y^k) about which we linearize the objective and the constraint functions, and the resulting relaxation MILP-OA$_k$ of the mixed-integer linear programming problem MILP-OA at each iteration k:

$$\begin{aligned}
z_{OA_k} := \min\ & \eta \\
\text{s.t. } & f(x^h, y^h) + \nabla f(x^h, y^h)^\top \begin{pmatrix} x - x^h \\ y - y^h \end{pmatrix} \leq \eta,\ h \in T^k \\
& g_j(x^h, y^h) + \nabla g_j(x^h, y^h)^\top \begin{pmatrix} x - x^h \\ y - y^h \end{pmatrix} \leq 0,\ j = 1, \dots, m,\ h \in T^k \\
& g_j(x^l, y^l) + \nabla g_j(x^l, y^l)^\top \begin{pmatrix} x - x^l \\ y - y^l \end{pmatrix} \leq 0,\ j = 1, \dots, m,\ l \in S^k \\
& x \in X \cap \mathbb{Z}^n, y \in Y, \eta \in \mathbb{R},
\end{aligned} \tag{MILP-OA$_k$}$$

where $T^k := \{h \leq k \mid y^h \text{ is the optimal solution of NLP}(x^h)\}$ and analogously $S^k := \{l \leq k \mid y^l \text{ is the optimal solution of NLP}(x^l)\}$. It is clear that on the one hand the solution of every feasible problem NLP(x^k) gives a primal bound for CMINLP since it is feasible and on the other hand the solution of MILP-OA$_k$ gives an increasing sequence of dual bounds for CMINLP. The iterative procedure stops when primal and dual bounds are within a specified tolerance. Finite convergence is ensured by the following theorem.

Theorem 3.6 ([75, 80])**.** *If assumptions A1-A3 hold and $|X| < \infty$, then Algorithm 8 terminates in a finite number of iterations at the optimal solution of CMINLP or with an indication that it is infeasible.*

We finally give a sketch of the general outer approximation scheme in Algorithm 8. Note that there exist instances for which Algorithm 8 needs exponentially many

Algorithm 8: Outer approximation algorithm

```
input  : $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}$, $g : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ convex and differentiable
output: $\min\{f(x,y) \mid g(x,y) \leq 0,\ (x,y) \in X \cap \mathbb{Z}^n \times Y\}$
Set $k := 0$, $T^k := \emptyset, S^k := \emptyset$, $U^k := \infty$, $L^k := -\infty$ and $\varepsilon > 0$. Choose
$x^k \in X \cap \mathbb{Z}^n$.
repeat
    Solve NLP($x^k$) (or NLPF($x^k$), if subproblem is infeasible). Let $y^k$ be
    the solution.
    Linearize $f$ and $g$ about $(x^k, y^k)$, set $T^{k+1} := T^k \cup \{k\}$ (or
    $S^{k+1} := S^k \cup \{k\}$, if subproblem is infeasible).
    if NLP($x^k$) is feasible and $f(x^k, y^k) < U^k$ then
        Set $(x^\star, y^*) = (x^k, y^k)$ and $U^{k+1} = f(x^k, y^k)$.
    else
        $U^{k+1} = U^k$.
    Solve MILP-OA$_k$.
    Let $(x^{k+1}, y^{k+1})$ be the solution and $L^{k+1} := z_{OA_k}$. Set $k = k + 1$.
until $U^k - L^k < \varepsilon$
STOP: $(x^k, y^k)$ is the optimal solution.
```

iterations to converge, see, e.g., [111], meaning we need to solve exponentially many mixed-inter linear programming problems. Figure 3.4, taken from [111], shows the empty set $B_n := \{x \in \{0,1\}^n \mid \sum_{i=1}^n (x_i - \frac{1}{2})^2 \leq \frac{n-1}{4}\}$ as the intersection of the vertices of an n-dimensional hypercube with a ball of radius $\frac{\sqrt{n-1}}{2}$ centered in $(\frac{1}{2}, \ldots, \frac{1}{2})$. Algorithm 8 would need 2^n iterations to converge, since only one vertex can be cut by an outer approximation constraint in each iteration.

3.2.3 Generalized Benders Decomposition

Generalized Benders Decomposition for CMINLP was devised by Geoffrion [92] in 1972, and is very similar to the presented outer approximation algorithm, but

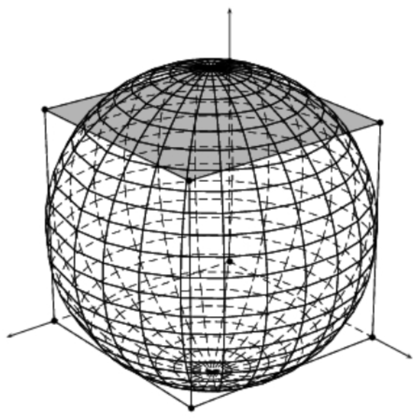

Figure 3.4: Worst-case example: exponentially many iterations are needed by outer approximation to detect infeasibility [111].

was proposed earlier. The difference between both approaches is the construction of the mixed-integer linear programming problems for computing the dual bounds. At iteration k, having solved NLP(x^k) with the corresponding solution y^k, instead of adding all linearizations of g about (x^k, y^k), the idea is to add a cut that turns out to be an aggregation of these linearizations, thus yielding a potentially weaker bound than the one resulting from outer approximation. This is why the majority of available solvers prefer outer approximation to the Generalized Benders Decomposition to solve convex MINLP (see Section 3.3). In [1], Abishek, Leyffer and Linderoth use the resulting Benders cuts to aggregate inequalities in their LP/NLP-based branch-and-bound software *FilMINT*. We will give a brief derivation of the MILP. Again, we require the assumptions A1-A3 on CMINLP and use arguments from nonlinear duality theory. We start with the dual representation of NLP(x^k). According to Definition 2.7 the Lagrangian dual is

$$z_{DNLP(x^k)} := \max_{\lambda \in \mathbb{R}^m_+} \min_{y \in Y} f(x^k, y) + \lambda^\top g(x^k, y). \qquad (\text{DNLP}(x^k))$$

Using assumptions A1-A3 we have that strong duality holds for NLP(x^k), i.e., $z_{NLP(x^k)} = z_{DNLP(x^k)}$. Thus, we have

$$\begin{aligned} z_{CMINLP} &= \min_{x^k \in V} z_{NLP(x^k)} \\ &= \min_{x^k \in V} \left(\max_{\lambda \in \mathbb{R}^m_+} \min_{y \in Y} L(x^k, y, \lambda) \right) \\ &= \min\{\alpha \mid \alpha \geq \min_{y \in Y} L(x^k, y, \lambda)\ \forall \lambda \geq 0,\ \alpha \in \mathbb{R},\ x^k \in V\}, \end{aligned}$$

where $V := \{x \in X \cap \mathbb{Z}^n \mid \exists y \in Y : g(x, y) \leq 0\}$ is the set of all feasible integer assignments for which there exists a feasible continuous part. Since V is not known explicitly, $x^k \in V$ has to be reformulated by inequality constraints. This is done by the feasibility problem NLPF(x^k), knowing that $x^k \in V$ if and only if NLPF(x^k) has a positive minimum. Considering its Lagrangian dual and again using strong duality yields the condition $x^k \in V$ if and only if

$$0 \geq \min_{y \in Y} \mu^\top g(x^k, y) \text{ for all } \mu \in \Lambda := \{\mu \in \mathbb{R}^m_+ \mid \sum_{i=1}^m \mu_i = 1\}.$$

Finally, we get the following MILP which is equivalent to CMINLP

$$\begin{aligned} z_{GBD} := \min\ & \eta \\ \text{s.t. } & \min_{y \in Y} L(x, y, \lambda) \leq \eta\ \forall \lambda \geq 0 \\ & \min_{y \in Y} \mu^\top g(x, y) \leq 0\ \forall \mu \in \Lambda \\ & x \in X \cap \mathbb{Z}^n,\ \eta \in \mathbb{R}. \end{aligned} \tag{MILP-GBD}$$

Also here, it is not useful to solve this problem directly, since there are infinitely many constraints. By linearization, we get the relaxation

$$\begin{aligned} \min\ & \eta \\ \text{s.t. } & L(x^h, y^h, \lambda^h) + \nabla_x L(x^h, y^h, \lambda^h)^\top (x - x^h) \leq \eta,\ h \in T^k \\ & (\mu^l)^\top \left(g(x^l, y^l) + \nabla_x g(x^l, y^l)^\top (x - x^l) \right) \leq 0,\ l \in S^k \\ & x \in X \cap \mathbb{Z}^n,\ \eta \in \mathbb{R}, \end{aligned} \tag{MILP-GBD$_k$}$$

where T^k and S^k are defined as for MILP-OA$_k$.

Finally we want to show that the cut, added to the mixed-integer linear programming problem, is an aggregation of those in outer approximation. From the KKT-conditions of NLP(x^k) we have

$$\nabla_y f(x^k, y^k) + (\lambda^k)^\top \nabla_y g(x^k, y^k) = 0 \tag{3.1}$$

$$(\lambda^k)^\top g(x^k, y^k) = 0, \tag{3.2}$$

where (y^k, λ^k) is the pair of primal and dual solution. The linearizations used in outer approximation are:

$$f(x^k, y^k) + \nabla f(x^k, y^k)^\top \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \leq \eta \tag{3.3}$$

$$g(x^k, y^k) + \nabla g(x^k, y^k)^\top \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \leq 0. \tag{3.4}$$

Using the equations (3.1),(3.2), and adding the inequalities of (3.3),(3.4) where the second one is multiplied by λ^k yields

$$\begin{aligned} & f(x^k, y^k) + \nabla_x f(x^k, y^k)^\top (x - x^k) + (\lambda^k)^\top \nabla_x g(x^k, y^k)(x - x^k) \leq \eta \\ \Leftrightarrow & L(x^k, y^k, \lambda^k) + \nabla_x L(x^k, y^k, \lambda^k)^\top (x - x^k) \leq \eta. \end{aligned}$$

Algorithm 9: General Benders Decomposition

```
input  : $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}$, $g : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ convex and differentiable
output: $\min\{f(x,y) \mid g(x,y) \leq 0,\ (x,y) \in X \cap \mathbb{Z}^n \times Y\}$
Set $k := 0$, $T^k := \emptyset, S^k := \emptyset$, $U^k := \infty$. Choose $x^k \in X \cap \mathbb{Z}^n$.
repeat
    Solve NLP($\bar{x}$) (or NLPF($\bar{x}$), if subproblem is infeasible). Let $y^k$ be the solution.
    Linearize $f$ and $g$ about $(x^k, y^k)$, set $T^{k+1} := T^k \cup \{k\}$ (or $S^{k+1} := S^k \cup \{k\}$, if subproblem is infeasible).
    if NLP($\bar{x}$) is feasible and $f(x^k, y^k) < U^k$ then
        Set $(x^\star, y^*) = (x^k, y^k)$ and $U^{k+1} = f(x^k, y^k)$.
    else
        $U^{k+1} = U^k$.
    Solve MILP-GBD$_k$.
    if MILP-GBD$_k$ is feasible then
        Let $(x^{k+1}, y^{k+1})$ be the solution. Set $k = k + 1$.
until MILP-GBD$_k$ is infeasible
STOP: $(x^k, y^k)$ is the optimal solution.
```

These inequalities are also known as *Benders Cuts*. Since they are aggregations of the cuts from the outer approximation procedure, the resulting dual bounds are weaker. We give a detailed scheme of the algorithm in Algorithm 9 that is identical to outer approximation, except of exchanging the problems MILP-OA$_k$ by MILP-GBD$_k$. Analogously we have the following optimality and convergence theorem.

Theorem 3.7 ([92]). *Assuming A1-A3 hold, then $z_{CMINLP} = z_{GBD}$ and the algorithm terminates in a finite number of steps at the optimal solution of CMINLP or with an indication that it is infeasible.*

3.2.4 Extended Cutting Planes

In 1960, Kelley [122] devised his cutting plane algorithm for solving convex NLPs. 35 years later, Westerlund and Pettersson [188] presented an extension of this algorithm for CMINLP. The difference to outer approximation and Generalized Benders Decomposition is that it iteratively solves a mixed-integer linear programming problem that is tightened by cuts without solving any NLP-relaxation. The cuts added at each iteration are chosen as the linearizations of the most violated constraint $g_j,\ j = 1, \ldots, m$ and the objective function about the current iterate solution point x^k. The solutions generated by MILP-ECP$_k$ are therefore non-decreasing. In every iteration the following MILP is solved:

$$
\begin{aligned}
z_{ECP_k} := \min\ & \eta \\
\text{s.t. } & f(x^k,y^k) + \nabla f(x^k,y^k)^\top \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \le \eta,\ (x^k,y^k) \in K \\
& g_j(x^k,y^k) + \nabla g_j(x^k,y^k)^\top \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \le 0,\ (x^k,y^k) \in K,\ j = J(x^k,y^k) \\
& x \in X \cap \mathbb{Z}^n,\ y \in Y,\ \eta \in \mathbb{R},\ (x^k,y^k) \in K,
\end{aligned}
\tag{MILP-ECP$_k$}
$$

where $J(x^k,y^k) := \operatorname{argmax}_{j=1,\dots,m} g_j(x^k,y^k)$ for every solution $(x^k,y^k) \in K$, the set of solutions to (MILP-ECP$_k$).

Algorithm 10: Extended Cutting Plane Algorithm

```
input  : $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}$, $g : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ convex and differentiable
output: $\min\{f(x,y) \mid g(x,y) \le 0,\ (x,y) \in X \cap \mathbb{Z}^n \times Y\}$
Choose $\varepsilon > 0$ and $(x^0,y^0) \in X \cap \mathbb{Z}^n$. Set $K = \emptyset$ and $k = 0$.
repeat
    Solve MILP-ECP$_k$. Let $(x^k,y^k,\eta^k)$ denote its optimal solution.
    Update
        $K = K \cup \{(x^k,y^k)\},\ t = \operatorname{argmax}_j\ g_j(x^k),\ J(x^k,y^k) = \{t\}$
    Set $k = k + 1$.
until $(g_j(x^{k-1},y^{k-1}) \le \varepsilon$ for all $j = 1,\dots,m)$ or $(f(x^{k-1},y^{k-1}) - \eta^{k-1} \le \varepsilon)$
STOP: $(x^k,y^k)$ is optimal.
```

Note that in case we want to solve a MILP, the ECP-method converges in one iteration since the ECP MILP-model is exactly the original problem. Nevertheless the convergence might be very slow if the nonlinearity of the underlying problem is very high. Variants of the ECP-method where more than one linearization is added in each iteration are also possible. The ECP-method is sketched in Algorithm 10.

Theorem 3.8 ([188]). *Assuming A1 and A2 hold, then z_{ECP_k} converges to z_{CMINLP}.*

3.2.5 LP/NLP-based Branch-and-Bound

LP/NLP-based branch-and-bound is a concept developed by Quesada and Grossmann [169] and extended for CMINLP by Leyffer in his PhD Thesis [137]. Its key idea is to combine branch-and-bound with outer approximation and can be considered as a branch-and-cut procedure. It starts to solve the initial MILP-OA$_k$ of outer approximation by branch-and-bound using its continuous relaxation. Each time an integer solution x^k is found, it solves NLP(x^k) and successively adds linearizations of objective functions and constraint functions about (x^k, y^k) to MILP-OA$_k$, where y^k is the corresponding optimal solution of NLP(x^k). The branch-and-bound tree is continued with the updated problem MILP-OA$_k$, i.e., the cuts are considered for every upcoming node. The main feature of LP/NLP-based branch-and-bound is the possibility to keep track of a single enumeration tree. This leads to a reduced number of nodes to be enumerated but nevertheless increases the number of NLP subproblems to be solved. Convergence properties are the same as for outer approximation. We give an outline in Algorithm 11.

Algorithm 11: LP/NLP-based branch-and-bound

```
input  : $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}$, $g : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ convex and differentiable
output: $\min\{f(x,y) \mid g(x,y) \leq 0,\ (x,y) \in X \cap \mathbb{Z}^n \times Y\}$
Set $U = \infty,\ k = 0$. Choose $x^k \in \mathbb{Z}^n$.
Solve NLP($x^k$) and build MILP-OA$_k$.
Solve MILP-OA$_k$ by branch-and-bound:
    if The solution $(\bar{x}, \bar{y})$ of any node is integer feasible then
        STOP branching, solve NLP($x^k$) (or NLPF($x^k$) with $x^k = \bar{x}$ to
        modify MILP-OA$_k$, set $k = k + 1$,.
        Continue branch-and-bound with the new MILP-OA$_k$.
```

3.2.6 Hybrid Methods

Bonami et al. [34] developed a hybrid method that combines branch-and-bound with outer approximation. In a first phase, for a given time limit τ, the classical OA algorithm is executed. At the end of this phase, we are given the current MILP-OA$_k$ and an incumbent primal bound U^k. Starting from that, a modified version of the LP/NLP-based branch-and-bound algorithm of Quesada and Grossmann is performed. The difference is that at every fixed number κ of nodes the nonlinear programming problem NLP(x^k) is solved and independent of the integrality of x^k the linearizations about (x^k, y^k) are added to MILP-OA$_k$ to strengthen its formulation, unlike in Algorithm 11, where NLP(x^k) is only solved when x^k is integral. Choosing $\tau = \infty$ gives the classical outer approximation, while choosing $\tau = 0$ and $\kappa = 1$ gives a classical branch-and-cut algorithm.

3.2.7 Primal Heuristics

In the context of CMINLP there are two categories of primal heuristics, originally designed for MILP and extended to convex MINLP, that have been studied. The first one is *Diving Heuristics* [33] where successively integer variables are fixed from the root node to a leaf of the tree to quickly get primal bounds. Diving follows the idea of a depth-first search strategy. The second one is *Feasibility Pumps* [35] where in alternation two sequences of iterates are generated: one consists of iterates $\bar{x}^i$ which satisfy all constraints except of integrality and the other consists of iterates $\hat{x}^i$ which are integral but do not necessarily satisfy the remaining constraints. The two sequences are generated in such a way that the distance on the integer variables $\sum_{j\in I} |\bar{x}^i_j - \hat{x}^{i+1}_j|$ between two consecutive iterates $\bar{x}^i$ and $\hat{x}^{i+1}$ is non-increasing. Here I denotes the index set of all integer variables. The process is iterated until some integer feasible solution is found. A good survey on heuristics for convex mixed-integer nonlinear programming problems is given by Bonami and Gonçalves [33].

3.3 Software for Convex Mixed-Integer Nonlinear Programming

In this section, we present the most used general-purpose software packages for solving CMINLP to optimality. All of them are mainly based on one or more of the algorithmic frameworks presented in Section 3.2. For each solver, we give a short explanation of the used algorithm(s) and, if important, some additional information. In Table 3.1, we summarize all the information, including five state-of-the-art software packages for convex mixed-integer quadratically constrained programming problems (MIQCP). Note that in nearly every presented solver a bunch of sophisticated enhancements are used, like primal heuristics, domain propagation, presolving techniques, enhanced branching strategies and node selection strategies (see Section 3.1.1) or cutting plane separation (see Section 3.1.2). In our overview, we are not considering more general software, e.g., the well-known solvers α-`BB`, `LaGO`, `LINDO-Global`, `ANTIGONE` and `GloMIQO` designed to solve general non-convex mixed-integer problems. We refer to the surveys of Burer and Letchford [45], Bussieck and Vigerske [46] and D'Ambrosio and Lodi [60]. We just mention `Baron` and `Couenne` that are most popular among the software for general non-convex MINLP.

3.3.1 Solvers

α-ECP (α-Extended Cutting Plane) [187] is based on the ECP Method (see Section 3.2.4) by Westerlund and Petersson [188] at the Åbo Akademi University

Table 3.1: Most used software packages for solving CMINLP.

solver	algorithm(s) implemented	open-source	author(s)/companies	dependencies
α-ECP	ECP [188]	no	T. Westerlund and K. Lundqvist	GAMS, CPLEX
	website: `users.abo.fi/twesterl/A-ECPManual.pdf`			
Baron	branch-and-reduce [171, 182]	no	N. Sahinidis and M. Tawarmalani	CPLEX, Minos/Snopt
	website: `http://archimedes.cheme.cmu.edu/?q=baron`			
Bonmin	B&B [105], OA [75, 80], LP/NLP-based B&B [169], Hybrid Methods [34]	yes	P. Bonami et al.	Cbc/CPLEX, Ipopt/filterSQP
	website: `https://projects.coin-or.org/Bonmin`			
DICOPT	OA [75, 80]	no	J.Viswanathan and I.E. Grossmann	GAMS, any MILP/NLP solver under GAMS
	website: `www.gams.com/dd/docs/solvers/dicopt.pdf`			
FilMINT	LP/NLP-based B&B [169]	no	K. Abishek, S. Leyffer and J. Linderoth	MINTO, filterSQP
	website: `www.mcs.anl.gov/~leyffer/papers/fm.pdf`			
KNITRO	LP/NLP-based B&B [169]	no	Ziena Optimization LLC	-
	website: `http://www.ziena.com/knitro.htm`			
MINLPBB	B&B [105]	no	S. Leyffer	filterSQP, bqpd
	website: `www-unix.mcs.anl.gov/~leyffer/solvers.html`			
MINOPT	GBD [92], OA [75, 80]	no	C.A. Schweiger C.A. Floudas	CPLEX/LPsolve, Minos/NPsol/ Snopt
	website: `http://titan.princeton.edu/MINOPT/`			
MINOTAUR	NLP-based B&B [105], LP/NLP-based B&B [169], branch-and-reduce [171],	yes	S. Leyffer, J. Linderoth J. Luedtke, A. Mahajan T. Munson	filterSQP/IPOPT
	website: `http://wiki.mcs.anl.gov/minotaur/index.php/MINOTAUR`			
SBB	B&B [105]	no	ARKI Consulting and Development	GAMS, CONOPT/ Minos/Snopt under GAMS
	website: `www.gams.com/solvers/solvers.htm#SBB`			
CPLEX	B&B	no	IBM	-
	website: `http://www.cplex.com/`			
GUROBI	B&B	no	Gurobi Optimization, Inc.	-
	website: `http://www.gurobi.com/`			
FICO Xpress	B&B	no	Fair Isaac Corporation	-
	website: `http://www.fico.com/xpress`			
MOSEK	B&B	no	MOSEK ApS	-
	website: `http://www.mosek.com/`			
SCIP	Spatial B&B	yes	Zuse Institut Berlin (ZIB)	SoPlex/CPLEX, CppAD/IPOPT
	website: `http://scip.zib.de`			

in Finland and was extended by Westerlund and Pörn [167, 189] for pseudo-convex objective functions and constraints. At every iteration the underlying Problem MILP-ECP$_k$ is solved by calling `CPLEX` [57], an optimization software suite maintained by IBM, or any other MILP solver. The resulting cuts are added to the MILP formulation until the algorithm converges. A major difference to all other software is that α-`ECP` does not require any NLP solver.

Baron (Branch-And-Reduce Optimization Navigator) [172, 183] is a computational system for solving general non-convex optimization problems to global optimality. Purely continuous, purely integer, and also mixed-integer nonlinear problems can be solved with the software. `Baron` implements algorithms of the branch-and-bound type enhanced with a variety of constraint propagation and duality techniques for reducing ranges of variables in the course of the algorithm. Furthermore, the implementation includes heuristics for the approximate solution of optimization problems to tighten the variable bounds. Originally the software was created by Nick Sahinidis during his time as Assistant and Associate Professor at the University of Illinois at Urbana-Champaign. Currently it is developed by Sahinidis at Carnegie Mellon University and M. Tawarmalani at Purdue University.

Bonmin (Basic Open-source Nonlinear Mixed INteger programming) [34] consists of four solvers: `B-BB` is an NLP-based branch-and-bound algorithm (see Section 3.2.1), `B-OA` uses the outer approximation algorithm (see Section 3.2.2), `B-QG` is an implementation of the LP/NLP-based branch-and-bound algorithm by Quesada and Grossmann (see Section 3.2.5) and `B-Hyb` is a hybrid version of `B-BB` and `B-OA` (see Section 3.2.6). The standard solvers for solving NLP and MILP problems are `Ipopt` (Interior Point OPTimizer) [186] and `Cbc` (COIN-OR branch-and-cut) but other solvers like `filterSQP` or `CPLEX` may also be chosen by the user. `Ipopt` is an interior point solver for large-scale NLP problems, written by A. Wächter and C. Laird and `Cbc` is a branch-and-cut framework for MILP problems, originally developed by J. Forrest. Both solvers are open-source and are part of the COIN-OR (Common Optimization Interface for OR) [147] initiative. `Bonmin` was written and is maintained by P. Bonami et al. in cooperation with researchers from Carnegie Mellon University and IBM Research.

DICOPT (Discrete Continuous OPTimizer) was implemented by I.E. Grossmann and J. Viswanathan [126, 127] from the Carnegie Mellon University and is based on an extension of the outer approximation algorithm and is used within the `GAMS` (General Algebraic Modeling System) [87] interface. `GAMS` is a commercial modeling system for mathematical programming and optimization including a huge variety of solvers by D. Kendrick and A. Meerhaus. For solving the occurring series of NLP and MILP problems the user may therefore choose between all available solvers under GAMS, e.g., CONOPT for NLP problems and CPLEX for MILP problems. A complete list can be found at [87]. A major restriction of DICOPT is that it can handle only integer variables that appear linearly in the constraints and objective function.

FilMINT (Filter-Mixed Integer Optimizer) [1] is a solver written by K. Abishek, S. Leyffer and J. Linderoth from the Argonne National Laboratory and Lehigh University. It is another implementation of the LP/NLP-based branch-and-bound algorithm and uses the solvers filterSQP [82] and MINTO (Mixed INTeger Optimizer) [160]. MINTO was developed by M. Savelsbergh and G. Nemhauser and is an LP-based branch-and-bound software for solving MILP problems and filterSQP is a sequential quadratic programming trust region solver by R. Fletcher and S. Leyffer for solving NLP problems. filterSQP itself uses the QP solver named bqpd that is an active set solver and was developed by the same authors.

KNITRO (Nonlinear Interior point Trust Region Optimization) [47] is a commercial software package for solving large scale mathematical optimization problems by R. Waltz, J. Nocedal, T. Plantenga and R. Byrd, first introduced in 2001 and now sold through Ziena Optimization LLC. KNITRO is specialized for nonlinear optimization, but can also handle mixed-integer nonlinear programming problems, both convex and non-convex. KNITRO implements a branch-and-bound algorithm, offering three different optimization algorithms for solving the NLP-relaxation. Two algorithms are of the interior point type, and one is of the active set type. Special features of KNITRO are the possibility of crossovers during the solution process from one algorithm to another and a multistart option.

MINLPBB (Mixed-Integer Nonlinear Programming Branch-and-Bound) [138] implements an NLP-based branch-and-bound algorithm and uses, like FilMINT, the NLP solver filterSQP. It was also developed by R. Fletcher and S. Leyffer at the University of Dundee.

MINOPT (A Modeling Language and Algorithmic Framework for Linear, Mixed-Integer, Nonlinear, Dynamic, and Mixed-Integer Nonlinear Optimization) is a software package by C.A. Schweiger [176] and C.A. Floudas from Princeton University and can handle many general optimization problems besides convex MINLP, including problems with differential and algebraic constraints or optimal control problems. Some algorithms implemented are outer approximation in many variants and the Generalized Benders Decomposition (see Section 3.2.3). MINOPT also includes a modeling language.

MINOTAUR (Mixed-Integer Nonconvex Optimization Toolbox - Algorithms, Underestimators, Relaxations) [148, 149] is developed by S. Leyffer, J. Linderoth, J. Luedtke, A. Mahajan and T. Munson at Argonne National Laboratory and the University of Wisconsin-Madison. It is open-source and can be used through `AMPL` and implements an NLP-based branch-and-bound algorithm, where the NLP relaxations are solved by `IPOPT` or `filterSQP`. Additionally, it offers an option to replace the NLP relaxations by faster-to-solve QP approximations. `MINOTAUR` also features a technique to check if the feasible region of certain nonlinear functions can be represented as a union of few convex regions, aiming at branching in a specific way to create convex subproblems of these nonlinear problems.

SBB (Simple Branch-and-Bound) [88] is a commercial software package by ARKI Consulting & Development A/S, running under `GAMS`. It uses standard NLP-based branch-and-bound and may therefore utilize several NLP solvers available under `GAMS`, e.g., the large-scale nonlinear optimization solver `CONOPT` [72] and `MINOS` [158] by B. Murtagh and M. Saunders or `SNOPT` (Sparse Nonlinear OPTimizer) [93] by P. Gill, W. Murray and M. Saunders.

The following solvers are capable of solving linear programming (LP), quadratic programming (QP), quadratically constrained programming (QCP), mixed integer linear programming (MILP) and mixed-integer (quadratically constrained) programming (MIQ(C)P) problems. If the MIQ(C)P problem has only binary variables in the indefinite quadratic matrices, the problem is usually reformulated to an equivalent convex MIQ(C)P problem by most of the solvers. All MIQ(C)P solvers implement a branch-and-bound algorithm that uses Q(C)P relaxations for computing the bounds.

CPLEX (IBM ILOG CPLEX Optimization Studio) [57] is a commercial optimization software package by IBM. It was named for the simplex method as implemented in the C programming language. It was originally developed by R. Bixby in 1988 for CPLEX Optimization Inc., which was acquired by ILOG in 1997; ILOG was subsequently acquired by IBM in 2009. Very recently, a nonconvex MIQP solver was added, which uses a *spatial branch-and-bound* algorithm. In a generic spatial branch-and-bound algorithm a lower and upper bound to the optimal solution value is computed for a subspace of the feasible region. In case the gap between those bounds are not sufficiently close, the subspace is further split and the process is repeated until all nodes can be pruned.

GUROBI (Gurobi Optimizer) [106] is a commercial optimization software package named for its founders: Z. Gu, E. Rothberg and R. Bixby. The former two were also involved in the CPLEX-development team. Together with `CPLEX` it is considered the most effective MIQ(C)P solver available nowadays.

FICO Xpress (FICO Xpress Optimization Suite) [78] was developed by Dash Optimization, later acquired by FICO. Besides `CPLEX` and `GUROBI`, it is also considered as one of the state-of-the-art solver for MIQ(C)P problems.

MOSEK (MOSEK optimization software) [156] was developed by MOSEK ApS. In particular, `MOSEK` is known for its effective interior point SOCP solver.

SCIP (Solving Constraint Integer Programs) [177] was developed by the optimization department at the Zuse Institute Berlin (ZIB), in particular by T. Achterberg [2], and its collaborators. Originally it was developed as a MILP solver, but it was constantly extended to handle also general non-convex MINLP problems [22, 23]. In contrast to the other MIQCP solvers `SCIP` is open-source. It implements a spatial branch-and-bound algorithm including sophisticated primal heuristics and preprocessing techniques like bound-tightening. Currently, `SCIP` is often considered as one of the fastest non-commercial solvers for mixed-integer nonlinear programming (MINLP) problems.

3.3.2 Performance Benchmarks and Collection of Test Instances

Recent benchmark tests of all these solvers for convex mixed-integer nonlinear programming problems are regularly provided by Hans Mittelmann from Arizona State University. Detailed results on the performance of different state-of-the-art solvers on a set of standard benchmark instances can be found on the website: `http://plato.asu.edu/bench.html`. In summary, the most recent results conclude that the outer approximation algorithm of Bonmin, `B-OA`, combined with `CPLEX` as the required LP solver has the best overall performance in terms of the average running times run on a test bed of 155 convex MINLP instances, see `http://plato.asu.edu/ftp/minlp.html`. For the 25 MIQ(C)P instances it is shown that `GUROBI` and `CPLEX` are by far the most effective solvers currently available, see `http://plato.asu.edu/ftp/miqp.html`. Justified by these results, we will choose `Bonmin` and `CPLEX` as competitors in the following Chapters 5 to 7. The results are taken from the links above and are summarized in the Tables 3.2 and 3.3.

Table 3.2: Benchmark Results of Hans Mittelmann for MIQP problems: Scaled shifted geometric means of running times in cpu-seconds.

Mixed-Integer Quadratic Programming Solver							
Bonmin	Couenne	CPLEX	GUROBI	SCIP	XPRESS	MINOTAUR	CBC
89	325	1.28	1	20.6	2.65	42.2	31.4

Table 3.3: Benchmark Results of Hans Mittelmann for convex MINLP problems: Scaled shifted geometric means of runtimes cpu-seconds.

Convex Mixed-Integer Nonlinear Programming Solver							
KNITRO	SCIP	α-ECP	B-OA	B-Hyb	DICOPT	MINOPT	SBB
12.4	1.59	4.11	1	3.36	2.7	3.66	14.4

To provide algorithm developers with a large set of both theoretical and practical test models, researchers started to gather benchmark instances. Recently the most popular public available libraries for MINLP are:

- **MacMINLP Library:** A collection of MINLP test problems is available at `http://wiki.mcs.anl.gov/leyffer/index.php/MacMINLP`. The library is maintained by Sven Leyffer from the Argonne National Laboratory and consists of 51 instances, from which 15 are convex. The website also provides additional information like the number of integer variables, the optimal objective value or best solution found.

- **CMU/IBM Library:** This collection of convex MINLP instances was created within a cooperation between researchers of Carnegie Mellon University and IBM. It contains 41 instances. The collection is publicly available at `http://egon.cheme.cmu.edu/ibm/page.htm`.

- **MINLPLib(2) Library:** Michael R. Bussieck from GAMS Development Corporation introduced this collection of 266 instances in GAMS format, all available at `http://www.gamsworld.org/minlp/minlplib.htm`. They include additional information like optimal solution value and origin of the instance. Very recently, the library started to get enlarged, now also containing NLP instances. The current version can be accessed at `http://www.gamsworld.org/minlp/minlplib2/html/`.

- **QPLIB2014:** This very recent library, still under construction, is specifically conceived for (mixed-integer) quadratic programming problems. It is intended to be a standard test set for the computational comparison of different generic quadratic optimization softwares. Currently no instances are available yet. However, the authors announced the release of more than 8000 instances at `http://www.lamsade.dauphine.fr/QPlib2014/doku.php` in the near future. The library is maintained by many researchers all over the world.

Chapter 4

Convex Quadratic Mixed-Integer Programming

In this chapter, we consider strictly *Convex Quadratic Mixed-Integer Programming* (CQIP) problems with box-constraints of the form

$$\min\{f(x) = x^\top Qx + c^\top x + d \mid x \in [l, u]^n,\ x_1, \dots, x_{n_1} \in \mathbb{Z}\}, \qquad \text{(CBMIQP)}$$

where $Q \in \mathbb{R}^{n\times n}$ is symmetric and positive definite, $c \in \mathbb{R}^n$, $l_i \in \mathbb{R} \cup \{-\infty\}$, $u_i \in \mathbb{R}\cup\{\infty\}$ for all $i = 1, \dots, n$, $d \in \mathbb{R}$ and $n_1 \in \{0, \dots, n\}$. This problem can be classified as a convex mixed-integer programming problem CMINLP, discussed in Chapter 3. From a methodological point of view it is natural to study CBMIQP as a first step towards generalizing MILP-methods for CMINLP. Moving from mixed-integer linear programming to mixed-integer convex quadratic programming does not change the overall complexity in theory, however solving these two types of problems in practice makes a huge difference. Similarly, this is also true when comparing the effort of solving CBMIQP and general convex nonlinear mixed-integer problems. Being aware of this, it is reasonable to take advantage of the given problem structure. Since the convex quadratic objective function and the box-constraints are much more specific than general convex mixed-integer programs it is not useful to apply the general methods presented in Section 3.3 directly to our problem. Instead, it is reasonable to develop an algorithm that fully exploits the problem structure and adjusts the existing methods to CBMIQP.

This convex quadratic mixed-integer problem has several applications. One practical application in electronics, known as *Filtered Approximation* [48], is to synthesize periodic waveforms by either bipolar or tripolar pulse codes. The problem can be modeled in form of CBMIQP, where all variables are required to be binary or ternary, i.e., $x \in \{-1, 0, 1\}^n$. Another more theoretical application is the so-called *Closest Vector Problem* (CVP) [184]: given a basis $b^1, \dots, b^n$ of $\mathbb{R}^n$ and an additional vector $v \in \mathbb{R}^n$, it asks for an integer linear combination of the basis vectors which is as close as possible to v with respect to the Euclidean distance.

It can be shown that this problem is equivalent to the unbounded version of CB-MIQP where $n_1 = 0$, i.e., additionally all variables are required to be integer. In the literature CVP is sometimes also known as the *Integer Least Squares Problem* (ILS). It has several applications in the areas of wireless communication, cryptography, Monte-Carlo second moment estimation, radar imaging and global navigation satellite systems, see e.g., [7, 107, 157]. The general box-constrained version is therefore called *Box-constrained Integer Least Squares Problem.* A common approach for solving ILS consists of a two-stage enumeration strategy. First, a reduction phase is performed, where Q is transformed to an upper triangular matrix using an orthogonal transformation, followed by a search phase. Typically they are based on either the Schnorr-Euchner algorithm [174] or the Fincke-Pohst algorithm [79]. A survey on search methods for ILS is given by [7]. Similarly, BILS is usually solved by extending one of the search algorithms to so-called sphere decoding methods, where a tree search is performed only on the lattice points within a hypersphere of a chosen radius around v, see, e.g., [40, 52, 61]. In order to speed up the search algorithm the columns are reordered in a preprocessing phase.

This chapter is organized as follows. After giving a short proof that the problem CBMIQP is NP-hard, we give an outline and the main ideas of a branch-and-bound scheme in Section 4.2. After this, in Section 4.3, we explain the major running time improvements, leading to a linear running time per node algorithm.

4.1 Complexity of CQIP

Solving CBMIQP is NP-hard. There are two straightforward ways to see this. The first one follows from the special case of CBMIQP where all the variables are required to be binary, i.e., $x_i \in \{0, 1\}$ for all $i = 1, \ldots, n$. In this case we have a close connection between CBMIQP and two classical and well-studied combinatorial optimization problems, namely Unconstrained Binary Quadratic Programming (UBQP) and Max-Cut (MC). To be precise, UBQP is a special case of CBMIQP since convexity of the objective function can be assumed without loss of generality. Both UBQP and MC are NP-hard [121, 165] and it is also proven that UBQP and MC have the same polyhedral description [181], so that the underlying *Boolean Quadric Polytope* and the *Cut Polytope* can be mapped to each other under a linear transformation. In fact, MC can be modeled as an unconstrained quadratic minimization problem over the set of $\{-1, 1\}$ variables. Therefore the algorithmic approaches for UBQP and MC may also be applied to CBMIQP in the case of binary variables. Here, we alternatively derive NP-hardness of CBMIQP by considering the special case of CBMIQP where all variables are integer and unbounded, i.e., $x_i \in \mathbb{Z}$ for all $i = 1, \ldots, n$. We show that this problem is equivalent to the already mentioned *Closest Vector Problem* which is also NP-hard itself [184].

Theorem 4.1. *The problem CBMIQP is NP-hard.*

Proof. We consider the following decision variant of CVP:

given: a basis $b^1, \dots, b^n$ of $\mathbb{R}^n$, an additional vector $v \in \mathbb{R}^n$ and a scalar $d \in \mathbb{R}$.
question: does there exist an integer linear combination of $b_1, \dots, b_n$ such that its Euclidean distance to the vector v is less or equal d?

We show the reduction CVP $\leq_T$ CQIP:
Given an instance of CVP, we are looking for integer scalars $\lambda_1, \dots, \lambda_n \in \mathbb{Z}$, such that $|| \sum_{i=1}^n \lambda_i b^i - v||^2 \leq d$. We choose $\tilde{Q} := B^\top B$, $\tilde{c} := B^\top v$ and $\tilde{d} := v^\top v$, where $B \in \mathbb{R}^{n \times n}$ consists of the columns $b^1, \dots, b^n$, and get an instance for CBMIQP. We have that

$$\begin{aligned}
&\exists \lambda \in \mathbb{Z}^n : \lambda^\top \tilde{Q} \lambda + \tilde{c} \lambda + \tilde{d} \leq d \\
\Longleftrightarrow &\exists \lambda \in \mathbb{Z}^n : \lambda^\top B^\top B \lambda - 2 v^\top B \lambda + v^\top v \leq d \\
\Longleftrightarrow &\exists \lambda \in \mathbb{Z}^n : ||B\lambda - v||^2 \leq d \\
\Longleftrightarrow &\exists \lambda \in \mathbb{Z}^n : || \sum_{i=1}^n \lambda_i b^i - v||^2 \leq d.
\end{aligned}$$

As we have shown that the unconstrained version of CBMIQP is NP-hard, the result follows immediately since this is the special case of CBMIQP where the bounds are $l = \{-\infty\}^n$ and $u = \{\infty\}^n$. □

4.2 A Branch-and-Bound Algorithm for CQIP

The following branch-and-bound algorithm for solving CBMIQP was presented by Buchheim, Caprara and Lodi [43]. It is a tailored version of an NLP-based branch-and-bound algorithm, as described in Section 3.2.1, and uses the following ingredients, mostly already explained in Section 3.1.1:

- simple primal bound computation using a depth-first enumeration strategy
- easy dual bound computation by using the continuous relaxation
- possible bound improvement by lattice-free ellipsoids
- fast incremental computation of the dual bounds
- branching by fixing variables to integer values
- sophisticated preprocessing phase by using a predetermined branching order

For simplicity, we assume without loss of generality, that the variables are sorted in a way such that the integer variables are $x_1, \ldots, x_{n_1}$ and the continuous variables are $x_{n_1+1}, \ldots, x_n$.

Branching At every node in our branch-and-bound scheme, we branch by fixing a single variable in increasing distance to its value in the solution of the continuous relaxation $\bar{x}$. The branching strategy works as follows. In a generic node of the search tree, we assume that the next variable to be fixed is x_i. The subsequent values to which we fix x_i are determined by the value $\bar{x}_i$ in the continuous minimizer of the current node. For example, if the closest integer value to $\bar{x}_i$ is $\lfloor \bar{x}_i \rfloor$, we fix x_i to integer values $\lfloor \bar{x}_i \rfloor$, $\lceil \bar{x}_i \rceil$, $\lfloor \bar{x}_i \rfloor - 1$, $\lceil \bar{x}_i \rceil + 1$, and so on. After each branching step, the resulting subproblem is a mixed-integer quadratic programming problem of type CBMIQP again, with a problem dimension decreased by one. We are not explicitly imposing bound constraints on the integer variables (i.e., $x_i \leq \lfloor \bar{x}_i \rfloor$ and $x_i \geq \lfloor \bar{x}_i \rfloor$), since they are implicitly taken into account as fixings (i.e., $x_i = \lfloor \bar{x}_i \rfloor$) in the construction of the reduced subproblem by adapting properly the matrices Q_ℓ, the linear term $\bar{c}$ and the constant term $\bar{d}$ (see Section 4.3). The branching order of these variables at every level $0 \leq \ell \leq n_1$ is set to $x_1, \ldots, x_{n_1-\ell}$, assuming that ℓ variables are already fixed. Hence, at every level we have a predetermined branching order. Let $\bar{x}_i$ be the value of the next branching variable in the continuous minimizer. Then, by the convexity of f, all consecutive lower bounds obtained by fixing x_i to integer values in increasing distance to $\bar{x}_i$, on each side of $\bar{x}_i$, are non-decreasing. Thus, we can cut off the current node of the tree and all its outer siblings as soon as we fix a variable to some value for which the resulting lower bound exceeds the current best known upper bound. Since f is strictly convex we get a finite algorithm even in the case where the variables are unbounded.

Dual Bounds We know from Theorem 2.16 that there exists a unique minimum $\bar{x}$ for the continuous relaxation of the unbounded version of CBMIQP, since Q is positive definite. At every depth ℓ we have

$$2Q_\ell \bar{x} + \bar{c} = 0 \Leftrightarrow \bar{x} = -\frac{1}{2} Q_\ell^{-1} \bar{c} \tag{4.1}$$

and

$$\begin{aligned} f(\bar{x}) &= -\frac{1}{2} \bar{c}^\top Q_\ell^{-\top} Q_\ell (-\frac{1}{2} Q_\ell^{-\top} \bar{c}) + \bar{c}^\top (-\frac{1}{2} Q_\ell^{-1} \bar{c}) \\ &= \frac{1}{4} Q_\ell^{-1} \bar{c} - \frac{1}{2} \bar{c}^\top Q_\ell^{-1} \bar{c} + \bar{d} \\ &= \bar{d} - \frac{1}{4} \bar{c}^\top Q_\ell^{-1} \bar{c}. \end{aligned}$$

Thus, the continuous minimum can be used as a dual bound for CBMIQP. Its direct computation simply asks for solving a system of linear equations which takes cubic running time in the reduced dimension $n-\ell$. We remark that the box-constraints are taken into account implicitly by the branch-and-bound scheme. We prune all nodes with invalid variable fixings outside of the box, instead of computing the box-constrained continuous minimizer, since the latter problem is much harder to solve in practice. Only at depth n_1, after all integer variables have been fixed, we need to compute a feasibles point within the box. This can be done by any common method for box-constrained QP, for example active set or interior point methods.

Primal Bounds Since the problem is only box-constrained, any mixed-integer point $x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n-n_1} \cap [l,u]^n$ is feasible and therefore at level n_1 of the enumeration tree, meaning all integer variables have been fixed, we get a primal bound. Alternatively, one can try to improve the primal bound during the enumeration by using primal heuristics. The authors use a genetic algorithm for 0-1 programming to improve primal bounds [146].

4.3 Improvement to Linear Running Time per Node

Preprocessing Since the dual bounds obtained by the continuous relaxation may be weak, the enumeration tree might become very large. Therefore the strategy of the branch-and-bound scheme is to enumerate the nodes quickly. Before starting the enumeration it takes advantage of a preprocessing phase that performs the most time-consuming computations in every node in advance. This is possible due to the fact that at every level ℓ of the branching tree, certain data that is needed to compute the continuous minimizer does not depend on the specific fixings but only on the current depth. To explain this, we first describe a useful effect of their way to branch by fixing. After each branching a new subproblem of the same form in a reduced dimension is built by updating the problem data. In this way, no constraint needs to be added for the new fixing. In particular having fixed ℓ variables $x_1, \dots x_\ell$ to integer values $r_1, \dots, r_\ell$, we get the reduced objective function $f : \mathbb{R}^{n-\ell} \to \mathbb{R}$ of the form

$$\bar{f}(x) = x^\top Q_\ell x + \bar{c}^\top x + \bar{d},$$

where the reduced constant term $\bar{d} = \sum_{i=1}^{\ell} c_i r_i + \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} q_{ij} r_i r_j$ contains the fixings from the quadratic and linear part and the components of the reduced linear part $\bar{c}_{j-d} = c_j + 2\sum_{i=1}^{\ell} q_{ij} r_i$, $j = \ell+1, \dots, n$, contain the fixings from the quadratic part. The reduced matrices Q_ℓ are obtained from Q by deleting the first

Algorithm 12: Basic Branch-and-Bound Scheme for CBMIQP

```
input  : a strictly convex function $f : \mathbb{R}^n \to \mathbb{R}, f(x) = x^\top Qx + c^\top x + d$
output: a vector $x^\star \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n-n_1}$ minimizing $f$

determine a variable order $x_1, \ldots, x_{n_1}$, set $\ell := 0$, $ub := \infty$
let $Q_\ell$ be the submatrix of $Q$ for rows and columns $\ell + 1, \ldots, n$
while $0 \leq \ell \leq n_1$ do
    define $\bar{f} : \mathbb{R}^{n-\ell} \to \mathbb{R}$ by $\bar{f}(x) := f(r_1, \ldots, r_\ell, x_1, \ldots, x_{n-\ell})$
    compute $\bar{c}$ and $\bar{d}$ such that $\bar{f}(x) = x^\top Q_\ell x + \bar{c}^\top x + \bar{d}$
    // compute lower bound
    if $\ell = n_1$ then
        compute $\bar{x} := \operatorname{argmin}\{f(x) \mid a \leq x \leq b\}$
    else
        compute $\bar{x} = -\frac{1}{2} Q_\ell^{-1} \bar{c} \in \mathbb{R}^{n-\ell}$ and set $lb := \bar{f}(\bar{x})$
    // compute upper bound
    set $r_j := \bar{x}_{j-\ell}$ for $j = \ell + 1, \ldots, n_1$ to form $r \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n-n_1}$
    // update solution
    if $\bar{f}(r_{\ell+1}, \ldots, r_n) < ub$ then
        set $r^* = r$
        set $ub = \bar{f}(r_{\ell+1}, \ldots, r_n)$
    // prepare next node
    if $lb < ub$ then
        // branch on variable $x_{\ell+1}$
        set $\ell := \ell + 1$;
        if $l_1 \leq \lfloor \bar{x}_1 \rceil \leq b_1$ then
            set $r_\ell := \lfloor \bar{x}_1 \rceil$;
    else
        // prune current node
        set $\ell := \ell - 1$;
        if $\ell > 0$ then
            // go to next node
            increment $r_\ell$ in increasing distance to the value in the
            continuous relaxation (see paragraph branching above);
```

ℓ rows and columns, and are therefore again positive definite. To compute the dual bounds according to (4.1) we need the inverse matrices Q_ℓ^{-1}. An important observation is that Q_ℓ and therefore also Q_ℓ^{-1} does not depend on the values r_i, $i = 1, \ldots, \ell$, to which the first ℓ variables are fixed. Exploiting this useful fact, we do not change the order of fixing the variables in the branch-and-bound tree at each level ℓ, i.e., we always fix the variables according to an order that is determined a priori, before starting the enumeration. On the one hand, we do not have the freedom to use sophisticated branching rules. But on the other hand, this implies that, in total, we only have to consider the n different matrices Q_ℓ during the whole enumeration tree, which we know in advance, since the fixing order is predetermined. Thus, we avoid to compute an exponential number of reduced matrices that would be needed if the order of variables to be fixed were allowed to be chosen freely. The basic branch-and-bound scheme is outlined in Algorithm 12. Using the preprocessing phase, every node in the enumeration tree can be processed in quadratic running time in the reduced dimension $n-\ell$. Here, the most time-consuming operation is still the computation of the continuous minimum and its corresponding objective function value, given the precomputed inverse matrix Q_ℓ^{-1}. In a first step, using the preprocessing leads to a reduction of the running time per node from $O(n-\ell)^3$ to $O(n-\ell)^2$.

Lemma 4.2 ([43])**.** *After a polynomial-time preprocessing, the running time per node of Algorithm 12 can be reduced to $O(n-\ell)^2$.*

Proof. Since the inverse matrices $Q_\ell^{-1} \in \mathbb{R}^{(n-\ell)\times(n-\ell)}$ are computed in polynomial time in the preprocessing, the running time is dominated by the computation of $\bar{c}^\top Q_\ell^{-1} \bar{c}$ which is quadratic in the reduced dimension $n-\ell$. □

Incremental Computations Apart from precomputing the reduced matrices Q_ℓ, the preprocessing phase can be used to further decrease the running time of every node at level ℓ to $O(n-\ell)$. Again, this is achieved by improving the running time for the computation of the continuous minima of $\bar{f}$. To do this, the authors propose an incremental technique. It determines the new continuous minimum from the old one in linear time whenever a new variable is fixed. For this, the authors make use of the surprising fact that in a given node, the continuous minima corresponding to all possible fixings of the next variable lie on a line and the direction of this line only depends on which variables have been fixed so far, but not on the values to which they were fixed. Again, this allows to shift some expensive computations into the preprocessing since the direction of this line is fully determined by the level of the current node. This observation is illustrated in Figure 4.1. Recall that $\bar{f}(x) = x^\top Q_\ell x + \bar{c}^\top x + \bar{d}$ denotes the function obtained from f by fixing variable x_i to r_i for $i = 1, \ldots, \ell$, and $\bar{x} \in \mathbb{R}^{n-\ell}$ is its continuous minimum, noting that $\bar{x}_1$ corresponds to the value of the original variable $x_{\ell+1}$.

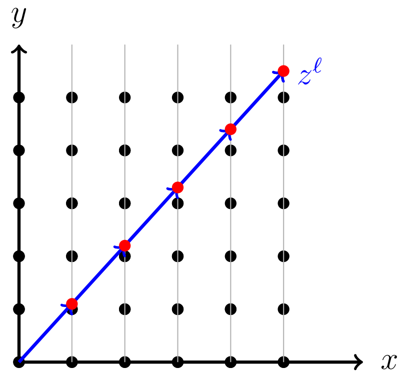

Figure 4.1: Incremental computation of the minimizer y (red dots), depending on the fixings of x by moving along the direction z^ℓ (blue arrow).

If we define the vectors

$$z^\ell := (1, (q_{\ell+1,\ell+2}, q_{\ell+1,\ell+3}, \dots, q_{\ell+1,n})Q_{\ell+1}^{-1})^\top \in \mathbb{R}^{n-\ell},$$

for every $\ell = 1, \dots, n_1$, we finally get the following result.

Proposition 4.3 ([43]). *If we fix variable $x_{\ell+1}$ to $r_{\ell+1}$ and reoptimize $\bar{f}$, the resulting continuous minimum is given by $\bar{x} + (r_{\ell+1} - \bar{x}_1)z^\ell$.*

Proof. Let $\bar{c}(u) \in \mathbb{R}^{n-\ell-1}$ denote the linear term of $\bar{f} : \mathbb{R}^{n-\ell-1} \to \mathbb{R}$ after fixing $x_{\ell+1}$ to u and removing the associated component. We have

$$\bar{c}(u)_{j-\ell-1} = c_j + 2\sum_{i=1}^{\ell} q_{i,j} r_i + 2q_{\ell+1,j}u, \; j = \ell+2, \dots, n.$$

Moreover, we denote by $\bar{\bar{x}} \in \mathbb{R}^{n-\ell}$ the continuous minimizer of $\bar{f}$ after fixing variable $x_{\ell+1}$ to the value $r_{\ell+1}$ and keeping the associated component. This means for the first component we have $\bar{\bar{x}}_1 = r_{\ell+1}$, yielding

$$\bar{\bar{x}} = (r_{\ell+1}, (-\frac{1}{2}\bar{Q}_{\ell+1}^{-1}\bar{c}(r_{\ell+1}))^\top)^\top.$$

It is clear that in case $r_{\ell+1} = \bar{x}_1$, we have $\bar{\bar{x}} = \bar{x}$, since in that case $x_{\ell+1}$ is fixed to its optimal continuous value and the continuous minimum is unchanged. Thus, we can write $\bar{x}$ as

$$\bar{x} = (\bar{x}_1, (-\frac{1}{2}\bar{Q}_{\ell+1}^{-1}\bar{c}(\bar{x}_1))^\top)^\top.$$

If follows that

$$\bar{\bar{x}} = \bar{x} + (r_{\ell+1} - \bar{x}_1, (-\frac{1}{2}\bar{Q}_{\ell+1}^{-1}(\bar{c}(r_{\ell+1}) - \bar{c}(\bar{x}_1)))^\top)^\top.$$

It remains to show that

$$(r_{\ell+1} - \bar{x}_1)z^\ell = (r_{\ell+1} - \bar{x}_1, (-\frac{1}{2}\bar{Q}_{\ell+1}^{-1}(\bar{c}(r_{\ell+1}) - \bar{c}(\bar{x}_1)))^\top)^\top,$$

or equivalently

$$\begin{aligned}&(r_{\ell+1} - \bar{x}_1)(1, -(q_{\ell+1,\ell+2}, q_{\ell+1,\ell+3}, \dots, q_{\ell+1,n})Q_{\ell+1}^{-1})^\top \\ =&(r_{\ell+1} - \bar{x}_1, (-\frac{1}{2}\bar{Q}_{\ell+1}^{-1}(\bar{c}(r_{\ell+1}) - \bar{c}(\bar{x}_1)))^\top)^\top.\end{aligned}$$

The equality for the first component is clear. For the other components, we observe that

$$\bar{c}(r_{\ell+1}) - \bar{c}(\bar{x}_1) = (2q_{\ell+1,\ell+2}(r_{\ell+1} - \bar{x}_1), \dots, 2q_{\ell+1,n}(r_{\ell+1} - \bar{x}_1))^\top \in \mathbb{R}^{n-\ell-1}$$

and hence we get

$$\begin{aligned}&(-\frac{1}{2}\bar{Q}_{\ell+1}^{-1}(\bar{c}(r_{\ell+1}) - \bar{c}(\bar{x}_1))^\top \\ =&(-\bar{Q}_{\ell+1}^{-1}(q_{\ell+1,\ell+2}, \dots, q_{\ell+1,n})^\top(r_{\ell+1} - \bar{x}_1))^\top \\ =&(r_{\ell+1} - \bar{x}_1)(-\bar{Q}_{\ell+1}^{-\top}(q_{\ell+1,\ell+2}, \dots, q_{\ell+1,n})^\top)^\top \\ =&(r_{\ell+1} - \bar{x}_1)(-(q_{\ell+1,\ell+2}, \dots, q_{\ell+1,n})\bar{Q}_{\ell+1}^{-1}).\end{aligned}$$

□

To guarantee sublinear running time of $O(n-\ell)$ per node, we also have to reduce the running time for the evaluation of the objective function values of the continuous minima, since a simple evaluation would take quadratic time and therefore dominate the rest of the computations. As to this, if we define

$$v^\ell := 2Q_\ell z^\ell \in \mathbb{R}^{n-\ell}, \quad s_\ell := (z^\ell)^\top Q_\ell z^\ell \in \mathbb{R},$$

we can state the following analogous result for the incremental computation of the associated optimal objective function value.

Proposition 4.4 ([43]). *If we fix variable $x_{\ell+1}$ to $r_{\ell+1}$ and reoptimize $\bar{f}$, the resulting optimal objective function value is given by*

$$\bar{f}(\bar{x} + \alpha z^\ell) = \bar{f}(\bar{x}) + \alpha(\bar{x}^\top v^\ell + \bar{c}^\top z^\ell) + \alpha^2 s_\ell,$$

where $\alpha = r_{\ell+1} - \bar{x}_1$.

Proof. We have

$$\begin{aligned}f(\bar{x} + \alpha z^\ell) &= (\bar{x} + \alpha z^\ell)^\top \bar{Q}_\ell(\bar{x} + \alpha z^\ell) + \bar{c}^\top(\bar{x} + \alpha z^\ell) + \bar{d} \\ &= \bar{x}^\top \bar{Q}_\ell \bar{x} + \alpha^2 z^{\ell\top} \bar{Q}_\ell z^\ell + 2\alpha \bar{x}^\top \bar{Q}_\ell z^\ell + \bar{c}^\top \bar{x} + \alpha \bar{c}^\top \bar{x} + \alpha \bar{c}^\top z^\ell + \bar{d} \\ &= \bar{f}(\bar{x}) + \alpha(\bar{x}^\top 2\bar{Q}_\ell z^\ell + \bar{c}^\top z^\ell) + \alpha^2 z^{\ell\top} \bar{Q}_\ell z^\ell \\ &= \bar{f}(\bar{x}) + \alpha(\bar{x}^\top v^\ell + \bar{c}^\top z^\ell) + \alpha^2 s_\ell\end{aligned}$$

□

Since $\bar{c}$ can be computed incrementally in $O(n-\ell)$ time, the last two propositions can be summarized in the following main result.

Theorem 4.5 ([43]). *If, in the preprocessing phase of Algorithm 12, we compute z^ℓ, v^ℓ and s_ℓ as defined above for $\ell = 0, \dots, n-1$, then the computation of the continuous minimizer and the associated lower bound can be carried out in $O(n-\ell)$ time per node.*

Proof. The result follows directly from Theorems 4.3 and 4.4. □

We close this chapter with an example.

Example 4.1. Consider the following unconstrained integer quadratic program

$$\min\{x^\top Qx + c^\top x \mid x \in \mathbb{Z}^3\},$$

where

$$Q = \begin{pmatrix} 2 & -1 & -3 \\ -1 & 2 & 4 \\ -3 & 4 & 9 \end{pmatrix} \succcurlyeq 0 \text{ and } c = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}.$$

The continuous minimizer is $\bar{x} = -\frac{1}{2}Q^{-1}c = (1.5, -7, 3.5)^\top$. Suppose we fix x_1 to $r_1 = 2$. For the first iteration we compute in a preprocessing phase the vectors

$$z^0 = (1, -(q_{1,2}, q_{1,3})\bar{Q}_1^{-1})^\top = (1, 1.5, 1)^\top, \text{ where } \bar{Q}_1 = \begin{pmatrix} 2 & 4 \\ 4 & 9 \end{pmatrix},$$

$v^0 = 2Qz^0 = (1, 0, 0)^\top$ and the scalar $s_0 = {z^0}^\top Qz^0 = \frac{1}{2}$.
Hence the new minimizer is

$$\bar{\bar{x}} = \bar{x} + (r_1 - \bar{x}_1)z^0 = (2, -7.75, 4)^\top$$

and its objective function value is

$$\bar{f} = \bar{f}(\bar{x}) + (r_1 - \bar{x}_1)(\bar{x}^\top v^0 + \bar{c}^\top z^0) + (r_1 - \bar{x}_1)^2 s_0 = -6.125.$$

□

Buchheim et al. [43] showed by extensive numerical experiments that for the unconstrained case ($x \in \mathbb{Z}^n$) and the ternary case ($x \in \{-1, 0, 1\}^n$) of CBMIQP the branch-and-bound algorithm outperforms all competitive software on a test set of randomly generated instances as well as real-world instances from an application in electronics. They conclude that `CPLEX` as the most competitive solver was outperformed by even several orders of magnitude.

Chapter 5

Active Set Based Branch-and-Bound

In this chapter we aim at extending the branch-and-bound Algorithm 12 of the previous Chapter 4 for mixed-integer optimization problems with strictly convex quadratic objective functions to the presence of additional linear inequalities:

$$\begin{array}{ll} \min & f(x) = x^\top Q x + c^\top x + d \\ \text{s.t.} & Ax \leq b \\ & x_i \in \mathbb{Z},\ i = 1, \ldots, n_1 \\ & x_i \in \mathbb{R},\ i = n_1 + 1, \ldots, n \end{array} \tag{CMIQP}$$

where $Q \in \mathbb{R}^{n \times n}$ is a positive definite matrix, $c \in \mathbb{R}^n$, $d \in \mathbb{R}$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $n_1 \in \{0, \ldots, n\}$. Therefore, we will present a generalization of the approach for CBMIQP. Up to now, all solution methods for convex mixed-integer quadratic programming are based on the methods presented in Section 3.2. Agrawal [6] presented a cutting plane algorithm that starts from the solution of the QP-relaxation. If the solution has a non-integral component that is required to be integer it adds a Chvátal-Gomory cut to the existing constraints and applies the Parameter t-Algorithm by Beale [18] to find its optimal integral value. The procedure is repeated until integrality for all integer variables is obtained. Bienstock [29] proposed a branch-and-cut algorithm using disjunctive cuts for solving a family of mixed-integer quadratic programming problems arising in a portfolio optimization application. By reformulating the original problem, Lazimy [133, 134] applied the Generalized Benders Decomposition method yielding Benders cuts that are linear instead of quadratic in the integer variables. The original mixed-integer quadratic problem is decomposed into a finite sequence of integer linear master programs, and quadratic programming subproblems. Al-Khayyal and Larsen [8] developed a branch-and-bound algorithm that uses different types of piecewise affine convex functions to underestimate the objective function. The relaxed linear problems are used to compute lower bounds. Instead of using linearizations

Leyffer [137] showed how improved lower bounds from the QP-relaxations can be used within a branch-and-bound algorithm. In particular Fletcher and Leyffer [81] compute lower bounds by parametrically taking one step of the dual active set method in every node of the search tree. Their numerical experiments also show that branch-and-bound is the most effective approach out of all the common methods to solve MIQP problems, since the convex QP-relaxations are easy to solve, e.g., by methods outlined in Section 2.4. Standard commercial solvers that can handle mixed-integer quadratically constrained programs (MIQCP), and therefore in particular also CMIQP, include `CPLEX` [57], `Xpress` [78], `Gurobi` [106] and `MOSEK` [156], while `Bonmin` [34] and `SCIP` [2] are well-known non-commercial software packages being capable of solving CMIQP, compare Section 3.3. Recent benchmarks show that `Gurobi` and `CPLEX` perform best on MIQP problems, both implementing state-of-the-art branch-and-bound algorithms.

We propose an extended version of the branch-and-bound algorithm 12 of Chapter 4 to solve CMIQP. Since dualizing the continuous relaxation yields another convex quadratic programming problem with only non-negativity constraints, as we will see in Section 5.4, a new feasible active set method for quadratic programs of that form is specifically designed to compute lower bounds. The main feature of the branch-and-bound algorithm 12 for solving CBMIQP, i.e., the sophisticated preprocessing phase, leading to a fast enumeration of the branch-and-bound nodes, can be easily adapted by small modifications to handle CMIQP. Moreover, our feasible active set method takes advantage of this preprocessing phase and is also well suited for reoptimization. Following the ideas of the branch-and-bound scheme sketched in the last chapter, we keep the fixed branching order that is determined in advance. Recall that on the one hand, we lose the flexibility of choosing sophisticated branching strategies usually used in recent state-of-the art software packages, as shortly reviewed in Section 3.1, but on the other hand, we gain the possibility of shifting expensive computations into a preprocessing phase. As one of the main ideas that differs from standard approaches, our algorithm solves the dual problem of the continuous relaxation in each node in order to determine a local lower bound. Using the concept of duality theory in nonlinear programming, as seen in Section 2.3, strong duality holds if the primal problem is feasible, since all constraints of the continuous relaxation of CMIQP are affine. Thus, computing a lower bound by the dual is useful since on the one hand it can be solved effectively by exploiting its structure and on the other hand we get a lower bound that is as strong as by solving the primal. If additionally the primal problem has a small number of constraints, the dual problem has a small dimension. Extending the sophisticated incremental computations of Chapter 4, the overall running time per node in our approach is still linear in the dimension n if the number of constraints is fixed and if we assume that the solution of the dual problem, which is of fixed dimension in this case, can be computed in constant time. In this sense our approach can be seen as an extension of the

Algorithm 12 without increasing the order of running time per node.

For the solution of the dual problem we investigate the performance of two active set methods. The first one, discussed in Section 5.2, is an *infeasible* active set method by Kunisch and Rendl [129], denoted by KR. It computes a KKT-point by relaxing primal and dual feasibility while requiring stationarity and complementary slackness at every iteration. This relaxation leads to a simplification of the KKT-system to a system of linear equations in the dimension of the set of inactive variables. A theoretical limitation of this algorithm is that it requires a strictly convex objective function. However, this problem can be tackled by adding a small multiple of the identity matrix to the diagonal of Q. To overcome this disadvantage, we propose as a second method, a new *feasible* active set method FAST-QPA, discussed in detail in Section 5.3. A key feature of that active set method is that each of its iterates is feasible. Combining this with properties of duality theory, it suffices to find an approximate solution, as each dual feasible solution yields a valid lower bound. We can thus prune the branch-and-bound node as soon as the current upper bound is exceeded by the value of any feasible iterate produced in a solution algorithm for the dual problem.

Another feature that is used in both active set algorithms within the branch-and-bound is the use of warmstarts: after each branching, corresponding to the fixing of a variable, we pass the optimal active set from the parent node as starting point to the child nodes. This leads to a significant reduction in the average number of iterations needed to solve the dual problem to optimality.

This chapter is organized as follows. In Section 5.1 we give a short overview of existing methods for non-negativity constrained convex quadratic programming problems. In Section 5.2 we discuss the infeasible active set method by Kunisch and Rendl for solving this kind of problem. In Section 5.3 we present our own alternative active set method. The properties of the proposed active set estimation are discussed and the convergence of the algorithm is analyzed. Section 5.4 presents an outline of the branch-and-bound algorithm: we discuss the advantages of considering the corresponding dual problem instead of the primal one. Afterwards, we explain the idea of reoptimization, i.e., using warmstarts within the branch-and-bound scheme. The end of the section deals with some tricks to speed up the algorithm by exploiting incremental computations and an intelligent preprocessing similar to Chapter 4. In Section 5.5 we present our computational results. We compared the performance of both active set methods used within our branch-and-bound framework to the MIQP solver of CPLEX 12.6 on a set of randomly generated instances. We will show that this new approach significantly outperforms the MIQP solver of CPLEX 12.6 for instances with a small number of constraints. Section 5.6 summarizes the results of this chapter.

5.1 Quadratic Programming Problems with Non-negativity Constraints

In the following two sections, we consider non-negatively constrained QP problems of the form

$$\begin{array}{ll} \min & q(x) = x^\top \tilde{Q} x + c^\top x + d \\ \text{s.t.} & x \geq 0 \\ & x \in \mathbb{R}^m, \end{array} \tag{QP}$$

where $\tilde{Q} \in \mathbb{R}^{m \times m}$ is positive definite, $c \in \mathbb{R}^m$ and $d \in \mathbb{R}$, implying that the minimum of QP is finite.

The vast majority of solution methods for (purely continuous) quadratic programs can be categorized into either interior point methods [192] or active set methods [25]. Besides the references in Section 2.5.3, we refer also to [162] and the references therein for further details. In interior point methods, a sequence of parameterized barrier functions is (approximately) minimized using Newton's method. The main computational effort consists in solving the Newton system to get the search direction. Some noteworthy contributions based on interior point methods for the solution of QP are made by Heinkenschloss, Ulbrich and Ulbrich [108] and D'Apuzzo and Marino [64]. In active set methods, at each iteration a working set that estimates the set of active constraints at the solution is iteratively updated. Usually, only a single active constraint is added or deleted to the active set at each iteration. However, when dealing with simple constraints, one can use projected active set methods. These methods usually add/delete more than one constraint at each iteration to/from the current estimated active set to find the optimal active set in a finite number of steps. Among the more recent work on active set based methods for solving QP, we mention the algorithms by Kunisch and Rendl [129], Moré and Toraldo [154, 155] and Dostál and Schöberl [71]. We will discuss the first one in more detail in Section 5.2. The last two algorithms both use the conjugate gradient method to explore the face of the feasible region defined by the current iterate and the reduced gradient projection method to expand the active set.

An advantage of active set methods is that they are well-suited for warmstarts, where a good estimate of the optimal active set is used to initialize the algorithm. This is particularly useful in applications where a sequence of QP problems is solved, e.g., in a sequential quadratic programming method. Since in our branch-and-bound framework we need to solve a large number of closely related quadratic programs, using active set strategies seems to be a reasonable choice.

In the next two sections we will use the following notation. Given a matrix M, we denote by $\lambda_{max}(M)$ the maximum eigenvalue of M. Furthermore, given a vector $v \in \mathbb{R}^m$ and an index set $I \subseteq \{1, \ldots, m\}$, we denote by v_I the sub-vector with components v_i with $i \in I$. Analogously, given the matrix $H \in \mathbb{R}^{m \times m}$

we denote by H_{II} the sub-matrix with components $h_{i,j}$ with $i, j \in I$. Finally, given two vectors $v, y \in \mathbb{R}^m$ we denote by $\max\{v, y\}$ the vector with components $\max\{v_i, y_i\}$, for $i \in \{1, \ldots, m\}$.

For our specific problem QP, recall that the active and non-active sets of variables are defined by the following:

Definition 5.1. Let $x^\star \in \mathbb{R}^m$ be an optimal solution for Problem (QP). We define as *active set* at $x^\star$ the following:

$$\mathcal{A}(x^\star) = \big\{ i \in \{1, \ldots, m\} : x_i^\star = 0 \big\}.$$

We further define as *non-active set* at $x^\star$ the complementary set of $\mathcal{A}(x^\star)$:

$$\mathcal{N}(x^\star) = \{1, \ldots, m\} \setminus \mathcal{A}(x^\star) = \big\{ i \in \{1, \ldots, m\} : x_i^\star > 0 \big\}.$$

The idea behind active set methods in constrained nonlinear programming is that of correctly identifying the set of active constraints at the optimal solution $x^\star$, which then yields $x^\star$ by the solution of an equality constrained QP, see Theorem 2.17. In the following, two specific algorithms to obtain $\mathcal{A}(x^\star)$ are presented.

5.2 The Kunisch-Rendl Active Set Algorithm

The first algorithm is a tailored active set method by Kunisch and Rendl [129] for solving QP. According to Definition 2.2 the KKT-System of QP is given by

$$2\tilde{Q}x + c - \lambda = 0 \tag{5.1}$$

$$-\lambda_i x_i = 0 \ \forall \ i = 1, \ldots, m \tag{5.2}$$

$$-x \leq 0 \tag{5.3}$$

$$\lambda \geq 0. \tag{5.4}$$

Using Theorems 2.14 and 2.13 we have that a pair $(x^\star, \lambda^\star)$ is a solution of that (nonlinear) system if and only if $x^\star$ is a global minimizer of QP.

The active set method uses a reduced KKT-system by relaxing both primal and dual feasibility, i.e., relaxing the constraints (5.3) and (5.4) and choosing $x_\mathcal{A} = 0$ and $\lambda_\mathcal{N} = 0$, where $\mathcal{A}$ and $\mathcal{N}$ denote the current estimates for $\mathcal{A}(x^\star)$ and $\mathcal{N}(x^\star)$, respectively. This a valid choice for the nonlinear equations (5.2), so that a system of linear equations remains. By partitioning (5.1) according to the active and inactive variables, we get

$$\begin{pmatrix} 2\tilde{Q}_{\mathcal{A},\mathcal{A}} & 2\tilde{Q}_{\mathcal{A},\mathcal{N}} \\ 2\tilde{Q}_{\mathcal{N},\mathcal{A}} & 2\tilde{Q}_{\mathcal{N},\mathcal{N}} \end{pmatrix} \begin{pmatrix} x_\mathcal{A} \\ x_\mathcal{N} \end{pmatrix} + \begin{pmatrix} c_\mathcal{A} \\ c_\mathcal{N} \end{pmatrix} + \begin{pmatrix} -\lambda_\mathcal{A} \\ -\lambda_\mathcal{N} \end{pmatrix} = 0. \tag{5.5}$$

Then we solve this system for $x_{\mathcal{N}}$, yielding

$$x_{\mathcal{N}} = -\frac{1}{2}\tilde{Q}^{-1}_{\mathcal{N},\mathcal{N}} c_{\mathcal{N}} \tag{5.6}$$

We can then compute $\lambda_{\mathcal{A}}$ by

$$\lambda_{\mathcal{A}} = c_{\mathcal{A}} + 2\tilde{Q}_{\mathcal{A},\mathcal{N}} x_{\mathcal{N}}. \tag{5.7}$$

Before resolving the updated reduced system (5.5), we compute a new guess for the optimal active set as follows

$$\mathcal{A}(x) = \{i \in \{1, \dots, m\} \mid x_i < 0 \vee \lambda_i > 0\}.$$

The choice of the updated active set can be justified by the following observation. On the one hand, if $\lambda_i > 0$, the solution is dual feasible, so our previous choice $i \in \mathcal{A}$ is confirmed and we keep the index in the subsequent active set. On the other hand, if $x_i < 0$, the solution is primal infeasible and we add the index to our subsequent active set. For the starting guess of the optimal active set $\mathcal{A}(x^\star)$, we choose $\mathcal{A}(x^\star) = \{1, \dots, m\}$, which gives $x = 0$ and $\lambda = c$. Another possible choice would be $A = \emptyset$, such that $\lambda = 0$ and $x = -\frac{1}{2}\tilde{Q}^{-1}c$ or simply choosing A randomly. The KR-algorithm is sketched in Algorithm 13. Note that in each iteration of Algorithm 13 the main effort consists of solving one system of linear equations of dimension $|\mathcal{N}|$, i.e., if $|\mathcal{N}|$ is small the solution process will be cheap. For a more detailed description of the algorithm and convergence results, see [129].

Algorithm 13: Infeasible Active Set Algorithm (KR)

```
input  : $\tilde{Q} \in \mathbb{R}^{m \times m}$ symmetric and positive definite matrix, $c \in \mathbb{R}^m$,
         $\mathcal{A} \subseteq \{1, \dots, m\}$
output: optimal solution $(x^\star, \lambda^\star)$ of (5.1)–(5.4)
while $(x, s)$ not optimal do
    solve (5.5), i.e., set $x_{\mathcal{A}} = 0, \lambda_{\mathcal{N}} = 0$ and compute $x_{\mathcal{N}}$ from (5.6)
    and (5.7). Set $\mathcal{A}(x) = \{i \mid x_i < 0 \vee \lambda_i > 0\}$
```

A drawback of the KR-algorithm is that it constructs iterates that are not necessarily feasible. Therefore not every arbitrary iterate can be used to construct a primal solution. There are two easy ways to adapt this algorithm to turn it into a feasible active set algorithm. The first way is to project any iterate x^k which is not feasible to the non-negative orthant, i.e., for any $i \in \{1, \dots, m\}$ such that $x_i^k < 0$, we simply set $x_i^k = 0$. A similar idea is studied by Hungerländer and Rendl [113]. They obtain primal feasibility by iteratively adding all infeasible components of x^k to the subsequent active set and resolving the system of linear equations (5.6) for x_I until primal feasibility is attained. For the cost of resolving (5.6) several times, we gain primal feasible iterates. Nevertheless both approaches do not improve the performance essentially [113], such that we devise a new feasible active set algorithm in the next section.

5.3 The FAST-QPA Active Set Algorithm

As an alternative to Algorithm 13, we describe a projected active set method for the solution of QP that tries to exploit the information calculated in a preprocessing phase at each level of the branch-and-bound tree. Our method is inspired by the work of Bertsekas [24], where a class of active set projected Newton methods is proposed for the solution of nonlinear problems with non-negativity constraints. The main difference between the two approaches is in the way the active variables are defined and updated. While the method described in [24] uses a line search procedure that both updates active and non-active variables at each iteration, our method, at a given iteration, first sets to zero the active variables (guaranteeing a sufficient reduction of the objective function), and then tries to improve the objective function in the space of the non-active variables. This gives us more freedom in the choice of the step size along the search direction, since the active variables are not considered in the line search procedure.

In this section we denote by $\nabla q(x) \in \mathbb{R}^m$ and $\nabla^2 q(x) \in \mathbb{R}^{m\times m}$ the gradient vector and the Hessian matrix of the objective function $q(x)$ in Problem QP, respectively. Explicitly, we have

$$\nabla q(x) = 2\tilde{Q}x + c, \quad \nabla^2 q(x) = 2\tilde{Q}\ .$$

The open ball with center x and radius $\rho > 0$ is denoted by $B_\rho(x)$. Finally, we denote the projection of a point $z \in \mathbb{R}^m$ onto $\mathbb{R}^m_+$ by $[z]^\sharp := \max\{0, z\}$.

5.3.1 Active Set Estimate

Our aim is to find a rule that leads us to the identification of the optimal active set $\mathcal{A}(x^\star)$ for Problem QP as quickly as possible. The rule we propose is based on the use of multiplier functions and follows the ideas reported in [77].

Definition 5.2. Let $x \in \mathbb{R}^m$. We define the following sets as estimates of the non-active and active sets at x:

$$\tilde{\mathcal{N}}(x) = \{i \in \{1, \dots, m\} : x_i > \varepsilon \, \nabla q_i(x)\}$$

and

$$\tilde{\mathcal{A}}(x) = \{1, \dots, m\} \setminus \tilde{\mathcal{N}}(x),$$

where $\varepsilon > 0$ is a positive scalar.

The following result is taken from [77]:

Theorem 5.1. *Let $x^\star \in \mathbb{R}^m$ be an optimal solution of Problem QP. Then, there exists a neighborhood of $x^\star$ such that, for each x in this neighborhood, we have*

$$\mathcal{A}^+(x^\star) \subseteq \tilde{\mathcal{A}}(x) \subseteq \mathcal{A}(x^\star),$$

with $\mathcal{A}^+(x^\star) = \mathcal{A}(x^\star) \cap \{i \in \{1, \dots, m\} : \nabla q_i(x^\star) > 0\}$.

Proof. To show the first inclusion, let i belong to $\mathcal{A}^+(x^\star)$. Then by definition of $\mathcal{A}^+(x^\star)$, we have

$$x_i^\star = 0 \quad \text{and} \quad \nabla q_i(x^\star) > 0.$$

Then, since both x and ∇q_i are continuous, the result follows immediately. To show the second inclusion, let $i \notin \mathcal{A}(x^\star)$. Noticing that, according to (5.1), for the optimal multiplier $\lambda^\star$ of QP we have $\lambda^\star = \nabla q(x^\star)$, and by using the complementarity condition (5.2) we get

$$x_i^\star > 0 \quad \text{and} \quad \nabla q_i(x^\star) = 0,$$

so that again by continuity of x and ∇q_i, we conclude $i \notin \tilde{\mathcal{A}}(x)$, and the second inclusion is proved. □

Furthermore, if strict complementarity holds, i.e., $\lambda_i^\star > 0$ for all $i \in \mathcal{A}(x^\star)$, we can state the following:

Corollary 2. *Let $x^\star \in \mathbb{R}^m$ be a an optimal solution of Problem QP where strict complementarity holds. Then, there exists a neighborhood of $x^\star$ such that, for each x in this neighborhood, we have*

$$\tilde{\mathcal{A}}(x) = \mathcal{A}(x^\star).$$

5.3.2 Outline of the Algorithm

We now give an outline of our Feasible Active SeT Quadratic Programming Algorithm (`FAST-QPA`) for solving Problem QP; see Algorithm 14 below.

At each iteration k the algorithm first determines the two sets $\mathcal{N}^k := \tilde{\mathcal{N}}(x^k)$ and $\mathcal{A}^k := \tilde{\mathcal{A}}(x^k)$ according to Definition 5.2. Then, the variables belonging to $\mathcal{A}^k$ are set to zero, thus obtaining a new point $\tilde{x}^k$, and a search direction d^k is calculated. More specifically $d^k_{\mathcal{A}^k}$, the components of d^k related to $\mathcal{A}^k$, are set to zero, while a gradient related direction $d^k_{\mathcal{N}^k}$ is calculated in $\mathcal{N}^k$. Here we define that a direction $d_{\tilde{\mathcal{N}}(x^k)}$ is gradient related in $\tilde{x}^k$ (see, e.g., [26]) if there exist $\sigma_1, \sigma_2 > 0$ such that $d_{\tilde{\mathcal{N}}(x^k)}$ satisfies the following two conditions:

$$d^\top_{\tilde{\mathcal{N}}(x^k)} \nabla q(\tilde{x}^k)_{\tilde{\mathcal{N}}(x^k)} \leq -\sigma_1 \|\nabla q(\tilde{x}^k)_{\tilde{\mathcal{N}}(x^k)}\|^2, \tag{5.8}$$

$$\|d_{\tilde{\mathcal{N}}(x^k)}\| \leq \sigma_2 \|\nabla q(\tilde{x}^k)_{\tilde{\mathcal{N}}(x^k)}\|. \tag{5.9}$$

Algorithm 14: Feasible Active SeT Quadratic Programming Algorithm (FAST-QPA)

```
input  : a quadratic programming problem of the form QP
output: a minimizer $x^\star \in \mathbb{R}^n$ of QP
Fix $\delta \in (0,1)$, $\gamma \in (0, \frac{1}{2})$, $\varepsilon > 0$ and maxit $> 0$.
Choose $x^0 \in \mathbb{R}^n_+$, set $k = 0$.
for $k = 0, 1, \ldots,$ maxit do
    Compute $\mathcal{A}^k := \tilde{\mathcal{A}}(x^k)$ and $\mathcal{N}^k := \tilde{\mathcal{N}}(x^k)$.
    Set $\tilde{x}^k_{\mathcal{A}^k} = 0$ and $\tilde{x}^k_{\mathcal{N}^k} = x^k_{\mathcal{N}^k}$.
    Set $d^k_{\mathcal{A}^k} = 0$.
    Compute a gradient related direction $d^k_{\mathcal{N}^k}$ in $\tilde{x}^k$ (Algorithm 15).
    Set $\alpha^k = \delta^j$ where $j$ is the first non-negative integer for which
        $$q([\tilde{x}^k + \delta^j d^k]^\sharp) \le q(\tilde{x}^k) + \gamma\, \delta^j\, \nabla q(\tilde{x}^k)^\top d^k.$$
    Set
        $$x^{k+1} = [\tilde{x}^k + \alpha^k d^k]^\sharp.$$
```

Algorithm 15: Calculation of the direction $d^k_{\mathcal{N}^k}$

```
input  : a current iterate $\tilde{x}^k$ of Algorithm 14
output: a gradient related direction $d^k_{\mathcal{N}^k}$ in $\tilde{x}^k$
Choose $y^0 = \tilde{x}^k_{\mathcal{N}^k}$.
Set $\nabla q^0 = 2\tilde{Q}y^0 + c$, $d^0 = -\nabla q^0$ and $l = 0$.
while Stopping Condition not satisfied do
    Compute step size along the search direction $d^l$:
        $$\alpha^l = \frac{\|\nabla q^l\|^2}{{d^l}^\top \tilde{Q} d^l}$$
    Update point $y^{l+1} = y^l + \alpha^l\, d^l$.
    Update gradient of the quadratic function $\nabla q^{l+1} = \nabla q^l + \alpha^l\, \tilde{Q}\, d^l$.
    Compute coefficient
        $$\beta^{l+1} = \frac{\|\nabla q^{l+1}\|^2}{\|\nabla q^l\|^2}.$$
    Determine new conjugate direction $d^{l+1} = -\nabla q^{l+1} + \beta^{l+1} d^l$.
    Set $l = l + 1$.
Set $d^k_{\mathcal{N}^k} = y^l - y^0$.
```

In order to obtain the direction $d^k_{\mathcal{N}^k}$, we apply Algorithm 15, a conjugate gradient method, to the following unconstrained quadratic problem in the subspace of non-active variables,

$$\begin{array}{ll} \min & q^k(y) = y^\top \tilde{Q}_{\mathcal{N}^k\mathcal{N}^k} y + c^\top_{\mathcal{N}^k} y \\ \text{s.t.} & y \in \mathbb{R}^{|\mathcal{N}^k|}. \end{array} \tag{QP$_k$}$$

When the minimum of Problem QP$_k$ is finite, it follows from Theorem 2.25 that the conjugate gradient method produces an optimal solution y of Problem QP$_k$ in at most $|\mathcal{N}^k|$ iterations. Otherwise, the algorithm stops after a finite number of iterations with a vector y which is a combination of the conjugate directions generated along the iterations by the conjugate gradient method. In both cases, once a vector y is calculated, we choose $d^k_{\mathcal{N}^k} := y - \tilde{x}^k_{\mathcal{N}^k}$. Using standard results [66, 101] it can be proven that $d^k_{\mathcal{N}^k}$ satisfies (5.8) and (5.9) and hence is gradient related. Finally, the new point x^{k+1} is computed by means of a projected Armijo line search along the direction d^k. Note that, even if the matrix $\tilde{Q}$ is ill-conditioned, the conjugate gradient method embedded into Algorithm 14 still generates a gradient related direction at each iteration. In case one wants to exactly solve Problem QP$_k$, suitable preconditioning techniques can be applied to the conjugate gradient method.

5.3.3 Convergence Analysis

The convergence analysis of `FAST-QPA` is based on two key results, namely Proposition 5.2 and Proposition 5.3 stated below. These results show that the algorithm obtains a significant reduction of the objective function both when fixing to zero the variables in the active set estimate and when we perform the projected Armijo line search.

Proposition 5.2 completes the properties of the active set identification strategy defined in Section 5.3.1. More specifically, it shows that, for a suitably chosen value of the parameter ε appearing in Definition 5.2, a decrease of the objective function is achieved by simply fixing to zero one or more variables whose indices belong to the estimated active set.

Proposition 5.2. *Assume that the parameter ε appearing in Definition 5.2 satisfies*

$$0 < \varepsilon < \frac{1}{2\lambda_{max}(\tilde{Q})} \, . \tag{5.10}$$

Given the point x^k and the set $\mathcal{A}^k$, let $\mathcal{A}^y$ be a set of indices and let y be a point such that

$$\mathcal{A}^y \subseteq \{i \in \mathcal{A}^k : x^k_i \neq 0\},$$
$$y_{\mathcal{A}^y} = 0, \quad y_{I\setminus\mathcal{A}^y} = x^k_{I\setminus\mathcal{A}^y},$$

with $I = \{1, \ldots, m\}$. Then,

$$q(y) - q(x^k) \leq -\frac{1}{2\varepsilon}\|y - x^k\|^2.$$

Proof. By taking into account the definition of the set $\mathcal{A}^y$ and the point y, we have

$$q(y) = q(x^k) + \nabla q_{\mathcal{A}^y}(x^k)^\top (y - x^k)_{\mathcal{A}^y} + \frac{1}{2}(y - x^k)^\top_{\mathcal{A}^y}(\nabla^2 q)_{\mathcal{A}^y\mathcal{A}^y}(y - x^k)_{\mathcal{A}^y} \ .$$

Since $\nabla^2 q = 2Q$, the inequality

$$q(y) \leq q(x^k) + \nabla q_{\mathcal{A}^y}(x^k)^\top (y - x^k)_{\mathcal{A}^y} + \lambda_{max}(\tilde{Q}) \, \|(y - x^k)_{\mathcal{A}^y}\|^2$$

holds. Using (5.10) we obtain

$$q(y) \leq q(x^k) + \nabla q_{\mathcal{A}^y}(x^k)^\top (y - x^k)_{\mathcal{A}^y} + \frac{1}{2\varepsilon}\|(y - x^k)_{\mathcal{A}^y}\|^2$$

and hence

$$q(y) \leq q(x^k) + \Big(\nabla q_{\mathcal{A}^y}(x^k) + \frac{1}{\varepsilon}(y - x^k)_{\mathcal{A}^y}\Big)^\top (y - x^k)_{\mathcal{A}^y} - \frac{1}{2\varepsilon}\|(y - x^k)_{\mathcal{A}^y}\|^2.$$

It thus remains to show

$$\Big(\nabla q_{\mathcal{A}^y}(x^k) + \frac{1}{\varepsilon}(y - x^k)_{\mathcal{A}^y}\Big)^\top (y - x^k)_{\mathcal{A}^y} \leq 0 \ ,$$

which follows from the fact that, for all $i \in \mathcal{A}^y$,

$$\Big(\nabla q_i(x^k) + \frac{1}{\varepsilon}(y_i - x_i^k)\Big)^\top (y_i - x_i^k) \leq 0.$$

Indeed, for all $i \in \mathcal{A}^y$ we have $x_i^k \geq 0$ and $y_i = 0$, hence $x_i^k - y_i = x_i^k \leq \varepsilon \, \nabla q_i(x^k)$, so that

$$\nabla q_i(x^k) + \frac{1}{\varepsilon}\,(y_i - x_i^k) \geq 0 \ .$$

□

The following result shows that the projected Armijo line search performed in Algorithm 14 terminates in a finite number of steps, and that the new point obtained guarantees a decrease of the objective function of QP.

Proposition 5.3. *Let $\gamma \in (0, \frac{1}{2})$. Then, for every $\bar{x} \in \mathbb{R}^n_+$ with $\nabla q(\bar{x})_{\tilde{\mathcal{N}}(\bar{x})} \neq 0$, there exist $\rho > 0$ and $\bar{\alpha} > 0$ such that*

$$q([x + \alpha d]^\sharp) - q(x) \leq \gamma \, \alpha \, d^\top_{\tilde{\mathcal{N}}(\bar{x})} \nabla q(x)_{\tilde{\mathcal{N}}(\bar{x})} \qquad (5.11)$$

for all $x \in \mathbb{R}^n_+$ with $x \in B_\rho(\bar{x})$ and for all $\alpha \in (0, \bar{\alpha}]$, where $d \in \mathbb{R}^n$ is the direction used at x satisfying (5.8) *and* (5.9)*.*

Its proof is similar to the proof of Proposition 2 in [24].

Proof. From the continuity of the objective function of problem QP, we have that the gradient ∇q is Lipschitz continuous on $\mathbb{R}^n_+$, so that there exists $L < \infty$ such that, for $s \in [0, 1]$

$$\|\nabla q(x) - \nabla q(x - s[x - x(\alpha)])\| \leq sL\|x - x(\alpha)\| \;\; \forall\, x \in B_\rho(\bar{x}) \cap \mathbb{R}^n_+,$$

where $x(\alpha) := (x + \alpha d)^\sharp$. Furthermore, ∇q is bounded on $B_\rho(\bar{x}) \cap \mathbb{R}^n_+$, hence, there exists $\sigma > 0$ such that

$$\|\nabla q(x)\| \leq \frac{\sigma}{\sigma_2}, \qquad \forall\, x \in B_\rho(\bar{x}) \cap \mathbb{R}^n_+$$

and by (5.9) d is also bounded on $B_\rho(\bar{x}) \cap \mathbb{R}^n_+$. For $x(\alpha) \in B_\rho(\bar{x}) \cap \mathbb{R}^n_+$ we have

$$\begin{array}{rcl} q(x(\alpha)) - q(x) & = & \nabla q(x)^\top (x(\alpha) - x) \\ & + & \displaystyle\int_0^1 \big(\nabla q(x - s[x - x(\alpha)]) - \nabla q(x)\big)^\top \big(x(\alpha) - x\big) ds \\ & \leq & \nabla q(x)^\top (x(\alpha) - x) + \|x(\alpha) - x\| \displaystyle\int_0^1 sL\|x(\alpha) - x\| ds \\ & = & \nabla q(x)^\top (x(\alpha) - x) + \dfrac{L}{2}\|x(\alpha) - x\|^2. \end{array}$$

Then, from the definition of $d_{\tilde{\mathcal{A}}(\bar{x})}$ in Algorithm 14, we have

$$x(\alpha)_{\tilde{\mathcal{A}}(\bar{x})} = x_{\tilde{\mathcal{A}}(\bar{x})}.$$

Therefore, it is possible to write

$$q(x(\alpha)) - q(x) \;\; \leq \;\; \nabla q(x)^\top_{\tilde{\mathcal{N}}(\bar{x})} (x(\alpha) - x)_{\tilde{\mathcal{N}}(\bar{x})} + \frac{L}{2}\|(x(\alpha) - x)_{\tilde{\mathcal{N}}(\bar{x})}\|^2. \quad (5.12)$$

We now majorize the terms in the right hand side. of (5.12). We first analyze the term $\|(x(\alpha) - x)_{\tilde{\mathcal{N}}(\bar{x})}\|^2$.

Let $x \in B_\rho(\bar{x}) \cap \mathbb{R}^n_+$. We distinguish between $d_i \geq 0$ and $d_i < 0$.

- If $d_i \geq 0$, we have

  $$x_i(\alpha) = x_i + \alpha d_i$$

  for all $\alpha \geq 0$.

  Thus,

  $$(x_i(\alpha) - x_i)^2 = \alpha^2 d_i^2. \quad (5.13)$$

- If $d_i < 0$, we have
$$x_i(\alpha) \geq x_i + \alpha d_i$$
for all $\alpha \geq 0$.

  Hence,
$$0 \leq x_i - x_i(\alpha) \leq -\alpha d_i,$$
so that
$$0 \leq (x_i - x_i(\alpha))^2 \leq \alpha^2 d_i^2. \tag{5.14}$$

Then by (5.13) and (5.14) we have
$$\|(x(\alpha) - x)_{\tilde{\mathcal{N}}(\bar{x})}\|^2 \leq \alpha^2 \|d_{\tilde{\mathcal{N}}(\bar{x})}\|^2.$$

Using (5.8) and (5.9)
$$\|(x(\alpha) - x)_{\tilde{\mathcal{N}}(\bar{x})}\|^2 \leq \alpha^2 \, \|d_{\tilde{\mathcal{N}}(\bar{x})}\|^2 \leq \alpha^2 \, \sigma_2^2 \, \|\nabla q(x)_{\tilde{\mathcal{N}}(\bar{x})}\|^2 \leq -\alpha^2 \, \frac{\sigma_2^2}{\sigma_1} \, d_{\tilde{\mathcal{N}}(\bar{x})}^\top \nabla q(x)_{\tilde{\mathcal{N}}(\bar{x})}. \tag{5.15}$$

Now we consider the first term in the right hand side of (5.12). By condition (5.9) on the search direction we have that for all $x \in B_\rho(\bar{x}) \cap \mathbb{R}^n_+$ and for all $i \in \tilde{\mathcal{N}}(\bar{x})$:
$$|d_i| \leq \|d_{\tilde{\mathcal{N}}(\bar{x})}\| \leq \sigma_2 \, \|\nabla q(x)_{\tilde{\mathcal{N}}(\bar{x})}\| \leq \sigma.$$

Then we introduce
$$\omega = \min_{i:\nabla q_i(x) > 0} \varepsilon \nabla q_i(x).$$

Consider first the case that $i \in \tilde{\mathcal{N}}(\bar{x})$ and $\nabla q_i(x) = 0$. For such indices we have of course that
$$\nabla q_i(x)(x(\alpha)_i - x_i) = \alpha \, \nabla q_i(x) d_i.$$

When $i \in \tilde{\mathcal{N}}(\bar{x})$ and $\nabla q_i(x) \neq 0$ according to our estimation of the non-active set, we have two different subcases:

- If $\nabla q_i(x) > 0$, then by choosing $\alpha \in (0, \omega/\sigma]$ we have that:
$$x_i(\alpha) - x_i = \alpha d_i. \tag{5.16}$$

- If $\nabla q_i(x) < 0$, we have for all $\alpha > 0$
$$x_i(\alpha) - x_i \geq \alpha d_i,$$
so that multiplying by $\nabla q_i(x) < 0$ we have
$$\nabla q_i(x)\big(x(\alpha)_i - x_i\big) \leq \alpha \, \nabla q_i(x) d_i. \tag{5.17}$$

Therefore, choosing $\alpha \in (0, \omega/\sigma]$ we have, from (5.16) and (5.17), that

$$\nabla q(x)_{\tilde{\mathcal{N}}(\bar{x})}^\top (x(\alpha) - x)_{\tilde{\mathcal{N}}(\bar{x})} \leq \alpha \, \nabla q(x)_{\tilde{\mathcal{N}}(\bar{x})}^\top d_{\tilde{\mathcal{N}}(\bar{x})}. \tag{5.18}$$

Substituting (5.15) and (5.18) into (5.12) we obtain that for all $\alpha \in (0, \omega/\sigma]$ and $x(\alpha) \in B_\rho(\bar{x}) \cap \mathbb{R}^n_+$

$$q(x(\alpha)) - q(x) \leq \alpha \, (1 - \alpha\sigma_2^2 L/2\sigma_1) \, \nabla q(x)_{\tilde{\mathcal{N}}(\bar{x})}^\top d_{\tilde{\mathcal{N}}(\bar{x})}. \tag{5.19}$$

Hence from (5.19) we see that (5.11) is satisfied by choosing α in such a way that the following inequality holds:

$$1 - \alpha \, \frac{L \, \sigma_2^2}{2\sigma_1} > \gamma.$$

We can conclude that the statement holds for all $x \in B_\rho(\bar{x}) \cap \mathbb{R}^n_+$, $\alpha \in (0, \bar{\alpha}]$, where

$$\bar{\alpha} = \min \left\{ \frac{\omega}{\sigma}, \frac{2\sigma_1 \, (1 - \gamma)}{\sigma_2^2 L} \right\}. \tag{5.20}$$

□

Proposition 5.4. *Suppose that* (5.10) *holds and that* **`FAST-QPA`** *produces an infinite sequence* $\{x^k\}$*. Then the sequence* $\{q(x^k)\}$ *converges.*

Proof. Let $\tilde{x}^k$ be the point produced in Algorithm 14. By setting $y = \tilde{x}^k$ in Proposition 5.2, we have

$$q(\tilde{x}^k) \leq q(x^k) - \frac{1}{2\varepsilon} \|\tilde{x}^k - x^k\|^2 .$$

Furthermore, by the fact that we use an Armijo line search in Algorithm 14 and by Proposition 5.3, we have that the chosen point x^{k+1} satisfies inequality (5.11), that is

$$q(x^{k+1}) - q(\tilde{x}^k) \leq \gamma \, \alpha \, d_{\mathcal{N}^k}^\top \nabla q(\tilde{x}^k)_{\mathcal{N}^k}.$$

By taking into account (5.8), we thus have

$$q(x^{k+1}) - q(\tilde{x}^k) \leq \gamma \, \alpha \, d_{\mathcal{N}^k}^\top \nabla q(\tilde{x}^k)_{\mathcal{N}^k} \leq -\sigma_1 \|\nabla q(\tilde{x}^k)_{\mathcal{N}^k}\|^2 .$$

In summary, we obtain

$$\begin{aligned} q(x^{k+1}) &\leq q(\tilde{x}^k) - \sigma_1 \|\nabla q(\tilde{x}^k)_{\mathcal{N}^k}\|^2 \leq \\ &\leq q(x^k) - \frac{1}{2\varepsilon} \|\tilde{x}^k - x^k\|^2 - \sigma_1 \|\nabla q(\tilde{x}^k)_{\mathcal{N}^k}\|^2 . \end{aligned} \tag{5.21}$$

In particular, the sequence $\{q(x^k)\}$ is monotonously decreasing and bounded from below by the minimum of QP, which by our assumption is finite. Hence it converges. □

Finally, we are able to prove the main result concerning the global convergence of `FAST-QPA`.

Theorem 5.5. *Assume that the parameter ε appearing in Definition 5.2 satisfies* (5.10). *Let $\{x^k\}$ be the sequence produced by Algorithm* ***`FAST-QPA`****. Then either an integer $\bar{k} \geq 0$ exists such that $x^{\bar{k}}$ is an optimal solution for Problem QP, or the sequence $\{x^k\}$ is infinite and every limit point $x^\star$ of the sequence is an optimal solution for Problem QP.*

Proof. Let $x^\star$ be any limit point of the sequence $\{x^k\}$ and let $\{x^k\}_K$ be the subsequence with

$$\lim_{k\to\infty,\, k\in K} x^k = x^\star .$$

By considering an appropriate subsequence, we may assume that there exist subsets $\bar{\mathcal{A}} \subseteq \{1,\dots,m\}$ and $\bar{\mathcal{N}} = \{1,\dots,m\} \setminus \bar{\mathcal{A}}$ such that $\mathcal{A}^k = \bar{\mathcal{A}}$ and $\mathcal{N}^k = \bar{\mathcal{N}}$ for all $k \in K$, since the number of possible choices of $\mathcal{A}^k$ and $\mathcal{N}^k$ is finite. In order to prove that $x^\star$ is optimal for Problem QP, it then suffices to show

(i) $\min\{\nabla q_i(x^\star), x_i^\star\} = 0$ for all $i \in \bar{\mathcal{A}}$, and

(ii) $\nabla q_i(x^\star) = 0$ for all $i \in \bar{\mathcal{N}}$.

In order to show (i), let $\hat{\imath} \in \bar{\mathcal{A}}$ and define a function $\Phi_{\hat{\imath}} : \mathbb{R}^m \to \mathbb{R}$ by

$$\Phi_{\hat{\imath}}(x) = \min\{\nabla q_{\hat{\imath}}(x), x_{\hat{\imath}}\} \ ,$$

we thus have to show $\Phi_{\hat{\imath}}(x^\star) = 0$. For $k \in K$, define $\tilde{y}^k \in \mathbb{R}^m$ as follows:

$$\tilde{y}_i^k = \begin{cases} 0 & \text{if } i = \hat{\imath} \\ x_i^k & \text{otherwise.} \end{cases} \tag{5.22}$$

Recalling that $\tilde{x}^k_{\bar{\mathcal{A}}} = 0$, as set in Algorithm 14, and using $\hat{\imath} \in \bar{\mathcal{A}}$, we have

$$\|\tilde{y}^k - x^k\|^2 = (\tilde{y}^k - x^k)^2_{\hat{\imath}} = (\tilde{x}^k - x^k)^2_{\hat{\imath}} \leq \|\tilde{x}^k - x^k\|^2 \ .$$

From (5.21) and Proposition 5.4 we obtain

$$\lim_{k\to\infty,\, k\in K} \|\tilde{x}^k - x^k\|^2 = 0 \ ,$$

hence

$$\lim_{k\to\infty,\, k\in K} \tilde{y}^k = x^\star \ . \tag{5.23}$$

By Definition 5.2, we have $0 \leq x^k_{\hat{\imath}} \leq \varepsilon \nabla q_{\hat{\imath}}(x^k)$ for all $k \in K$. Using Assumption (5.10), there exists $\xi \geq 0$ such that

$$\varepsilon \leq \frac{1}{2\tilde{Q}_{\hat{\imath}\hat{\imath}} + \xi} \ .$$

As $\tilde{y}^k_{\hat{\imath}} = 0$, we obtain

$$x^k_{\hat{\imath}} - \tilde{y}^k_{\hat{\imath}} = x^k_{\hat{\imath}} \leq \varepsilon \nabla q_{\hat{\imath}}(x^k) \leq \frac{1}{2\tilde{Q}_{\hat{\imath}\hat{\imath}} + \xi} \nabla q_{\hat{\imath}}(x^k)$$

and hence

$$(2\tilde{Q}_{\hat{\imath}\hat{\imath}} + \xi)(x^k_{\hat{\imath}} - \tilde{y}^k_{\hat{\imath}}) \leq \nabla q_{\hat{\imath}}(x^k),$$

which can be rewritten as

$$\nabla q_{\hat{\imath}}(x^k) + 2\tilde{Q}_{\hat{\imath}\hat{\imath}}(\tilde{y}^k_{\hat{\imath}} - x^k_{\hat{\imath}}) \geq \xi(x^k_{\hat{\imath}} - \tilde{y}^k_{\hat{\imath}}) \geq 0,$$

yielding $\nabla q_{\hat{\imath}}(\tilde{y}^k) \geq 0$. Together with $\tilde{y}^k_{\hat{\imath}} = 0$, we obtain $\Phi_{\hat{\imath}}(\tilde{y}^k) = 0$. By (5.23) and the continuity of $\Phi_{\hat{\imath}}$, we derive $\Phi_{\hat{\imath}}(x^\star) = 0$, which proves (i).

To show (ii), assume on contrary that $\nabla q(x^\star)_{\bar{\mathcal{N}}} \neq 0$. By Proposition 5.3, there exists $\bar{\alpha} > 0$ such that for $k \in K$ sufficiently large

$$q([\tilde{x}^k + \alpha d^k]^\sharp) - q(\tilde{x}^k) \leq \gamma\alpha \sum_{i \in \mathcal{N}^k} \nabla q_i(\tilde{x}^k) d^k_i$$

for all $\alpha \in (0, \bar{\alpha}]$, as $\tilde{x}^k_{\bar{\mathcal{N}}}$ converges to $x^\star_{\bar{\mathcal{N}}}$. As we use an Armijo type rule in Algorithm 14, we thus have $\delta^{j-1} \geq \bar{\alpha}$ and hence

$$\alpha^k = \delta^j \geq \delta\bar{\alpha}\ . \tag{5.24}$$

Again by the Armijo rule, using (5.8) and (5.24), we obtain

$$\begin{aligned} q(\tilde{x}^k) - q([\tilde{x}^k + \alpha^k d^k]^\sharp) &\geq -\gamma\alpha^k \sum_{i \in \mathcal{N}^k} \nabla q_i(\tilde{x}^k) d^k_i \\ &\geq \gamma\sigma_1\alpha^k \|\nabla q(\tilde{x}^k)_{\mathcal{N}^k}\|^2 \\ &\geq \gamma\sigma_1\delta\bar{\alpha} \|\nabla q(\tilde{x}^k)_{\mathcal{N}^k}\|^2\ . \end{aligned}$$

As the left hand side expression converges to zero by Proposition 5.4, while the right hand side expression converges to

$$\gamma\sigma_1\delta\bar{\alpha}\|\nabla q(x^\star)_{\bar{\mathcal{N}}}\|^2 > 0\ ,$$

we have the desired contradiction. □

Corollary 3. *If* (5.10) *holds, then the sequence* $q(x^k)$ *converges to the minimum of QP.*

As a final result, we prove that, if strict complementarity holds, `FAST-QPA` finds an optimal solution in a finite number of iterations.

Theorem 5.6. *Assume that there exists an accumulation point* $x^\star$ *of the sequence* $\{x^k\}$ *generated by* `FAST-QPA` *such that* *(i) strict complementarity holds in $x^\star$ and*

(ii) the problem

$$\begin{array}{ll} \min & y^\top \tilde{Q}_{\mathcal{N}\mathcal{N}} y + c_{\mathcal{N}}^\top y \\ \text{s.t.} & y \in \mathbb{R}^{|\mathcal{N}|}, \end{array} \tag{5.25}$$

with $\mathcal{N} = \mathcal{N}(x^\star)$, admits a finite solution.

Then `FAST-QPA` produces a minimizer of Problem QP in a finite number of steps.

Proof. Let $\{x^k\}$ be the sequence generated by `FAST-QPA` and let $\{x^k\}_K$ be a converging subsequence. By Theorem 5.5, the limit point $x^\star$ of $\{x^k\}_K$ is a minimizer of Problem QP. Now define

$$\xi = \min_{i \in \mathcal{N}(x^\star)} \{x_i^\star\} > 0 \; .$$

From Corollary 2, we have that for $k \in K$ sufficiently large

$$\mathcal{A}^k = \mathcal{A}(x^\star) \tag{5.26}$$

and

$$\min_{i \in \mathcal{N}^k} \{x_i^k\} = \min_{i \in \mathcal{N}^k} \{\tilde{x}_i^k\} \geq \frac{\xi}{2} \; .$$

The search direction $d_{\mathcal{N}^k}^k$ computed in Algorithm 14 is gradient related and, since

$$\lim_{k \to \infty,\, k \in K} \|\nabla q_{\mathcal{N}^k}(\tilde{x}^k)\| = 0, \tag{5.27}$$

we have

$$\lim_{k \to \infty,\, k \in K} \|d_{\mathcal{N}^k}^k\| = 0.$$

Therefore, for k sufficiently large,

$$(x^k + d^k)_{\mathcal{N}^k} = (\tilde{x}^k + d^k)_{\mathcal{N}^k}$$

and

$$(\tilde{x}^k + d^k)_i \geq \frac{\xi}{2} > 0, \quad \forall i \in \mathcal{N}^k \; .$$

This means that

$$[(\tilde{x}^k + d^k)_{\mathcal{N}^k}]^\sharp = (\tilde{x}^k + d^k)_{\mathcal{N}^k} \; ,$$

and $\tilde{x}^k + d^k$ is feasible for Problem QP. For k sufficiently large, $k \in K$, Corollary 2, (5.26) and (5.27) ensure that

$$\mathcal{A}(x^\star) = \mathcal{A}^k = \mathcal{A}^{k+1} \tag{5.28}$$

and

$$\mathcal{N}(x^\star) = \mathcal{N}^k = \mathcal{N}^{k+1} \; . \tag{5.29}$$

Since, taking into account (5.29) and point *ii*) in our assumptions, $(\tilde{x}^k + d^k)_{\mathcal{N}^k}$ is the optimal solution for problem QP_k, we have

$$\nabla q_{\mathcal{N}^k}(\tilde{x}^k + d^k) = 0, \tag{5.30}$$

$$q(\tilde{x}^k + d^k) = q(\tilde{x}^k) + \frac{1}{2}\nabla q(\tilde{x}^k)^\top d^k. \tag{5.31}$$

By (5.31), the test in the projected Armijo line search of Algorithm 14 is satisfied with $\alpha = 1$. Hence, at the end of Algorithm 14, we set $x^{k+1} = \tilde{x}^k + d^k$. By (5.30), we obtain $\nabla q_{\mathcal{N}^k}(x^{k+1}) = 0$, so that x^{k+1} is stationary with respect to the non-active variables.

To conclude the proof, we need to show that x^{k+1} is stationary with respect to the active variables, too. By (5.28), we have

$$x^{k+1}_{\mathcal{A}^k} = 0, \quad \nabla q_{\mathcal{A}^k}(x^{k+1}) \geq 0,$$

and the stationarity of x^{k+1} with respect to the active variables is also proved. □

5.4 A Branch-and-Bound Algorithm for CMIQP

In this section we shortly summarize the key ingredients of our branch-and-bound algorithm, called `FAST-QPA-BB`, that can be used with any active set algorithm for the node relaxations. In particular we analyze its performance in combination with the infeasible `KR` algorithm and the feasible `FAST-QPA` algorithm, presented in Sections 5.2 and 5.3. The branch-and-bound framework we consider is based on the one discussed in Chapter 4 where the unconstrained case is addressed. As this branch-and-bound scheme aims at a fast enumeration of the nodes, it is reasonable to use a relaxation that can be solved quickly to compute the lower bound, e.g., the continuous relaxation of CMIQP. These computations can be improved by using duality theory for nonlinear programming. Instead of considering the primal formulation of the continuous relaxation of CMIQP, we deal with the dual one. The resulting non-negativity constrained convex quadratic programming problems are solved by either `KR` or `FAST-QPA`. We use the solution as a lower bound for f over all feasible integer points. The obtained lower bounds are as strong as the ones obtained by solving the primal problem, since strong duality holds.

Our branching strategy and its advantages are discussed in Section 5.4.1. In Section 5.4.2, we have a closer look at the relation between the primal and the dual problem, while in Section 5.4.3 we shortly discuss the advantage of reoptimization. Using a predetermined branching order, some of the expensive calculations can be moved into a preprocessing phase, as described in Section 5.4.4.

5.4.1 Branching

We adapt our branching strategy of Algorithm 12 in Section 4.2 to `FAST-QPA-BB`. As explained in this section, at every node in our branch-and-bound scheme, we branch by fixing a single *primal* variable in increasing distance to its value in the solution of the continuous relaxation $\bar{x}$. The branching order is kept predetermined. Besides adapting properly Q_ℓ, $\bar{c}$ and $\bar{d}$ at every level according to Section 4.3, we now also adapt A_ℓ and $\bar{b}$ of the reduced linear constraints. This is done directly by deleting the corresponding columns of A and moving the values of the fixed variables to the right hand side. Recall that the resulting subproblems are again quadratic programming problems of type CMIQP with a dimension decreased by one. Analogously to the bound-constrained case, due to the strict convexity of f, we get a finite branch-and-bound algorithm.

5.4.2 Dual Approach

In the following, we derive the dual problem of the continuous relaxation of CMIQP and point out some advantages when using the dual approach in the branch-and-bound framework. The connection between a convex quadratic program and its dual was first investigated by Dorn [69]. The dual can be computed by first forming the Lagrangian of the relaxation

$$\mathscr{L}(x,\lambda) = x^\top Q x + c^\top x + d + \lambda^\top (Ax - b)$$

and then, for fixed λ, minimizing $\mathscr{L}$ with respect to the primal variables x. As Q is assumed to be positive definite, the unique minimizer can be computed from the stationarity condition of the KKT-system, see Definition 2.2.

$$\nabla_x \mathscr{L}(x,\lambda) = 2Qx + c + A^\top \lambda = 0 \Longleftrightarrow x = -\frac{1}{2} Q^{-1}(c + A^\top \lambda). \tag{5.32}$$

Having x as a function of λ, we can insert it into the Lagrangian $\mathscr{L}$ yielding the following dual function

$$\mathscr{L}(\lambda) = \lambda^\top \big(-\frac{1}{4} A Q^{-1} A^\top \big) \lambda - \big(b^\top + \frac{1}{2} c^\top Q^{-1} A^\top \big) \lambda - \frac{1}{4} c^\top Q^{-1} c + d.$$

Defining $\tilde{Q} := \frac{1}{4} A Q^{-1} A^\top$, $\tilde{c} := \frac{1}{2} A Q^{-1} c + b$ and $\tilde{d} := \frac{1}{4} c^\top Q^{-1} c - d$, we can thus write the dual of the continuous relaxation of CMIQP as

$$\begin{aligned} -\min\ & \lambda^\top \tilde{Q} \lambda + \tilde{c}^\top \lambda + \tilde{d} \\ \text{s.t. } & \lambda \geq 0 \\ & \lambda \in \mathbb{R}^m. \end{aligned} \tag{5.33}$$

Note that (5.33) is again a convex QP, since $\tilde{Q}$ is positive semidefinite.

The first crucial difference in considering the dual problem is that its dimension changed from n to m, which is beneficial if $m \ll n$. The second one is that $\lambda = 0$ is always feasible for (5.33). Finally, note that having the optimal solution $\lambda^\star \in \mathbb{R}^m$ of (5.33), it is easy to reconstruct the corresponding optimal primal solution $x^\star \in \mathbb{R}^n$ using the first order optimality condition (5.32).

Within a branch-and-bound framework, a special feature of the dual approach is *early pruning*: we can stop the iteration process and prune the node as soon as the current iterate λ^k is feasible and its objective function value exceeds the current upper bound, since each dual feasible solution yields a valid bound. Note however that, in case we cannot prune, an optimal solution of the dual problem is required, since it is needed for the computation of the corresponding primal solution $x^\star$ which in turn is needed to decide the enumeration order in the branch-and-bound scheme.

During the tree search it may occur that a node relaxation is infeasible due to the current fixings. In this case infeasibility of the primal problem implies the unboundness of the dual problem. Therefore, during the solution process of the dual problem, an iterate will be reached such that its objective function value exceeds the current upper bound and the node can be pruned. This is why in our implementation of `FAST-QPA` we set the following stopping criterion: the algorithm stops either if the norm of the projected gradient is less than a given optimality tolerance, or if an iterate is computed such that its objective function value exceeds the current upper bound. In case the primal problem has no feasible solution, no finite upper bound is available. This has to be decided in advance, e.g., by using a phase-1 approach. In our implementation we used `CPLEX` for this.

We want to emphasize two advantages of `FAST-QPA` compared to `KR`. First, while we can always use `FAST-QPA` directly to solve (5.33), the algorithm `KR` of Kunisch and Rendl can only handle strictly convex objective functions. In case the function becomes positive semidefinite during the search tree, it is needed to make it artificially positive definite by adding a small multiple of the identity matrix to the quadratic matrix $\tilde{Q}$. From a numerical and practical point of view that does not make any difference, nevertheless the problem gets slightly modified by the pertubation of $\tilde{Q}$. Second, `FAST-QPA` generates feasible iterates in contrast to `KR`. This is very useful for the early pruning.

5.4.3 Reoptimization

At every node of the branch-and-bound tree, we use our algorithm `FAST-QPA` described in Section 5.3 for solving Problem (5.33). A crucial advantage of using an active set method is the possibility of working with warmstarts, i.e., of passing information on the optimal active set from a parent node to its children. In the dual approach the dimension of all subproblems is m, independently of the depth ℓ

in the branch-and-bound tree. When fixing a variable, only the objective function changes, given by $\tilde{Q}$, $\tilde{c}$ and $\tilde{d}$. So as the starting guess in a child node, we choose $\mathcal{A}^0 := \tilde{\mathcal{A}}(\lambda^\star)$, i.e., we use the estimated active set for the optimal solution $\lambda^\star$ of the parent node. We also pass the solution $\lambda^\star$ to the child nodes to initialize the variables in the line search procedure in Algorithm 14, that is we set $\tilde{x}^0_{\mathcal{N}^0} = \lambda^\star_{\mathcal{N}^0}$. Our experimental results presented in Section 5.5 show that this warmstarting approach reduces the average number of iterations of `FAST-QPA` significantly.

5.4.4 Incremental Computations and Preprocessing

Analogously to the ideas of Section 4.3, a remarkable speed-up can be achieved by exploiting the fact that the subproblems enumerated in the branch-and-bound tree are closely related to each other. Let $\ell \in \{0, \dots, n_1 - 1\}$ be the current depth in the branch-and-bound tree and recall that after fixing the first ℓ variables, the problem reduces to the minimization of

$$\bar{f} \colon \mathbb{Z}^{n_1-\ell} \times \mathbb{R}^{n-n_1} \to \mathbb{R},\ x \mapsto x^\top Q_\ell x + \bar{c}^\top x + \bar{d}$$

over the feasible region $\bar{\mathcal{F}} = \{x \in \mathbb{Z}^{n_1-\ell} \times \mathbb{R}^{n-n_1} \mid A_\ell x \leq \bar{b}\}$, where $Q_\ell \succ 0$ is obtained by deleting the corresponding ℓ rows and columns of Q and $\bar{c}$ and $\bar{d}$ are adapted properly by

$$\bar{c}_j := c_j + 2\sum_{i=1}^{\ell} q_{ij} r_i, \text{ for } j = 1, \dots, n - \ell$$

and

$$\bar{d} := d + \sum_{i=1}^{\ell} c_i r_i + \sum_{i=1}^{\ell}\sum_{j=1}^{\ell} q_{ij} r_i r_j,$$

where $r = (r_1, \dots, r_\ell) \in \mathbb{Z}^\ell$ is the current fixing at depth ℓ. Similarly, A_ℓ is obtained by deleting the corresponding ℓ columns of A and the reduced right hand side $\bar{b}$ is updated according to the current fixing.

In our algorithm, we predetermine the order in which variables are fixed. By this, the reduced matrices Q_ℓ, Q_ℓ^{-1} and A_ℓ only depend on the depth ℓ, but not on the specific fixings. Along with the reduced matrix Q_ℓ, the quadratic parts of the reduced dual objective function $\tilde{Q}_\ell$ can then be computed in the preprocessing phase, because they only depend on Q_ℓ and A_ℓ. The predetermined fixing order also allows the computation of the maximum eigenvalues $\lambda_{max}(\tilde{Q}_\ell)$ in the preprocessing phase, needed for ensuring proper convergence of our active set method as described in Section 5.3; compare (5.10).

Concerning the linear part $\tilde{c}$ and the constant part $\tilde{d}$ of the dual reduced problem, both can be computed incrementally in linear time per node: Again, let $r \in \mathbb{Z}^l$

be the current fixing at depth ℓ. By definition of $\tilde{c}$, we have

$$\tilde{c}(r) = \frac{1}{2} A_\ell Q_\ell^{-1} c(r) + b(r) ,$$

where the suffix (r) always denotes the corresponding data after fixing the first ℓ variables to r.

Theorem 5.7. *After a polynomial time preprocessing, the vector $\tilde{c}(r)$ can be constructed incrementally in $O(n - \ell + m)$ time per node.*

Proof. Defining $y(r) := -\frac{1}{2} Q_\ell^{-1} c(r)$, we have

$$\frac{1}{2} A_\ell Q_\ell^{-1} c(r) = -A_\ell y(r).$$

Note that $y(r)$ is the unconstrained continuous minimizer of $f(r)$. In Section 4.3, it was shown in Proposition 4.3 that $y(r)$ can be computed incrementally by

$$y(r) := [y(r') + \alpha z^{\ell-1}]_{1,\dots,n-\ell} \in \mathbb{R}^{n-\ell}$$

for some vector $z^{\ell-1} \in \mathbb{R}^{n-\ell+1}$ and $\alpha := r_\ell - y(r')_\ell \in \mathbb{R}$, where $r' = (r_1, \dots, r_{\ell-1})$ is the fixing at the parent node. This is due to the observation that the continuous minima according to all possible fixings of the next variable lie on a line, for which $z^{\ell-1}$ is the direction. As defined in description of the preprocessing phase in Section 4.3, the vectors $z^{\ell-1}$ only depend on the depth ℓ and can be computed before the enumeration of the nodes. Updating y thus takes $O(n - \ell)$ time. We now have

$$\begin{aligned}
\tilde{c}(r) &= -A_\ell [y(r') + \alpha z^{\ell-1}]_{1,\dots,n-\ell} + b(r) \\
&= -A_\ell [y(r')]_{1,\dots,n-\ell} - \alpha A_\ell [z^{\ell-1}]_{1,\dots,n-\ell} + b(r) \\
&= -(A_{\ell-1} y(r') - y(r')_{n-\ell+1} \cdot A_{\bullet,n-\ell+1}) - \alpha A_\ell [z^{\ell-1}]_{1,\dots,n-\ell} + b(r).
\end{aligned}$$

In the last equation, we used the fact that the first part of the computation can be taken over from the parent node by subtracting column $n - \ell + 1$ of A, scaled by the last component of $y(r')$, from $A_{\ell-1} y(r')$, which takes $O(m)$ time. The second part $A_\ell [z^{\ell-1}]_{1,\dots,n-\ell}$ can again be computed in the preprocessing. The result then follows from the fact that also $b(r)$ can easily be computed incrementally from $b(r')$ in $O(m)$ time. □

Theorem 5.8. *After a polynomial time preprocessing, the scalar $\tilde{d}(r)$ can be constructed incrementally in $O(n - \ell)$ time per node.*

Proof. Recalling that

$$\tilde{d}(r) = \frac{1}{4} c(r)^\top Q_\ell^{-1} c(r) - d(r) ,$$

this follows from the fact that $y(r) = -\frac{1}{2} Q_\ell^{-1} c(r)$ and $c(r)$ can be computed in $O(n - \ell)$ time per node by Proposition 4.3. □

Corollary 4. *After a polynomial time preprocessing, the dual problem* (5.33) *can be constructed in $O(n - \ell + m)$ time per node.*

Besides the effort for solving the QP with the active set method, computing the optimal solution of the primal problem from the dual solution is the most time consuming task in each node. The following observation is used to speed up its computation.

Theorem 5.9. *After a polynomial time preprocessing, the optimal primal solution $x^\star(r)$ can be computed from the optimal dual solution $\lambda^\star(r)$ in $O(m \cdot (n - \ell))$ time per node.*

Proof. From (5.32) we derive

$$\begin{aligned} x^\star(r) &= -\frac{1}{2} Q_\ell^{-1} \left(\sum_{i=1}^{m} \lambda^\star(r)_i a_i + c(r) \right) \\ &= y(r) + \sum_{i=1}^{m} \lambda^\star(r)_i \left(-\frac{1}{2} Q_\ell^{-1} a_i \right) . \end{aligned}$$

The first part can again be computed incrementally in $O(n-\ell)$ time per node. For the second part, we observe that $-\frac{1}{2}Q_\ell^{-1}a_i$ can be computed in the preprocessing phase for all $i = 1, \ldots, m$. □

The above results show that the total running time per node is linear in $n - \ell$ when the number m of constraints is considered a constant and when we make the reasonable assumption that Problem QP is solved in constant time in fixed dimension.

5.5 Experimental Results

We performed extensive numerical experiments to show the effectiveness of the three key components in `FAST-QPA-BB`: duality, warmstarts and fast active set based lower bound computation. To emphasize the stepwise improvements we compared different versions of our branch-and-bound algorithm implemented in C++/Fortran 90 to the MIQP solver of `CPLEX 12.6` on randomly generated instances. We considered the following active set based algorithms, that were used within `FAST-QPA-BB`:

- (`P`) The branch-and-bound algorithm with a standard active set method [162] for solving the QP relaxations
- (`D`) as (`P`) but considering the *dual* problems of the QP relaxations, as described in Section 5.4.2

(`D+w`) as (`D`) but with reoptimization, as described in Section 5.4.3

(`KR`) as (`D+w`) but using the `KR` algorithm as described in Section 5.2

(`QPA`) as (`D+w`) but using the `FAST-QPA` algorithm as described in Section 5.3

Whenever applicable, the incremental computations described in Section 5.4.4 have been used in all approaches. The differences between `D+w`, `KR` and `QPA` consist in the choice of the active set method to solve the node relaxations. All experiments were carried out on Intel Xeon processors running at 2.60 GHz. We used an absolute optimality tolerance and of 10^{-6} for all algorithms. Concerning the feasibility tolerance, we experienced some numerical issues for `FAST-QPA-BB`. If the dual problem is bounded, strong duality guarantees the existence of a primal feasible solution. However, in practice, computing the primal solution by (5.32) can affect its feasibility. This numerical problem is negligible in the pure integer case. On the other hand it is relevant in the mixed-integer case, in the nodes where all integer variables have been fixed, as the optimal solution has to be obtained in one of these nodes. In order to overcome this difficulty we implemented an adaptive feasibility tolerance control. In the mixed-integer case, every time when all integer variables have been fixed, the optimality tolerance of our active set method is iteratively increased until a desired precision for the feasibility of the primal solution is reached. Starting from 10^{-3}, the optimality tolerance of the active set solver is successively divided by 10 until the feasibility tolerance is achieved, stopping however at an optimality tolerance of 10^{-8}. Unfortunately, this was not always sufficient to obtain the desired feasibility accuracy: we were able to guarantee primal feasibility within a tolerance of 10^{-5} only for problems with up to 10 constraints. Hence, for a fair comparison, we set the feasibility tolerance for `FAST-QPA-BB` to 10^{-3} so that primal feasibility was reached for all instances.

Altogether, we randomly generated 1600 different problem instances for CMIQP, considering percentages of integer variables $p := \frac{n_1}{n} \in \{0.25, 0.50, 0.75, 1.0\}$. For $p = 0.25$ we chose $n \in \{50, 100, 150, 200, 250\}$. For $p = 0.5/0.75/1.0$, we chose $n \in \{50, 75, 100, 125, 150\}$ / $\{50, 60, 70, 80, 90\}$ / $\{50, 55, 60, 65, 70\}$, respectively. The number of constraints m was chosen in $\{1, 10, 25, 50\}$. For each combination of p, n and m, we generated 10 instances. For every group of instances with a given percentage of integer variables p, the parameter n was chosen up to a number such that at least one of the tested algorithms was not able to solve all of the 10 instances to optimality for $m = 1$ within our time limit of 3 cpu-hours.

For generating the positive definite matrix Q, we chose n eigenvalues λ_i uniformly at random from $[0, 1]$ and orthonormalized n random vectors v_i, each entry of which was chosen uniformly at random from $[-1, 1]$, then we set $Q = \sum_{i=1}^{n} \lambda_i v_i v_i^\top$. The entries of c were chosen uniformly at random from $[-1, 1]$, moreover we set $d = 0$. For the entries of A and b, we considered two different choices:

(a) the entries of b and A were chosen uniformly at random from $[-1, 1]$,

(b) the entries of A were chosen uniformly at random from $[0, 1]$ and we set $b_i = \frac{1}{2}\sum_{j=1}^{n} a_{ij}$, $i = 1, \ldots, m$.

The constraints of type (b) are commonly used to create hard instances for the knapsack problem. All generated instances are publicly available online [153].

To evaluate the performance of all active set methods, they have been compared to each other on the 200 pure integer instances, i.e., where $p = 1.0$ and $n \in \{50, 55, 60, 65, 70\}$, since they are by far the hardest instances. The results for these instances can be found in the Table 5.1. We compared all active set based methods to see all the improvement steps achieved by the different key components.

From the results we can see that instances with up to 10 linear inequalities and up to 55 variables can be effectively solved to optimality by the dual approach using the tailored active set method, while the simple primal approach as well as `CPLEX 12.6` both suffer from the increasing number of variables. As expected, for larger number of inequalities the running times of our algorithm increase rapidly, it is generally outperformed by `CPLEX 12.6` for $m \geq 10$.

By comparing the running times of (`P`), (`D`), (`D+w`), (`KR`) and (`QPA`), one can observe the stepwise improvement obtained by considering the dual problem, by reoptimization, and by using the special active set method of Kunisch and Rendl. It is also worth to mention that the average number of systems of linear equations needed to be solved at each node stays very small, independently of n and m. Again, a decrease in the number of such systems to be solved is observed in each of the improvement steps.

We can see that the overall running times of our approach are much faster for moderate sizes of m, emphasizing both the quick enumeration process within the branch-and-bound tree and the benefit of using reoptimization.

The dominating algorithm (`QPA`), in the following referred to as `FAST-QPA-BB`, was then also run on all mixed-integer instances and compared to the most recent version of the MIQP solver of `CPLEX 12.6` to further demonstrate its effectiveness. We also tested the branch-and-bound solver `B-BB` of `Bonmin 1.74` on our instances. However, we did not include the running times for the latter into the tables since its performance was not competitive at all, not even for mixed-integer instances with a few number of integer variables. The results for instances of type (a) can be found in Tables 5.2–5.5. We do not include the tables for instances of type (b), since there are no significant differences in the results, except that they are in general easier to solve for our algorithm as well as for `CPLEX 12.6`. All running times are measured in cpu-seconds. The tables include the following data for the comparison between `FAST-QPA-BB` and `CPLEX 12.6`: numbers of instances solved within the time limit, average preprocessing time,

Table 5.1: Comparison between all active set based methods and `CPLEX 12.6` on randomly generated pute integer instances of type (a): For each algorithm, the first column shows the numbers of instances solved within the time limit. The second column shows the average running times in cpu-seconds. All averages are taken over the set of instances solved within the time limit.

inst		Active Set Based Methods											
		P		D		D+w		`KR`		`QPA`		`CPLEX 12.6`	
n	m	#	time	#	time	#	time	#	time	#	time	#	time
50	1	10	926.43	10	43.90	10	30.24	10	9.55	10	6.83	10	53.50
50	10	8	5708.45	9	2151.94	10	864.46	10	225.60	10	83.15	10	189.50
50	25	0	-	0	-	0	-	7	5433.36	9	2337.87	10	2413.50
50	50	0	-	0	-	0	-	0	-	0	-	0	-
55	1	9	4302.08	10	169.54	10	115.58	10	36.17	10	28.47	10	210.71
55	10	1	8214.78	6	4814.54	9	2129.98	10	1192.06	10	427.66	10	1154.91
55	25	0	-	0	-	0	-	2	1183.17	4	3843.30	5	3949.40
55	50	0	-	0	-	0	-	0	-	0	-	0	-
60	1	4	7222.25	10	742.50	10	518.73	10	161.36	10	133.32	9	353.89
60	10	0	-	1	5524.25	4	7405.43	7	3596.24	8	1894.81	7	3477.91
60	25	0	-	0	-	0	-	0	-	2	8963.19	1	6149.31
60	50	0	-	0	-	0	-	0	-	0	-	0	-
65	1	1	3408.13	10	1918.30	10	1388.65	10	438.11	10	349.11	10	2105.36
65	10	0	-	0	-	0	-	6	5486.61	8	4010.42	4	5503.84
65	25	0	-	0	-	0	-	0	-	0	-	0	-
65	50	0	-	0	-	0	-	0	-	0	-	0	-
70	1	0	-	8	4410.96	9	3531.64	10	1347.01	10	1113.47	7	5133.59
70	10	0	-	0	-	0	-	1	9354.72	4	6915.67	0	-
70	25	0	-	0	-	0	-	0	-	0	-	0	-
70	50	0	-	0	-	0	-	0	-	0	-	0	-

average running times, average number of branch-and-bound nodes, average number of iterations of `FAST-QPA` in the root node and average number of iterations of `FAST-QPA` per node in the rest of the enumeration tree. All averages are taken over the set of instances solved within the time limit.

From our experimental results for the pure integer case ($p = 1.0$), we can conclude that most instances with up to ten linear inequalities and up to 65 variables can be effectively solved to optimality by `FAST-QPA-BB`, using the tailored active set method `FAST-QPA`, while `CPLEX 12.6` suffers from an increasing number of variables. As expected, for larger number of inequalities the running times of our algorithm increase rapidly, but `FAST-QPA-BB` nevertheless still clearly outperforms `CPLEX 12.6` for m up to 25 if $p \in \{0.25; 0.50\}$ and is at least competitive to `CPLEX 12.6` if $p = 0.75$. For the mixed-integer case we can see that the average running times of `FAST-QPA-BB` compared to `CPLEX 12.6` are better the bigger the percentage of continuous variables is, even with a larger number of constraints. For the pure integer case we still outperform `CPLEX 12.6` for m up to 10. For all instances, the preprocessing time is negligible.

This experimental study shows that `FAST-QPA-BB` is able to solve 644 instances of type (a) to optimality, while `CPLEX 12.6` can only solve 593 instances. Note that the average number of branch-and-bound nodes in our dual approach is approximately 30 times greater than that needed by `CPLEX 12.6`. Nevertheless the overall running times of our approach are much faster for moderate sizes of m, emphasizing both the quick enumeration process within the branch-and-bound tree and the benefit of using reoptimization. Note that the performance of our approach highly depends on m. As the number of constraints grows, the computational effort for both solving the dual problem and recomputing the primal solution (see Theorem 6.3), is growing as well.

In Table 5.6 we compare the performance of `FAST-QPA-BB` with `FAST-QPA-BB-NP`, a version in which the early pruning is not implemented (see Section 5.4.2). We show the results for the pure integer instances of type (a) with $p = 1.0$. The benefits from the early pruning are evident: the average number of iterations of `FAST-QPA` is decreased leading to faster running times so that 9 more instances can be solved.

Our experimental results also underline the strong performance of `FAST-QPA`. The number of iterations of `FAST-QPA` needed in the root node of our branch-and-bound algorithm is very small on average: for $m = 50$ it is always below 60 and often much smaller. Using warmstarts, the average number of iterations drops to 1–6.

Besides the tables of average running times, we visualized our results by performance profiles in Figure 5.1, as proposed in [68]. They confirm the result that `FAST-QPA-BB` outperforms `CPLEX 12.6` significantly.

Table 5.2: Results for instances of type (a) with $p = 1.0$.

inst		FAST-QPA-BB						CPLEX 12.6		
n	m	#	ptime	time	nodes	it root	it	#	time	nodes
50	1	10	0.00	6.83	9.24e+6	1.50	1.16	10	53.50	5.06e+5
50	10	10	0.00	83.15	3.16e+7	3.80	1.54	10	189.50	1.45e+6
50	25	9	0.01	2337.87	1.62e+8	8.56	2.09	10	2413.50	1.25e+7
50	50	0	-	-	-	-	-	0	-	-
55	1	10	0.00	28.47	3.62e+7	1.60	1.25	10	210.71	1.75e+6
55	10	10	0.00	427.66	1.46e+8	3.40	1.48	10	1154.91	7.53e+6
55	25	4	0.01	3843.30	3.37e+8	7.50	1.86	5	3949.40	1.82e+7
55	50	0	-	-	-	-	-	0	-	-
60	1	10	0.00	133.32	1.60e+8	1.30	1.11	9	353.89	2.61e+6
60	10	8	0.01	1894.81	6.86e+8	2.62	1.51	7	3477.91	2.04e+7
60	25	2	0.02	8963.19	7.55e+8	5.50	2.00	1	6149.31	2.68e+7
60	50	0	-	-	-	-	-	0	-	-
65	1	10	0.01	349.11	4.13e+8	1.40	1.15	10	2105.36	1.31e+7
65	10	8	0.01	4010.42	1.50e+9	2.75	1.48	4	5503.84	2.83e+7
65	25	0	-	-	-	-	-	0	-	-
65	50	0	-	-	-	-	-	0	-	-
70	1	10	0.01	1113.47	1.30e+9	1.60	1.27	7	5133.59	2.88e+7
70	10	4	0.01	6915.67	2.38e+9	2.50	1.51	0	-	-
70	25	0	-	-	-	-	-	0	-	-
70	50	0	-	-	-	-	-	0	-	-

Table 5.3: Results for instances of type (a) with $p = 0.75$.

inst		FAST-QPA-BB						CPLEX 12.6		
n	m	#	ptime	time	nodes	it root	it	#	time	nodes
60	1	10	0.00	1.75	2.01e+6	1.30	1.11	10	16.27	1.22e+5
60	10	10	0.01	5.54	1.82e+6	2.80	1.44	10	22.26	1.33e+5
60	25	10	0.02	30.98	2.36e+6	9.10	1.65	10	34.27	1.53e+5
60	50	10	0.08	580.05	6.64e+6	55.80	2.14	10	161.88	4.83e+5
70	1	10	0.01	11.92	1.30e+7	1.60	1.20	10	105.08	6.63e+5
70	10	10	0.01	26.39	8.19e+6	3.40	1.43	10	101.17	5.10e+5
70	25	10	0.03	153.71	1.23e+7	9.20	1.60	10	241.66	9.10e+5
70	50	10	0.08	2047.06	2.79e+7	18.60	1.96	10	751.75	1.92e+6
80	1	10	0.02	63.38	6.51e+7	1.40	1.12	10	563.33	2.83e+6
80	10	10	0.03	149.43	4.37e+7	3.80	1.42	10	491.01	2.08e+6
80	25	10	0.04	930.12	7.06e+7	10.30	1.60	10	990.54	3.18e+6
80	50	7	0.09	3070.16	5.38e+7	15.29	1.82	9	2379.88	5.22e+6
90	1	10	0.03	413.50	3.79e+8	1.30	1.11	10	3004.33	1.24e+7
90	10	10	0.04	1505.40	4.36e+8	3.00	1.40	10	3504.12	1.25e+7
90	25	10	0.06	4347.03	3.55e+8	5.90	1.55	9	3891.51	1.07e+7
90	50	1	0.09	4883.54	1.02e+8	10.00	1.72	3	4963.04	9.62e+6
100	1	6	0.05	1824.11	1.69e+9	1.33	1.10	4	6703.76	2.31e+7
100	10	6	0.06	5797.23	1.52e+9	4.83	1.39	0	-	-
100	25	1	0.06	4734.57	3.95e+8	5.00	1.51	0	-	-
100	50	1	0.11	3348.44	8.05e+7	15.00	1.74	1	3460.59	5.50e+6

Table 5.4: Results for instances of type (a) with $p = 0.5$.

inst		`FAST-QPA-BB`						`CPLEX 12.6`		
n	m	#	ptime	time	nodes	it root	it	#	time	nodes
50	1	10	0.00	0.01	9.83e+3	1.50	1.16	10	0.24	1.22e+3
50	10	10	0.00	0.02	5.46e+3	3.80	1.48	10	0.15	7.38e+2
50	25	10	0.01	0.08	4.68e+3	8.20	1.71	10	0.23	8.62e+2
50	50	10	0.06	0.98	8.30e+3	21.40	2.35	10	0.65	1.65e+3
75	1	10	0.01	0.20	1.76e+5	1.60	1.21	10	2.71	1.31e+4
75	10	10	0.02	0.64	1.65e+5	3.60	1.42	10	3.02	1.22e+4
75	25	10	0.02	2.14	1.68e+5	9.80	1.57	10	4.45	1.45e+4
75	50	10	0.09	11.13	1.70e+5	17.00	1.88	10	7.72	1.73e+4
100	1	10	0.05	13.48	1.08e+7	1.50	1.15	10	98.43	3.27e+5
100	10	10	0.05	14.37	3.46e+6	4.80	1.39	10	80.06	2.36e+5
100	25	10	0.07	56.13	4.35e+6	8.50	1.53	10	113.76	2.64e+5
100	50	10	0.13	311.71	5.39e+6	51.80	1.72	10	273.26	4.45e+5
125	1	10	0.11	189.76	1.33e+8	1.40	1.13	10	2984.36	6.61e+6
125	10	10	0.13	516.64	1.15e+8	3.40	1.37	10	2850.20	5.67e+6
125	25	10	0.14	1321.35	1.03e+8	8.80	1.46	10	2652.18	4.21e+6
125	50	10	0.21	4344.63	9.28e+7	52.60	1.62	10	3603.98	4.27e+6
150	1	9	0.22	3953.23	2.34e+9	1.78	1.25	2	6389.09	9.71e+6
150	10	6	0.24	6749.47	1.29e+9	3.50	1.38	0	-	-
150	25	3	0.24	9332.23	6.65e+8	7.33	1.46	0	-	-
150	50	0	-	-	-	-	-	0	-	-

Table 5.5: Results for instances of type (a) with $p = 0.25$.

inst		`FAST-QPA-BB`						`CPLEX 12.6`		
n	m	#	ptime	time	nodes	it root	it	#	time	nodes
50	1	10	0.01	0.01	1.30e+2	1.50	1.18	10	0.02	3.43e+1
50	10	10	0.00	0.00	1.96e+2	3.80	1.53	10	0.01	4.52e+1
50	25	10	0.01	0.01	1.40e+2	8.20	1.98	10	0.02	3.87e+1
50	50	10	0.06	0.07	1.14e+2	21.40	3.85	10	0.06	3.22e+1
100	1	10	0.06	0.08	1.05e+4	1.50	1.16	10	0.59	1.03e+3
100	10	10	0.06	0.07	3.88e+3	4.80	1.40	10	0.44	6.37e+2
100	25	10	0.06	0.14	4.91e+3	8.50	1.56	10	0.59	7.32e+2
100	50	10	0.12	0.65	8.24e+3	51.80	1.80	10	1.52	1.61e+3
150	1	10	0.25	0.60	1.68e+5	1.70	1.23	10	9.49	1.32e+4
150	10	10	0.22	0.94	1.24e+5	3.00	1.39	10	8.79	1.09e+5
150	25	10	0.24	2.75	1.56e+5	6.40	1.46	10	15.49	1.65e+5
150	50	10	0.31	4.52	7.77e+4	11.20	1.60	10	11.34	8.84e+3
200	1	10	0.58	14.73	6.38e+6	1.50	1.16	10	332.01	2.59e+5
200	10	10	0.52	21.40	3.00e+6	3.20	1.37	10	297.02	2.17e+5
200	25	10	0.57	74.25	4.03e+6	5.90	1.44	10	414.67	2.55e+5
200	50	10	0.68	388.64	7.44e+6	40.60	1.54	10	982.40	4.84e+5
250	1	10	1.12	326.69	1.21e+8	1.50	1.15	6	3933.41	1.79e+6
250	10	10	1.10	606.83	7.35e+7	3.80	1.35	8	5757.74	2.41e+6
250	25	10	1.17	2105.21	1.01e+8	6.90	1.41	4	4206.78	1.54e+6
250	50	9	1.28	3643.73	6.27e+7	49.11	1.51	4	6371.74	1.92e+5

Table 5.6: Results for instances of type (a) with $p = 1.0$ turning early pruning on/off.

inst		FAST-QPA-BB				FAST-QPA-BB-NP			
n	m	#	time	nodes	it	#	time	nodes	it
50	1	10	6.83	9.24e+6	1.16	10	8.79	9.24e+6	1.48
50	10	10	83.15	3.16e+7	1.54	10	145.95	3.16e+7	2.74
50	25	9	2337.87	1.62e+8	2.09	7	4587.82	1.35e+8	4.25
50	50	0	-	-	-	0	-		-
55	1	10	28.47	3.62e+7	1.25	10	34.86	3.62e+7	1.74
55	10	10	427.66	1.46e+8	1.48	10	743.12	1.46e+8	2.53
55	25	4	3843.30	3.37e+8	1.86	3	4216.79	1.56e+8	3.63
55	50	0	-	-	-	0	-	-	-
60	1	10	133.32	1.60e+8	1.11	10	154.74	1.60e+8	1.34
60	10	8	1894.81	6.86e+8	1.51	8	3259.91	6.86e+8	2.62
60	25	2	8963.19	7.55e+8	2.00	0	-		-
60	50	0	-	-	-	0	-	-	-
65	1	10	349.11	4.13e+8	1.15	10	410.14	4.13e+8	1.47
65	10	8	4010.42	1.50e+9	1.48	7	5238.44	1.23e+9	2.52
65	25	0	-	-	-	0	-	-	-
65	50	0	-	-	-	0	-	-	-
70	1	10	1113.47	1.30e+9	1.27	10	1318.95	1.30e+9	1.81
70	10	4	6915.67	2.38e+9	1.51	1	8714.63	1.31e+9	2.95
70	25	0	-	-	-	0	-	-	-
70	50	0	-	-	-	0	-	-	-

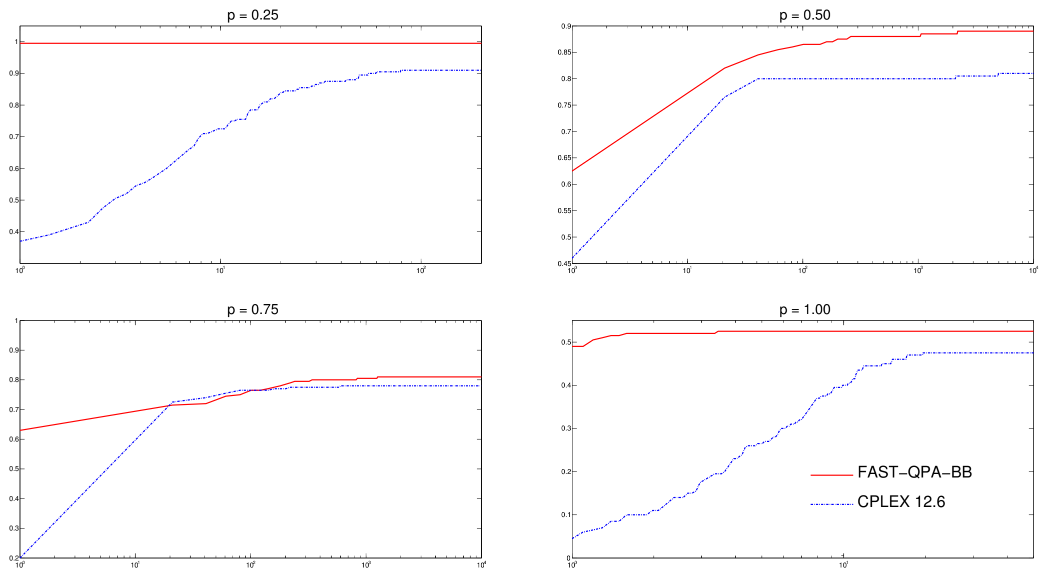

Figure 5.1: Performance profiles for all instances of type (a).

5.6 Conclusion

In this chapter we have shown that the approach of embedding tailored active set methods into a branch-and-bound scheme is very promising for solving convex quadratic integer programming problems of type CMIQP. We also presented a new branch-and-bound algorithm for convex quadratic mixed-integer minimization problems based on the use of an active set method for computing lower bounds. Using dual instead of primal information considerably improves the running times, as it may allow an early pruning of the nodes. Moreover, the dual problem only contains non-negativity constraints, making the problem accessible to our tailored active set method `FAST-QPA`. Our sophisticated rule to estimate the active set leads to a small number of iterations of `FAST-QPA` in the root node that however grows as the number of constraints increases. This shows that for a large number of constraints the QPs addressed by `FAST-QPA` are nontrivial and their solution time has a big impact on the total running time, since we enumerate a large number of nodes in the branch-and-bound tree. Nevertheless, reoptimization helps to reduce the number of iterations of `FAST-QPA` per node substantially, leading to an algorithm that outperforms `CPLEX 12.6` on nearly all problem instances considered.

Chapter 6

Quadratic Outer Approximation

When moving from convex quadratic to general convex nonlinear objective functions, all state-of-the art solvers use at least one of the common methods presented in Section 3.2. As shown by the benchmark results of H. Mittelmann in Section 3.3 the best convex MINLP solver in terms of average running times is `B-OA`, an algorithm in the open-source solver `Bonmin` based on the concept of outer approximation [75] (see Algorithm 8 in Chapter 3). The crucial reason for its success is the availability of advanced and sophisticated ILP solvers that can handle the underlying integer linear subproblems effectively. Since the basic idea of outer approximation is based on linearizations of the objective and constraint functions, the original problem is approximated by linear models, thus in general yielding an inaccurate approximation if the functions are highly nonlinear. Hence, the iterative process of adding linearizations may result in a slow convergence of the algorithm, see [111]. Therefore, we present a new quadratic outer approximation scheme for solving convex box-constrained mixed-integer programs, where suitable quadratic approximations are used to underestimate the objective function instead of classical linear approximations. As a resulting surrogate problem we need to consider the problem of minimizing a function given as the maximum of k finitely many convex quadratic functions having the same Hessian matrix. A fast algorithm for minimizing such functions over integer variables is presented, based on the fast branch-and-bound approach for convex quadratic integer programming proposed by Buchheim, Caprara and Lodi [43], described in Chapter 4. Again, we take over the idea of a fast incremental computation of the continuous global minimum, by implicitly reducing the surrogate problem to $2^k - 1$ convex quadratic integer programs of the form CBMIQP. Hence, each node of the branch-and-bound algorithm can be processed in $O(2^k n)$ time. Experimental results for a special class of ternary and unbounded convex integer programming instances with exponential objective functions are presented. Compared with Bonmin's outer approximation algorithm `B-OA` and branch-and-bound algorithm `B-BB`, running times turn out to be very competitive.

In the further course of this chapter, we are concerned with the minimization of a convex and twice differentiable objective function f over a mixed-integer set subject to box-constraints $l \leq x \leq u$ for some fixed lower and upper bound vectors $l \in (\mathbb{R} \cup \{-\infty\})^n$ and $u \in (\mathbb{R} \cup \{\infty\})^n$. The resulting convex box-constrained integer nonlinear program has the form

$$\begin{aligned} \min\ & f(x) \\ \text{s.t. } & x_i \in \mathbb{Z},\ i = 1, \dots, n_1 \\ & x_i \in \mathbb{R},\ i = n_1 + 1, \dots, n. \\ & l \leq x \leq u, \end{aligned} \tag{CBIP}$$

where $n_1 \in \{0, \dots, n\}$ is the number of integer variables. We will denote the feasible region by $\mathcal{B} := \{x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n-n_1} \mid l \leq x \leq u\}$. It follows directly that CBIP belongs to the class of NP-hard problems, since minimizing a convex quadratic function over the integer lattice is equivalent to the Closest Vector Problem, as shown in Theorem 4.1. This emphasizes also the theoretical and practical importance of CBIP. Our algorithm for solving CBIP that we will present in the following is based on the classical outer approximation scheme.

In Section 6.1, we first give a short recapitulation of the standard linear outer approximation scheme for the case of box-constrained optimization problems. Then, we describe our extension of the latter method to a decomposition approach using quadratic global underestimators. In Section 6.2 we present our approach for solving a convex piecewise quadratic integer program with constant Hessian matrix, which occurs as a surrogate problem in every iteration of our extended outer approximation scheme. In Section 6.3, we present computational results and analyze the effectiveness of the proposed algorithm, applied to a special class of convex integer nonlinear optimization problems. We compare our scheme to the state-of-the-art software package `Bonmin`. Finally, we summarize our main results in Section 6.4 and conclude.

6.1 Linear vs. Quadratic Outer Approximation

6.1.1 Linear Outer Approximation

The main idea of the classical linear outer approximation approach is to equivalently transform the original mixed-integer nonlinear problem into a mixed-integer linear problem by iteratively adding linearizations to the objective function and constraints. Usually the algorithm solves an alternating sequence of mixed-integer linear models and nonlinear models. Since in our problem we have only box-constraints the stopping criterion can be simplified. We have reached optimality as soon as we obtain an iterate that has already been computed in

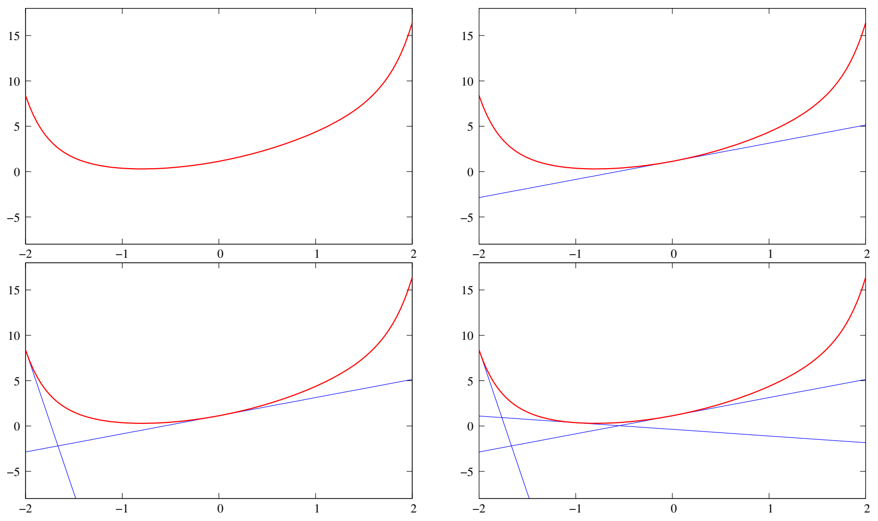

Figure 6.1: Linear outer approximation applied to the minimization of $f\colon \{-2,2\} \to \mathbb{R}$, $f(x) = (x+1)^2 + e^{x^2-2}$. The iterates are $x^1 = 0$, $x^2 = -2$, $x^3 = -1$, and $x^4 = -1$.

an earlier iteration. This is due to the fact that in each iteration we obtain a feasible point and both lower and upper bounds coincide. Applied to CBIP, we get a simplified scheme of the outer approximation introduced in Section 3.2.2. It is sketched in Algorithm 16. The main computational effort of the presented approach lies in the computation of the mixed-integer minimizer in each iteration, by solving a convex box-constrained piecewise linear mixed-integer program, which can be formulated as a mixed-integer linear program. The approach is illustrated in Figure 6.1.

Note that the stopping criterion of the scheme is a valid alternative to the standard criterion where the difference between primal and dual bound goes below a given optimality tolerance.

6.1.2 Quadratic Outer Approximation

The main drawback of linear outer approximation is that in general many iterations are necessary to obtain an appropriate approximation of the original objective function. The basic idea of our approach is to modify Algorithm 16 by replacing the linearizations by appropriate quadratic underestimators. Unfortunately, the second-order Taylor approximation is not necessarily a global underestimator of the original function, as we can see from the simple example in Figure 6.2. One important challenge therefore is to find a suitable quadratic underestimator. The following theorem gives a sufficient condition on the matrix Q of the quadratic function to get a global underestimator for the original function.

Algorithm 16: Simplified linear outer approximation scheme

```
input : convex and continuously differentiable function $f$
output: mixed-integer minimizer $x^\star \in \mathcal{B}$ of $f$
Set $k := 1$ and choose any $x^1 \in \mathcal{B}$.
do
    Compute supporting hyperplane for $f$ in $x^k$:
        $f(x) \geq f(x^k) + \nabla f(x^k)^\top (x - x^k)$.
    Compute $x^{k+1} \in \mathcal{B}$ as an integer minimizer of
        $\max_{i=1,\dots,k} \{f(x^i) + \nabla f(x^i)^\top (x - x^i)\}$.
    Set $k := k + 1$.
while $x^{k+1} \neq x^i$ for all $i \leq k$
Return $x^{k+1}$
```

Theorem 6.1. *Let f be twice continuously differentiable and let $Q \in \mathbb{R}^{n\times n}$ such that $Q \preccurlyeq \nabla^2 f(x)$ for all $x \in [l,u]$, i.e., $Q - \nabla^2 f(x)$ is negative semidefinite for all $x \in [l,u]$. Consider the supporting quadratic function*

$$T(x) := f(x^k) + \nabla f(x^k)^\top (x - x^k) + \tfrac{1}{2}\,(x - x^k)^\top Q (x - x^k).$$

Then $f(x) \geq T(x)$ for all $x \in [l,u]$.

Proof. By construction, we have $\nabla^2(f-T)(x) = \nabla^2 f(x) - Q \succcurlyeq 0$ for all $x \in [l,u]$ and $\nabla(f-T)(x^k) = \nabla f(x^k) - \nabla f(x^k) = 0$. This implies that $f - T$ is convex with minimizer x^k, yielding $f(x) - T(x) \geq f(x^k) - T(x^k) = 0$ for all $x \in [l,u]$. □

Unfortunately the existence of such a quadratic underestimator is not always guaranteed. We have the following necessary and sufficient condition. Note that strict convexity is not sufficient as for example the exponential function does not allow any quadratic underestimator with a non-zero matrix Q.

Observation 6.2. *Let f be a twice continuously differentiable strictly convex function. There exists $0 \neq Q \in \mathbb{R}^{n\times n}$ such that $Q \preccurlyeq \nabla^2 f(x)$ for all $x \in [l,u]$ if and only if $\inf_x \lambda_{\min}(x) > 0$ for all $x \in [l,u]$, where $\lambda_{\min}(x)$ denotes the minimum eigenvalue of $\nabla^2 f(x)$.*

Proof. Let $\bar\lambda := \inf_{x\in[l,u]} \lambda_{\min}(x) > 0$. We can choose $0 \neq Q := diag(\bar\lambda) \cdot I \succcurlyeq 0$ and we further have that $\nabla^2 f(x) - Q \succcurlyeq 0$ for all $x \in [l,u]$. The other direction is obviously true. □

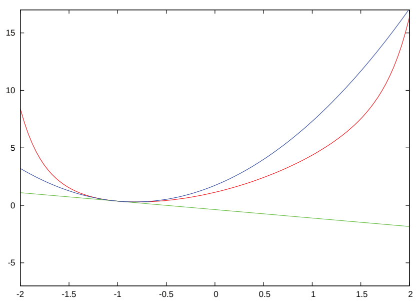

Figure 6.2: The first and second order Taylor Approximation $T_1(x,-1)$ (green) and $T_2(x,-1)$ (blue) for the function $f(x) = (x+1)^2 + e^{x^2-2}$ (red).

Algorithm 17: Quadratic outer approximation scheme

```
input  : convex and twice continuously differentiable function $f$,
         matrix $Q$ s.t. $0 \preccurlyeq Q \preccurlyeq \nabla^2 f(x)$ for all $x \in [a,b]$
output: mixed-integer minimizer $x^\star \in \mathcal{B}$ of $f$
Set $k := 1$ and choose any $x^1 \in \mathcal{B}$.
do
    Compute supporting quadratic underestimator for $f$ in $x^k$:
        $f(x) \geq f(x^k) + \nabla f(x^k)^\top (x - x^k) + \frac{1}{2}(x - x^k)^\top Q (x - x^k).$
    Compute $x^{k+1} \in \mathcal{B}$ as a mixed-integer minimizer of
        $\max_{i=1,\dots,k} \{ f(x^i) + \nabla f(x^i)^\top (x - x^i) + \frac{1}{2}(x - x^i)^\top Q (x - x^i) \}.$
    Set $k := k + 1$.
while $x^{k+1} \neq x_i$ for all $i \leq k$
Return $x^{k+1}$
```

The entire quadratic outer approximation scheme is sketched in Algorithm 17.

In Algorithm 17 the new underestimator for a given iterate x^k is computed as follows:

$$\begin{aligned} & f(x^k) + \nabla f(x^k)^\top (x - x^k) + \tfrac{1}{2}(x - x^k)^\top Q (x - x^k) \\ = & \underbrace{f(x^k) - \nabla f(x^k)^\top x^k + \tfrac{1}{2} {x^k}^\top Q x^k}_{=:d_{k+1}} + \underbrace{\left(\nabla f(x^k)^\top - {x^k}^\top Q\right)}_{=:c_{k+1}^\top} x + \tfrac{1}{2} x^\top Q x \,, \end{aligned}$$

so that

$$g_{k+1}(x) := \tfrac{1}{2} x^\top Q x + c_{k+1}^\top x + d_{k+1}$$

is a convex quadratic function with Hessian $Q \succcurlyeq 0$ not depending on k.

Algorithm 17 requires to compute a mixed-integer minimizer of a quadratic program instead of a linear program, as was the case in Algorithm 16. Although the hardness of this surrogate problem increases from a practical point of view, this approach might pay off if the number of iterations decreases significantly with respect to linear approximation. In fact, we observed in our experiments that the number of iterations stays very small in general, even for problems in higher dimensions; see Section 6.3.

However, the surrogate problem of solving a convex box-constrained piecewise quadratic mixed-integer program, being the most expensive part of Algorithm 17, requires an effective solution method to keep the whole algorithm fast. The surrogate problem can be formulated as a minimization problem of the form

$$\min_{x\in\mathcal{B}} \max_{i=1,\dots,k} \left(\tfrac{1}{2}\, x^\top Q x + c_i^\top x + d_i\right), \tag{6.1}$$

where $Q \succcurlyeq 0$, $c_1,\dots,c_k \in \mathbb{R}^n$, and $d_1,\dots,d_k \in \mathbb{R}$. At this point it is crucial to emphasize that the Hessian Q of each quadratic function $g_k(x)$ is the same, so that we can rewrite (6.1) as

$$\min_{x\in\mathcal{B}} \left(\tfrac{1}{2}\, x^\top Q x + \max_{i=1,\dots,k} (c_i^\top x + d_i)\right),$$

which in turn can be reformulated as a linearly constrained mixed-integer quadratic program using an auxiliary variable $\alpha \in \mathbb{R}$:

$$\begin{aligned} \min\ & x^\top Q x + \alpha \\ \text{s.t.}\ & c_i^\top x + d_i \le \alpha \quad \forall i = 1,\dots,k \\ & x \in \mathcal{B} \\ & \alpha \in \mathbb{R}. \end{aligned} \tag{6.2}$$

Note that the reformulated problem (6.2) is only convex in (x,α), although it is strictly convex in x. For this reason, it cannot be solved directly by `FAST-QPA-BB` from Chapter 5. A small example for the quadratic outer approximation scheme is illustrated in Figure 6.3. In this small example, the algorithm terminates after two iterations.

6.2 Computing Lower Bounds

Solving the convex piecewise quadratic mixed-integer program (6.1) in each iteration is the core task of the quadratic outer approximation scheme. We implicitly reduce this problem, in iteration k, to at most $2^k - 1$ convex quadratic mixed-integer programs, which are solved by Algorithm 12 (see Section 4.2). Our computational results in Section 6.3 show that only few iterations of Algorithm 17 are

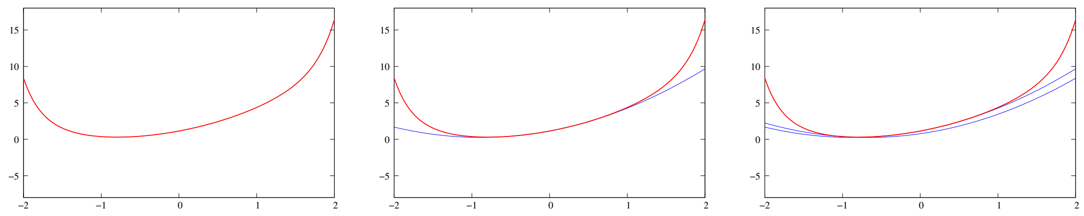

Figure 6.3: Quadratic outer approximation applied to the minimization of $f\colon \{-2,2\} \to \mathbb{R}$, $f(x) = (x+1)^2 + e^{x^2-2}$, using $Q = 2 + 2e^{-2}$. The iterates are $x^1 = 0$, $x^2 = -1$, and $x^3 = -1$.

necessary in general to solve an instance to optimality, so that the number $2^k - 1$, though being exponential in the number of iterations k, remains reasonably small in practice.

We recall that after fixing the first ℓ variables, every quadratic function f_i reduces to a function

$$\bar{f}_i\colon \mathbb{Z}^{n-\ell} \to \mathbb{R},\ x \mapsto \tfrac{1}{2}\, x^\top Q_\ell x + \bar{c}_i^\top x + \bar{d}_i$$

where $Q_\ell \succcurlyeq 0$ is obtained by deleting all rows and columns of Q corresponding to fixed variables and $\bar{c}_i$ and $\bar{d}_i$ are adapted appropriately. To solve the surrogate problem (6.1) with a branch-and-bound algorithm, we compute a lower bound at every node by solving its continuous relaxation

$$\min_{x\in\mathbb{R}^{n-\ell}} \max_{i=1,\dots,k} \left(\tfrac{1}{2}\, x^\top Q_\ell x + \bar{c}_i^\top x + \bar{d}_i\right) .$$

The main idea is to decompose this problem into subproblems, namely the minimization of several auxiliary quadratic functions defined on affine subspaces of $\mathbb{R}^{n-\ell}$, and finally make use of the incremental computation technique described in the last subsection. To describe this decomposition procedure, we need to introduce some definitions. First, we define

$$\bar{f}(x) := \max_{i=1,\dots,k} \bar{f}_i(x)$$

as the maximum of the reduced functions

$$\bar{f}_i(x) := \tfrac{1}{2}\, x^\top Q_\ell x + \bar{c}_i^\top x + \bar{d}_i,\ i = 1,\dots,k .$$

For all $J \subseteq \{1,\dots,k\}$, $J \neq \emptyset$, we define

$$U_J := \{x \in \mathbb{R}^{n-\ell} \mid \bar{f}_i(x) = \bar{f}_j(x)\ \forall i,j \in J\}$$

and consider the auxiliary function

$$\bar{f}_J(x) : U_J \to \mathbb{R}, \quad \bar{f}_J(x) := \bar{f}_i(x),\ i \in J .$$

As all functions $\bar{f}_i$ have the same Hessian matrix Q_ℓ, each set U_J is an affine subspace of $\mathbb{R}^{n-\ell}$. In particular, we can compute the minimizers $x_J^\star$ of all $\bar{f}_J$ incrementally as described in Chapter 4 by adapting the initial lower bound computations in the preprocessing for every subspace U_J. This can be done by solving the corresponding KKT-system. In our case the restriction to an affine subspace can be expressed as a linear equation and the KKT-system is a system of linear equations. Hence, the polynomial running time in the preprocessing is maintained.

The following theorem gives a condition that allows to exclude certain candidates.

Theorem 6.3. *For each $J \subseteq \{1, \dots, k\}$ with $J \neq \emptyset$, let $x_J^\star$ be a minimizer of $\bar{f}_J$ over U_J. Then the global minimum of $\bar{f}$ is*

$$\min \{\bar{f}_J(x_J^\star) \mid \emptyset \neq J \subseteq \{1, \dots, k\},\ \bar{f}(x_J^\star) = \bar{f}_J(x_J^\star)\}.$$

Proof. Clearly "$\leq$" holds. To show "$\geq$", let $x^\star \in \mathbb{R}^{n-\ell}$ be the global minimizer of $\bar{f}$ and define

$$J^\star := \{i \mid \bar{f}_i(x^\star) = \bar{f}(x^\star)\} \neq \emptyset.$$

Then it follows that $x^\star \in U_{J^\star}$ and $\bar{f}(x^\star) = \bar{f}_{J^\star}(x^\star)$. Moreover, $x^\star$ minimizes $\bar{f}_{J^\star}$ over $U_{J^\star}$, hence $x^\star = x_{J^\star}^\star$ by strict convexity. In summary,

$$\bar{f}(x_{J^\star}^\star) = \bar{f}(x^\star) = \bar{f}_{J^\star}(x^\star) = \bar{f}_{J^\star}(x_{J^\star}^\star)\ ,$$

from which the result follows. □

Corollary 5. *The running time of the modified branch-and-bound algorithm for solving the surrogate problem* (6.1) *is $O(2^k \cdot (n - \ell))$ per node.*

Proof. The cardinality of the power set of J is 2^k. Since for each subset the corresponding continuous minimizer can be computed in $O(n-\ell)$ time per node, the statement follows immediately. □

Note that the index set J does not need to be considered any more in the current node and its branch-and-bound subtree as soon as the value of $\bar{f}_J(x_J^\star)$ exceeds the current upper bound.

Example 6.1. Consider the special case of $k = 2$. Then we have the following reduced functions:

$$\bar{f}_{1/2} \colon \mathbb{R}^{n-\ell} \to \mathbb{R},\ x \mapsto x^\top Q_\ell x + \bar{c}_{1/2}^\top x + \bar{d}_{1/2}.$$

It follows from the definition of the subpaces that $U_{\{1\}} = U_{\{2\}} = \mathbb{R}^{n-\ell}$ and $U_{\{1,2\}} = \{x \in \mathbb{R}^{n-\ell} \mid \bar{f}_2(x) = \bar{f}_1(x)\} = \{x \in \mathbb{R}^{n-\ell} \mid (\bar{c}_2 - \bar{c}_1)^\top x = \bar{d}_2 - \bar{d}_1\}$.

Following our idea, we incrementally compute the minimizers $\bar{x}_{\{1\}}$ of $\bar{f}_1$ on $\mathbb{R}^{n-\ell}$, $\bar{x}_{\{2\}}$ of $\bar{f}_2$ on $\mathbb{R}^{n-\ell}$ and $\bar{x}_{\{1,2\}}$ of $\bar{f}_1$ (or $\bar{f}_2$) on $U_{\{1,2\}}$. According to Theorem 6.3, we check if $\bar{f}_1(x^\star_{\{1\}}) = \bar{f}(x^\star_{\{1\}})$. Since $k = 2$, the condition $\bar{f}(x^\star_J) = \bar{f}_J(x^\star_J)$ is true for exactly one subset J. In this case, when $\bar{f}_1(x^\star_{\{1\}}) \geq \bar{f}_2(x^\star_{\{1\}})$ we have that $x^\star_{\{1\}}$ is the global minimizer, in case $\bar{f}_2(x^\star_{\{2\}}) \geq \bar{f}_1(x^\star_{\{2\}})$ we have that $x^\star_{\{2\}}$ is the global minimizer. Otherwise we know that the minimizer is $x^\star_{\{1,2\}}$. $\square$

6.3 Experimental Results

To show the potential of our approach, we carried out two types of experiments. First, we compared our branch-and-bound algorithm for solving the surrogate problems (6.1), called `CPQIP`, to `CPLEX 12.4`. Second, we created a class of hard convex integer programs, to illustrate the effectiveness of our quadratic outer approximation algorithm, called `QOA`, and to compare its performance to that of `B-OA` and `B-BB 1.5.1` [37] using `Cbc 2.7.2` and `Ipopt 3.10`. All experiments were carried out on Intel Xeon E5-2640 processors at 2.50 GHz.

6.3.1 Implementation Details

We implemented our algorithm in C++. To speed up our algorithm, we used a straightforward local search heuristic to determine a good starting point. We start by taking the origin $x = (0, \ldots, 0)$ and continue to increase the first variable x_1 until no further improvement can be found. In the same way we test if decreasing the variable leads to a better solution. We repeat this procedure for every consecutive variable $x_2, \ldots, x_n$.

Another small improvement in running time is achieved by using the optimal solution $x^\star_k$ of the surrogate problem in iteration k to get an improved global upper bound UB for the next iteration, i.e.,

$$UB := \max_{i=1,\ldots,k+1} \tfrac{1}{2}\, {x^\star_k}^\top Q x^\star_k + c_i^\top x^\star_k + d.$$

6.3.2 Surrogate Problem

We randomly generated 160 instances for the surrogate problem (6.1), 10 for each combination of $n \in \{20, 30, 40, 50\}$ and $k \in \{2, 3, 4, 5\}$. We chose $\mathcal{B} = \mathbb{Z}^n$, i.e., we consider unbounded instances. Similar to Chapter 5, we used a random generator to build our instances. For the positive semidefinite matrix Q, we chose n eigenvalues λ_i uniformly at random from $[0, 1]$ and orthonormalized n random vectors v_i, where all entries were chosen uniformly at random from $[-10, 10]$, then we set $Q = \sum_{i=1}^{n} \lambda_i v_i v_i^\top$. The entries of all c_i and d_i, $i = 1, \dots, k$, were chosen uniformly at random from $[-10, 10]$.

We compared our algorithm `CPQIP` with the MIQP solver of `CPLEX` applied to Problem (6.2). As already mentioned in Section 6.1.2, it would be intuitive to use also `FAST-QPA-BB` from Chapter 5 for a comparison. However, since Problem (6.2) is not strictly convex, `FAST-QPA-BB` is not able to handle our instances directly. Numerically, this problem can be overcome by adding a small value $\varepsilon > 0$ to the diagonal element of Q corresponding to the linearization variable α to make the problem become strictly convex again. Even if this trick does not allow a completely fair comparison, we include the results of `FAST-QPA-BB` on the slightly perturbed instances to give an impression on how well the approach of `CPQIP` works. To make the problems strictly convex, we chose ε to be 10^{-8}. For all three algorithms, we used a time limit of 3 hours, an absolute optimality tolerance of 10^{-6} and a relative optimality tolerance of 10^{-10}.

The results are summarized in Table 6.1. Running times are measured in cpu-seconds and the numbers of nodes explored in the branch-and-bound trees are given in the corresponding column. First, we can observe that `CPQIP` could solve 158 out of 160 instances in total within the given time limit, while `CPLEX` and `FAST-QPA-BB` managed to solve only 154 and 150 instances, respectively. Second, for instances with a large number n of variables but small k, `CPQIP` turns out to be significantly faster than `CPLEX` but also `FAST-QPA-BB`. Instances of this type are the most relevant instances in a quadratic outer approximation scheme, as shown in Section 6.3.3. The results confirm that the decomposition approach used in `CPQIP` works well and in particular takes into account the specific problem structure of Problem (6.1).

As expected, `CPQIP` tends to be slower than `CPLEX` on most of the instances with $k = 5$, since the running time per node in our branch-and-bound algorithm is exponential in k.

6.3.3 Quadratic Outer Approximation

In order to evaluate the entire quadratic outer approximation scheme, we consider the following class of problems:

$$\min_{x \in \mathcal{B}} f(x), \; f : \mathcal{B} \subseteq \mathbb{R}^n \to \mathbb{R}, \; f(x) = \sum_{i=1}^{n} \exp(q_i(x)) \tag{6.3}$$

where $q_i(x) = x^\top Q_i x + c_i^\top x + d_i$. We assume $Q_i \succ 0$ for all $i = 1, \dots n$, so that f is a strictly convex function.

In order to determine a feasible matrix Q according to Theorem 6.1, we first

Table 6.1: The table shows the average running times, the numbers of instances solved and average numbers of branch-and-bound nodes for `CPQIP`, `FAST-QPA-BB` and `CPLEX 12.4` on randomly generated unbounded instances for the surrogate problem of type (6.1).

inst		`CPQIP`			`FAST-QPA-BB`			`CPLEX 12.4`		
n	k	#	time	nodes	#	time	nodes	#	time	nodes
20	2	10	0.02	1.4e+4	10	8.56	4.2e+4	10	0.24	2.5e+3
20	3	10	0.03	9.8e+3	10	4.77	2.5e+4	10	0.18	1.7e+3
20	4	10	0.06	9.5e+3	10	5.98	2.8e+4	10	0.21	1.6e+3
20	5	10	0.05	2.9e+3	10	0.22	7.4e+4	10	0.10	1.1e+3
30	2	10	0.07	4.2e+4	10	18.97	1.3e+5	10	0.90	1.3e+4
30	3	10	0.40	1.1e+5	10	67.56	3.3e+5	10	1.94	2.8e+4
30	4	10	4.48	4.2e+5	10	246.44	1.2e+6	10	7.75	9.1e+4
30	5	10	4.57	2.6e+5	10	30.11	7.6e+5	10	2.90	3.9e+5
40	2	10	10.41	5.4e+6	10	2206.24	1.6e+7	10	138.77	1.3e+6
40	3	10	30.70	7.2e+6	10	2449.90	2.1e+7	10	69.58	7.9e+5
40	4	10	66.13	7.2e+6	9	1144.60	1.5e+7	10	73.32	8.3e+5
40	5	10	141.53	7.2e+6	10	211.83	7.2e+6	10	92.80	1.0e+6
50	2	10	320.30	1.5e+8	7	2954.61	1.3+e8	10	2337.91	1.8e+8
50	3	10	844.18	1.7e+8	8	4528.46	1.6+e8	8	646.15	5.4e+7
50	4	10	1925.06	1.8e+8	8	5696.67	1.7e+8	8	864.92	7.0e+7
50	5	8	1650.26	6.9e+7	8	4999.94	2.0e+8	8	931.00	7.5e+7

compute $m_i := \min_{x\in\mathcal{B}} q_i(x)$ for all $i = 1, \dots, n$ and then choose

$$Q := \sum_{i=1}^{n} 2Q_i \exp(m_i) \succ 0 \ .$$

We can easily verify that Q can be used in Algorithm 17.

Observation 6.4. *With the choices above, we have $\nabla^2 f(x) - Q \succcurlyeq 0$ for all $x \in \mathbb{R}^n$.*

Proof. We have

$$\nabla f(x) = \sum_{i=1}^{n} \exp(q_i(x))(2Q_i x + c_i)$$

and

$$\nabla^2 f(x) = \sum_{i=1}^{n} (2Q_i x + c_i)(2Q_i x + c_i)^\top \exp(q_i(x)) + 2Q_i \exp(q_i(x)).$$

Table 6.2: The table shows the average running times, numbers of instances solved and the average and maximum numbers of iterations of `QOA` compared to `B-OA` and `B-BB 1.5.1` for randomly generated ternary instances of problem type (6.3).

QOA					B-OA		B-BB 1.5.1	
n	#	it	max	time (s)	#	time (s)	#	time (s)
20	10	1.00	1	0.03	10	91.53	10	2.12
30	10	1.50	4	5.87	5	1017.99	10	148.85
40	10	1.40	3	20.19	0	-	9	4573.80
50	10	2.00	3	69.54	0	-	0	-
60	9	2.11	4	1154.66	0	-	0	-
70	5	3.80	6	3363.11	0	-	0	-

We define

$$\tilde{Q}_i := (2Q_i x + c_i)(2Q_i x + c_i)^\top \exp(q_i(x)) \succcurlyeq 0$$

and

$$Q := \sum_{i=1}^{n} 2Q_i \exp(m_i) \succ 0 \;\forall x \in \mathbb{R}^n.$$

Then

$$\nabla^2 f(x) - Q = \sum_{i=1}^{n} \tilde{Q}_i + 2Q_i(\exp(q_i(x)) - \exp(m_i)) \succcurlyeq 0$$

since $\tilde{Q}_i, Q_i \succcurlyeq 0 \;\forall i = 1, \dots, n$ and the set of positive semidefinite matrices is a cone. □

The quality of Q and therefore of the quadratic underestimator strongly depends on $\mathcal{B}$: the smaller the set $\mathcal{B}$ is, the larger are the values of m_i, and the better is the global underestimator.

To test the quadratic outer approximation scheme, we randomly generated ternary and unbounded instances of type (6.3), i.e., we consider the box $\mathcal{B} = [-1, 1]^n$ and the unconstrained case $\mathcal{B} = \mathbb{R}^n$. All data were generated in the same way as in Section 6.3.2, except that all coefficients of Q and $c_i, d_i,\ i = 1, \dots, n$, were scaled by 10^{-5}, to avoid numerical issues in the function evaluations. This is due to the effect of the exponential function that leads to very large objective function values, even on a small domain around the origin.

We tested our algorithm against `B-OA` and `B-BB`, which are state-of-the-art solvers for convex mixed-integer nonlinear programming. While `B-OA` is a decomposition approach based on outer approximation, `B-BB` is a simple branch-and-bound algorithm based on solving a continuous nonlinear program at each node of the search tree and branching on the integer variables. Again the time limit per instance was set to 3 hours.

Table 6.3: The table shows the average running times, the numbers of instances solved and the average and maximum numbers of iterations of `QOA` compared to `B-BB 1.5.1` for randomly generated unbounded instances of problem type (6.3).

	QOA				B-BB 1.5.1	
n	#	it	max	time (s)	#	time (s)
20	10	2.30	3	0.12	10	113.13
30	8	3.12	5	26.38	1	4897.10
40	6	2.83	3	134.42	0	-
50	2	3.00	3	7630.12	0	-

Results for ternary and unbounded instances are shown in Tables 6.2 and 6.3, respectively. `QOA` denotes our quadratic approximation scheme using the local search heuristic. Running times are again measured in seconds. The columns named "#","it" and"max" show the number of instances solved, the average and maximum number of iterations in our approach, respectively.

Overall our quadratic outer approximation approach could solve more instances and turns out to be considerably faster for the ternary instances as well as the unbounded instances. For the unbounded instances, we unfortunately experienced some numerical issues with `B-OA` so that it did not converge properly. The reason might be the resulting big values in the function evaluations. However, the branch-and-bound solver `B-BB` was clearly outperforming `B-OA` on these instances anyway, so that we included only the running times of `B-BB` in the corresponding table. In the ternary case, such problems did not occur.

An important observation in all our experiments is that both the average and maximum number of iterations in our outer approximation scheme tend to be small, here up to 4 for the instance sizes we could solve to optimality within the given time limit. In particular, the number of iterations does not seem to increase significantly with the number of variables n, contrary to the standard linear outer approximation approach.

6.4 Conclusion

In this chapter we proposed a quadratic outer approximation scheme for solving box-constrained convex integer nonlinear programs, based on the classical linear outer approximation scheme. From our computational results, we can conclude that quadratic underestimators have the potential to yield significantly better approximations, which might lead to considerably fewer iterations of the entire algorithm. While the standard linear outer approximation scheme requires to solve an integer linear program in each iteration, our method requires the solution of integer quadratic programs with linear constraints. Therefore we proposed an algorithm which is based on the reduction of the surrogate problems to a set of box-constrained/unconstrained convex quadratic integer programs, which are effectively solved by Algorithm 12. For a special class of instances with exponential objective function, we found out that the idea of improving the approximation by quadratic underestimators pays off in terms of a small number of iterations, growing only slightly with the dimension n of the problems, and therefore results in better running times.

Chapter 7

Frank-Wolfe Based Branch-and-Bound

In this chapter we see how branch-and-bound algorithms can be used for real-world problems. We present a tailored algorithm for the risk-averse capital budgeting problem, a special application in portfolio optimization. In numerical experiments on real-world data, we show that it may help an investor to make decisions on building his portfolio depending on his own personal risk profile.

In finance, mathematical programming approaches are widely used to solve mean-variance optimization (MVO) problems based on the models and methods developed by Harry Markowitz in his seminal paper in 1952 [151]. In MVO we assume an investor to hold a certain amount of money that he is willing to invest into a given number of securities (e.g., stocks, bonds, ...) that have uncertain returns. For each security the expected return and variance is given. Following Markowitz's principle of diversification an investor is interested in maximizing a risk-adjusted expected return, defined as the expected return minus a scaled proportion of the variance, trying to find a trade-off between expected returns and the market volatility, i.e., the risk of the portfolios. Note that this is one way to model MVO problems, another one is to maximize the expected return of the portfolio while limiting the variance of its return, yielding a nonlinear constraint. We refer to the book of Cornuéjols and Tütüncü [56] for a more detailed description of mathematical programming models for MVO and their comparison. The general setting that we consider is the following:

We are given an investor, having a budget $b > 0$ and a number of securities $S_1, \dots, S_n$ ($n \geq 2$), where security S_i has an expected return of r_i, a standard deviation σ_i of the return and a cost of $a_i > 0$ per unit. Furthermore, let $M = (\sigma_{ij}) \in \mathbb{R}^{n \times n}$ be the symmetric positive semidefinite covariance matrix with $\sigma_{ii} = \sigma_i^2$ and $\sigma_{ij} = \rho_{ij}\sigma_i\sigma_j$, for $i \neq j$. Here, ρ_{ij} denotes the correlation coefficient of the returns of S_i and S_j for $i \neq j$, and $\rho_{ii} := 1$ for all $i = 1, \dots, n$. Sometimes short-selling, i.e., the sale of a security that is not owned by the seller, or that the seller has borrowed, is not prefered. If we denote by y_i the total amount invested in security S_i, these restrictions can be expressed as non-negativity constraints on the variables y_i. A popular special case of modeling mean-variance optimization problems is the so-called *risk-averse capital budgeting problem* using binary variables:

$$z := \max \left\{ \sum_{i \in I} r_i y_i - \Omega \sqrt{\sum_{i \in I} \sigma_i^2 y_i} : \sum_{i \in I} a_i y_i \leq b,\ y \in \{0,1\}^I \right\}, \tag{7.1}$$

where $I := \{1, \ldots, n\}$ is the set of securities and the constant Ω serves as a parameter that models risk-aversion. In this model covariances are not taken into account, i.e., $\sigma_{ij} = 0$, for all $i \neq j$, yielding a diagonal matrix M. Bertsimas and Popescu [27] suggested to choose $\Omega = \sqrt{\frac{1-\varepsilon}{\varepsilon}}$, yielding a variant of the classical value-at-risk (VaR) problem [30] that takes into account uncertainties in the expected returns. This problem was studied by several authors before. Atamtürk and Narayanan [11] proposed a SOCP-based branch-and-bound algorithm. In [16] Baumann et al. [16] used techniques from Lagrangian decomposition while in [15] Baumenn et al. used a branch-and-cut approach to solve it. In (7.1) binary variables are used to decide, whether the investor should invest in security S_i or not. Several other authors studied Problem (7.1) in the context of different optimization problems [51, 114, 163, 179, 185]. Other exact approaches for problems including mean-variance portfolio optimization problems are devised by [28, 29, 39, 84, 135, 140, 178]. Mostly, versions of the *Limited Asset Markowitz* model are considered, where the number of assets to invest money into is limited by some upper bound to reduce transaction costs. Usually an MIQP model is used to solve it.

In our model, we want to generalize (7.1) in three ways:

1. Besides binary variables it is convenient to consider also general non-negative integer variables, motivated by the fact that portfolios are often not only restricted to binary but also general discrete choices in practice, for example when considering stocks.

2. The parameter Ω changes the investor's risk-aversion only linearly in the risk-term $\sqrt{y^\top M y}$. This might be too restrictive, since an investor's risk-aversion may vary. We propose to model the risk by the composite function $h \circ g : \mathbb{R}^n \to \mathbb{R}$, where $g : \mathbb{R}^n \to \mathbb{R}$, $g(y) = \sqrt{y^\top Q y}$ models the quantity of risk and $h : \mathbb{R}^+ \to \mathbb{R}$ is a convex, differentiable and non-decreasing function, adjusting the risk-aversion of the investor. The objective function can then be written as $r^\top y - h(g(y))$. Possible choices for h are $h(t) = \Omega t$, yielding

the formulation (7.1), $h(t) = \Omega t^2$, which gives a convex MIQP problem or

$$h(t) = \tilde{h}(t) := \begin{cases} 0 & , \ t \leq \gamma \\ \exp(t-\gamma) - (t-\gamma+1) & , \ t > \gamma, \end{cases}$$

such that the investor's risk-aversion increases exponentially in the growth of the risk as soon as it reaches a threshold of γ.

3. The majority of approaches considered in the literature requires M to be diagonal, meaning that covariances are ignored. They explicitly exploit the fact that the objective function is submodular in that case.

To tackle the proposed *generalized risk-averse capital budgeting problem* with mixed-integer variables, we consider the optimization problem of the following form:

$$z := \max \left\{ r^\top y - h(\sqrt{y^\top M y}) : a^\top y \leq b, \ y \geq 0, \ y_1, \dots, y_{n_1} \in \mathbb{Z} \right\}, \tag{7.2}$$

where $h : \mathbb{R}^+ \to \mathbb{R}$ is a convex differentiable function, $M \in \mathbb{R}^{n \times n}$ is a symmetric positive definite matrix, $a, r \in \mathbb{R}^n_+$, $b \in \mathbb{R}^+$, and $n_1 \in \{0, \dots, n\}$ is the number of integer variables. Note that assuming M to be positive definite is not a restriction in practice, since it is equivalent to having no redundancies in the set of available securities. Problem (7.2) can then be turned into a convex mixed-integer minimization problem of the form

$$z := -\min \left\{ h(\sqrt{y^\top M y}) - r^\top y : a^\top y \leq b, \ y \geq 0, \ y_1, \dots, y_{n_1} \in \mathbb{Z} \right\}. \tag{GCBP}$$

The remainder of this chapter is organized as follows. In Section 7.1 we describe our modified Frank-Wolfe algorithm to efficiently compute the dual bounds for the node relaxations. This section also includes an in-depth convergence analysis of our algorithm. In Section 7.2 we shortly explain the small adpations of our branch-and-bound algorithm (Algorithm 12 in Chapter 4) to our Problem GCBP. In particular, we present several effective warmstart strategies to accelerate the dual bound computation. In Section 7.3 we test our algorithm on real-world instances. We show computational results and compare the performances of our algorithm and `CPLEX 12.6` for different risk-functions h. In case $h(t) = t^2$ we also compared our algorithm with `Bonmin 1.8.1`. Finally, in Section 7.4 we summarize the results and give a conclusion.

7.1 Dual Bounds by the Frank-Wolfe Method

In this section, we describe the non-monotone version of the Frank-Wolfe algorithm with away-steps, originally proposed in [104]. We embed this algorithm into our branch-and-bound framework and fully analyze its convergence properties. Using a Frank-Wolfe type method in the context of convex mixed-integer nonlinear programming is twofold. On the one hand, the algorithm gives a valid dual bound for the original MINLP problem at each iteration. This dual bound is available for free as a byproduct of computing the descent direction in each iteration. Thus, it enables an early pruning of the nodes in the branch-and-bound tree (see Section 5.4.2). On the other hand, the cost per iteration is very low, as in each iteration a simple LP with a special structure is solved. The original method described in [104] utilized an exact line search to determine the step size along the descent direction at every iteration to compute the new iterate. When the exact line search is too expensive (i.e., too many objective function/gradient evaluations are required), different alternative rules can be used for the step size calculation (see, e.g., [86]). In particular, inexact line search methods, like the Armijo or the Goldstein conditions (see Section 2.5), can be applied [74] to calculate the step size. Due to the large number of subproblems we need to solve, a reduction of the number of function evaluations for the step size calculation is desirable. Therefore, inspired by the work in [99, 100, 101], we decided to use a non-monotone version of the Armijo line search, in case an exact line search is not applicable. This choice allows to accept a step size that gives a safe growth of the objective function, obtaining a new iterate with less computational effort.

The continuous relaxation of Problem (7.2), obtained by relaxing the integrality constraints, is the following convex nonlinear programming problem:

$$\begin{aligned} \min\ & h\Big(\sqrt{y^\top M y}\Big) - r^\top y \\ \text{s.t. } & a^\top y \le b \\ & y \ge 0 \\ & y \in \mathbb{R}^n. \end{aligned} \tag{7.3}$$

By the transformation $y_i = Diag(\frac{b}{a_i})x_i$ Problem (7.3) becomes

$$\begin{aligned} \min\ & f(x) = h\Big(\sqrt{x^\top Q x}\Big) - \mu^\top x \\ \text{s.t. } & e^\top x \le 1 \\ & x \ge 0 \\ & x \in \mathbb{R}^n, \end{aligned} \tag{7.4}$$

where we relabeled $Q = Diag\Big(\frac{b^2}{a_i^2}\Big)M$, $\mu = Diag(\frac{b}{a_i})r$ and $e = (1, \dots, 1)^\top$ denotes the vector of ones. The feasible regions of Problems (7.3) and (7.4) are illustrated in Figure 7.1.

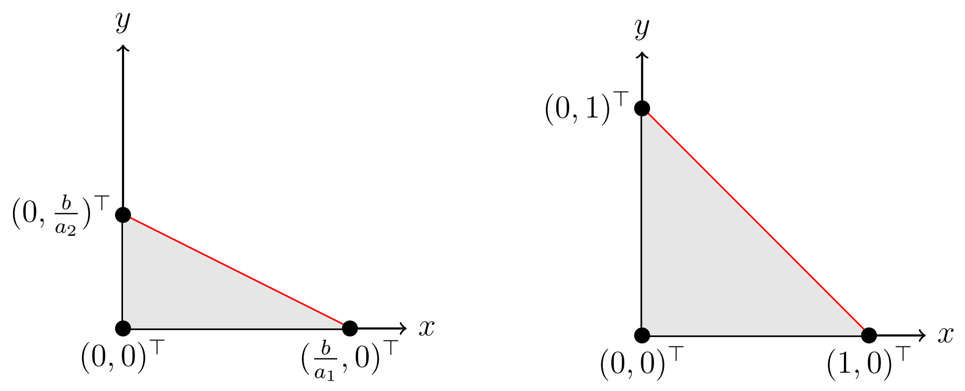

Figure 7.1: The initial and transformed feasible region of Problem (7.3) and (7.4), respectively, for $n = 2$. The red line depicts the area in which the optimal solution lies if it is not zero, in case $h(x)$ is chosen as Ωx.

7.1.1 Checking Optimality in Zero

A first difficulty in dealing with Problem (7.4) comes from the fact that the objective function may not be differentiable in $x = 0$. Therefore, we report a result [26] that enables checking if the point $x^\star = 0$ is the optimal solution of (7.4).

Proposition 7.1 ([26])**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then $x^\star$ minimizes f over a convex set $X \subset \mathbb{R}^n$ if and only if there exists a subgradient $d \in \partial f(x^\star)$, such that*

$$d^\top (x - x^\star) \geq 0 \quad \forall x \in X.$$

By Proposition 7.1, we have that $x^\star = 0$ is an optimal solution for Problem (7.4) if and only if there exists a subgradient $d \in \partial h(g(0))$, such that $(d - \mu)^\top x \geq 0$, for all $x \in \{x \in \mathbb{R}^n : e^\top x \leq 1, x \geq 0\}$. From standard results on convex analysis (see, e.g., [26]), we derive that

$$\partial h(g(0)) = h'(0) Q^{\frac{1}{2}} v,$$

where $v \in \{w \in \mathbb{R}^n : \|w\| \leq 1\}$. Then, from Proposition 7.1, we have that $x^\star = 0$ is an optimal solution for Problem (7.4) if and only if

$$\exists\, v \in \mathbb{R}^n \text{ such that } \|v\| \leq 1 \text{ and } \left(h'(0) Q^{\frac{1}{2}} v - \mu\right)^\top x \geq 0, \tag{7.5}$$

for all $x \in S = \{x \in \mathbb{R}^n : e^\top x \leq 1, x \geq 0\}$.

Since $x \in S$ implies $x \geq 0$, condition (7.5) is equivalent to the following:

$$\exists\, v \in \mathbb{R}^n \text{ with } \|v\| \leq 1 \text{ and } h'(0) Q^{\frac{1}{2}} v - \mu \geq 0. \tag{7.6}$$

We notice that in case $h'(0) = 0$, since $\mu \geq 0$ and $\mu \neq 0$, condition (7.6) cannot be satisfied for any $v \in \mathbb{R}^n$, such that $\|v\| \leq 1$. This means that, in the case $h'(0) = 0$, $x^\star = 0$ is not the only optimal solution of Problem (7.4).

By (7.6) we have a necessary and sufficient condition to decide if zero is the optimal solution of Problem (7.4).

7.1.2 Starting Point Computation

In case zero is not the optimal solution of Problem (7.4), we could produce a point $x^0 \neq 0$ such that $f(x^0) < f(0)$, and use it as a starting point for our method. For the classical case considered in the literature, where $h(t) = \Omega t$, we have the following theoretical result.

Proposition 7.2. *The optimal solution x^* of Problem (7.4) for $h(t) = \Omega t$ is either zero or it satisfies $e^\top x^\star = 1$.*

Proof. There exists an optimal solution $x^\star$ of Problem (7.4) due to the compactness of S and the continuity of f. Suppose $x^\star \neq 0$, then $f(x^\star) < 0$. We assume that for the optimal solution $x^\star > 0$ we have $e^\top x^\star < 1$. Since $x^\star$ is not a vertex of S, we can find a vector $y \in S$ and a scalar $\alpha \in (0, 1)$ such that

$$x^\star = (1 - \alpha)0 + \alpha y = \alpha y.$$

It follows that $0 > f(x^\star) = f(\alpha y) = \Omega\sqrt{(\alpha y)^\top Q(\alpha y)} - \mu^\top(\alpha y) = \alpha f(y)$. Therefore $f(y) < 0$, since $\alpha > 0$. But we have that $f(x^\star) = \alpha f(y) > f(y)$ since $\alpha < 1$, which is a contradiction to the optimality of $x^\star$. $\square$

This means we can restrict the search space to any point in S that lies on the hyperplane $e^\top x = 1$, in case the optimal solution is not zero. Thus, Problem (7.4) is equivalent to the minimization problem $\min\{f(x) \mid e^\top x = 1,\ x \geq 0\}$ and our objective function becomes differentiable on the reduced feasible set. It is sufficient to start with any feasible point satisfying $e^\top x = 1$, e.g., any unit vector since the optimal solution lies on the face induced by the inequality.

In the general case, where the objective function is non-differentiable, we can always compute a starting point for Problem (7.4). From a practical point of view, checking condition (7.6) is equivalent to solving the following convex quadratic program with non-negativity constraints with any QP-method:

$$\begin{aligned} &\min\ ||v||^2 \\ &\text{s.t. } h'(0)Q^{\frac{1}{2}}v \geq \mu. \end{aligned} \tag{7.7}$$

By the transformation $z := Q^{\frac{1}{2}}v - \frac{\mu}{h'(0)}$ we can transform the problem into the non-negativity constrained convex quadratic program

$$\begin{aligned} &\min\ z^\top Q^{-1} z + \frac{2}{h'(0)}\mu^\top z + \frac{1}{h'(0)^2}\mu^\top Q^{-1}\mu \\ &\text{s.t. } z \geq 0. \end{aligned} \tag{7.8}$$

To solve Problem (7.8) to optimality, we can use for example the tailored QP-solver `FAST-QPA` proposed in Chapter 5. If follows that the optimal solution of (7.7) is less or equal than 1, if and only if zero is the optimal solution of (7.4).

7.1.3 Outline of the Algorithm

Now, we can define our method and analyze its convergence properties. From now on we denote by S the feasible set of (7.4), that is $S = \{x \in \mathbb{R}^n : e^\top x \leq 1, x \geq 0\}$, which is compact and convex. Since the points x^k at each iteration are produced such that $f(x^k) \leq f(x^0)$, we have that $x^k \in \mathcal{L}(x^0) \cap S$ and $0 \notin \mathcal{L}(x^0) \cap S$, where $\mathcal{L}(x^0) := \{x \in \mathbb{R}^n \mid f(x) \leq f(x^0)\}$ is the x^0-*sub-level set* of f. We have the following observations:

- $\mathcal{L}(x^0) \cap S$ is compact, as it is a closed subset of the compact set S.
- f is continuously differentiable in $\mathcal{L}(x^0) \cap S$.
- h is Lipschitz continuous in S, since it is differentiable and S is compact.
- f is Lipschitz continuous in S with Lipschitz constant $L\sqrt{\lambda_{max}(Q)} + \|\mu\|$, where L is the Lipschitz constant of h.

Indeed, we have

$$\begin{aligned} \|\nabla f(x)\| &= \left\| h'(\|Q^{1/2}x\|)\tfrac{Qx}{\|Q^{1/2}x\|} - \mu \right\| \\ &\leq |h'(\|Q^{1/2}x\|)| \left\| Q^{1/2}\tfrac{Q^{1/2}x}{\|Q^{1/2}x\|} \right\| + \|\mu\| \\ &\leq |h'(\|Q^{1/2}x\|)| \|Q^{1/2}\| + \|\mu\| \\ &\leq L\sqrt{\lambda_{max}(Q)} + \|\mu\|. \end{aligned}$$

It follows that f is uniformly continuous in S.

In Algorithm 18 we report the basic scheme of our algorithm.

At each iteration k, the algorithm first computes a descent direction, choosing among a standard toward-step and an away-step direction (see Subsection 7.1.4).

Algorithm 18: NM-MFW

```
input  : a convex programming problem of the form (7.4)
output: the minimizer $x^\star$ of (7.4)
Fix maxit $> 0$ and choose a suitable starting point $x^0 \in S$.
for $k = 0, 1, \dots,$ maxit do
    Compute a descent direction $d^k$.
    if $\nabla f(x^k)^\top d^k = 0$ then
        STOP.
    Calculate a step size $\alpha^k \in (0, 1]$ by using a line search.
    Set $x^{k+1} = x^k + \alpha^k d^k$.
```

Then, in case optimality conditions are not satisfied, it calculates a step size along the given direction by a non-monotone line search (see Subsection 7.1.5), updates the point, and starts a new iteration.

7.1.4 Computation of a Feasible Descent Direction

For the computation of a feasible descent direction we follow the *away-step approach* described in [104]. While using away-step directions, even a linear convergence rate can be guaranteed, if an exact line search is used and suitable assumptions on the objective function are made.

At every iteration k, we first solve the following linearized problem for the toward-step

$$\begin{aligned} \hat{x}^k_{TS} := \text{argmin}\ & \nabla f(x^k)^\top (x - x^k) \\ \text{s.t.}\ & x \in S, \end{aligned} \tag{7.9}$$

and define $d^k_{TS} \in \mathbb{R}^n$ as $d^k_{TS} = \hat{x}^k_{TS} - x^k$. Then, we consider the problem related to the away-step

$$\begin{aligned} \hat{x}^k_{AS} := \text{argmax}\ & \nabla f(x^k)^\top (x - x^k) \\ \text{s.t.}\ & x \in S \\ & x_i = 0, \text{ if } x^k_i = 0, \end{aligned} \tag{7.10}$$

and define $d^k_{AS} \in \mathbb{R}^n$ as $d^k_{AS} = x^k - x^k_{AS}$.

In order to choose between the two directions, we use the criterion presented in [104]. In particular, if

$$\nabla f(x^k)^\top d^k_{TS} \le \nabla f(x^k)^\top d^k_{AS}, \tag{7.11}$$

then we set the toward-step direction: $\hat{x}^k = \hat{x}^k_{TS}$ and $d^k = \hat{x}^k - x^k = d^k_{TS}$. Otherwise we select the away-step direction: $\hat{x}^k = \hat{x}^k_{AS}$ and $d^k = x^k - \hat{x}^k = d^k_{AS}$.

We want to notice that for both Problems (7.9) and (7.10), we minimize a linear function over a polytope. Hence, the solution can be found by checking the objective function value on the vertices of the feasible set $S = \mathrm{conv}(\{0, e_1, \dots, e_n\})$. Therefore, we just need to compare the values $\nabla_i f(x^k)$, $i = 1, \dots, n$ and both solutions can be obtained at a computational cost of $O(n)$. In the following we denote by $\hat{\imath}$ the index of the vertex, at which the optimal solution is obtained.

7.1.5 Computation of a Suitable Step Size

In case an exact line search can be used and the objective function satisfies specific assumptions (for details, see, e.g., [104, 130]), the Frank-Wolfe algorithm using away-steps converges linearly to an optimal solution. When an exact line search approach cannot be considered, we combine the away-step approach with a non-monotone inexact line search. Even if in the latter case there is no proof that the algorithm converges linearly, good results can be observed in practice as we will see in our experimental results.

Now we can describe the non-monotone line search used in our algorithm for the general case: a step size is accepted as soon as it gives a point which allows a sufficient decrease with respect to a reference value. A classical choice for the reference value $\bar{f}$ in Algorithm 19 is the maximum among the last nm objective function values computed, where nm is a positive integer. We report the detailed scheme below. The maximum step size α_{max} used in Algorithm 19 is set to α_{TS} if the toward-step direction is chosen; it is set to α_{AS} otherwise.

Algorithm 19: Non-monotone Armijo line search

```
input  : iterates $x^0, \dots, x^k$ and descent direction $d^k$ in Algorithm 18
output: a step size $\alpha$ for Algorithm 18 in iteration $k$
Set $\delta \in (0,1)$, $\gamma_1 \in (0, \frac{1}{2})$, $\gamma_2 \geq 0$, $nm > 0$.
Update
    $\bar{f}^k = \max_{0 \leq i \leq \min\{nm,k\}} f(x^{k-i}).$
Calculate starting step size $\alpha \in (0, \alpha_{max}]$.
while $f(x^k + \alpha d^k) > \bar{f}^k + \gamma_1 \alpha \nabla f(x^k)^\top d^k - \gamma_2 \alpha^2 \|d^k\|^2$ do
    Set $\alpha = \delta\alpha$.
```

When the away-step direction is chosen, then the starting step size for the line search is set to

$$\alpha_{AS} = \max\{\alpha \geq 0 \mid x^k + \alpha d^k_{AS} \in S\}.$$

Namely, if $\hat{x}^k_{AS} = e_{\hat{\imath}}$ we set $\alpha_{AS} = \frac{x^k_{\hat{\imath}}}{1 - x^k_{\hat{\imath}}}$, otherwise $\alpha_{AS} = \frac{1 - e^\top x^k}{e^\top x^k}$. On the other hand, if the toward-step direction is chosen, the starting step size for the line search is simply set to $\alpha_{TS} = 1$. In case an away-step is chosen but $\alpha_{AS} < \beta$, where $0 < \beta \ll 1$ is a certain threshold value, we decide to perform a toward-step instead to ensure proper convergence, as will become clear in Section 7.1.7.

The following result states that the non-monotone Armijo line search terminates in a finite number of steps.

Proposition 7.3. *Assume that $\nabla f(x^k)^\top d^k < 0$. Then the non-monotone Armijo line search determines, in a finite number of iterations, a step size α^k such that*

$$f(x^k + \alpha^k d^k) \leq \bar{f}^k + \gamma_1 \alpha^k \nabla f(x^k)^\top d^k - \gamma_2 (\alpha^k)^2 \|d^k\|^2. \tag{7.12}$$

The result is proved using similar arguments as in the proof of Proposition 3 in [98].

Proof. Let j be the iteration counter of the cycle in the Armijo line search of Algorithm 18 and α^j be the current step size of iteration j. We can write

$$\alpha^j = \delta^j \alpha^0,$$

so that $\alpha^j \to 0$ for $j \to \infty$. Let us assume, by contradiction, that the cycle does not terminate. Then, for all j we have

$$f(x^k + \alpha^j d^k) > \max_{0 \leq i \leq \min\{nm,k\}} [f(x^{k-i})] + \gamma_1 \alpha^j \nabla f(x^k)^\top d^k - \gamma_2 (\alpha^j)^2 \|d^k\|^2.$$

It follows

$$\frac{f(x^k + \alpha^j d^k) - f(x^k)}{\alpha^j} - \gamma_1 \nabla f(x^k)^\top d^k > -\gamma_2 \alpha^j \|d^k\|^2. \tag{7.13}$$

Taking the limits in (7.13) for $j \to \infty$, as $\gamma_1 < 1$ and $\alpha^j \to 0$, we obtain $\nabla f(x^k)^\top d^k \geq 0$, which contradicts the assumption on d^k. □

7.1.6 Incremental Computations

Similar as in Chapter 4, we again exploit the fact that we can speed up the objective function evalutations by using fast incremental updates. During the loop of the Armijo line search in Algorithm 19, by storing the values of ${x^k}^\top Q x^k \in \mathbb{R}$, $Qx^k \in \mathbb{R}^n$, and $\mu^\top x^k \in \mathbb{R}$ for every iteration k, we are able to evaluate the terms ${y^k}^\top Q y^k + \mu^\top y^k$ for all trial points $y^k = x^k + \alpha d^k$ in constant time per node. In particular, if a toward-step is performed in the line search, i.e., $d^k = d^k_{TS} = e_{\hat{\imath}} - x^k$, we have the following incremental updates:

$$\begin{aligned}
{y^k}^\top Q y^k &= (x^k + \alpha d^k)^\top Q (x^k + \alpha d^k) \\
&= {x^k}^\top Q x^k + \alpha(\alpha - 2){x^k}^\top Q x^k + 2\alpha(1-\alpha){x^k}^\top Q e_{\hat{\imath}} + \alpha^2 Q_{\hat{\imath}\hat{\imath}} \\
&= (1-\alpha)^2 {x^k}^\top Q x^k + 2\alpha(1-\alpha)(Qx^k)_{\hat{\imath}} + \alpha^2 Q_{\hat{\imath}\hat{\imath}} \\
\mu^\top y^k &= (1-\alpha)\mu^\top x^k + \alpha \mu_{\hat{\imath}}.
\end{aligned}$$

In case $\hat{x} = 0$, i.e., $d^k = -x^k$, the incremental updates simplify to

$$\begin{aligned} {y^k}^\top Q y^k &= (1-\alpha)^2 {x^k}^\top Q x^k \\ \mu^\top y^k &= (1-\alpha)\mu^\top x^k. \end{aligned}$$

The running time for evaluating the objective function f at the trial points y^k depends on the choice of the risk-function h. In case h can be evaluated in constant time, this is also true for the computation of $f(y^k)$ and the objective function evaluations can be carried out in $O(1)$ time per iteration of Algorithm 19.

Note that the running time for computing the new values ${x^{k+1}}^\top Q x^{k+1} \in \mathbb{R}$, $Qx^{k+1} \in \mathbb{R}^n$, and $\mu^\top x^{k+1} \in \mathbb{R}$ for the next iteration of Algorithm 18 is $O(n)$ per iteration, since it is dominated by the update of $Qx^{k+1} \in \mathbb{R}^n$. We can use the same formulas from above to update the scalars ${x^{k+1}}^\top Q x^{k+1} \in \mathbb{R}$ and $\mu^\top x^{k+1} \in \mathbb{R}$ in constant time, while the vectors $Qx^{k+1} \in \mathbb{R}^n$ can be updated in linear time by

$$Qx^{k+1} = \begin{cases} (1-\alpha)Qx^k + \alpha Q e_{\hat{\imath}} & \text{if } \hat{x}^k = e_{\hat{\imath}} \\ (1-\alpha)Qx^k & \text{if } \hat{x}^k = 0. \end{cases}$$

Similar computations apply if an away-step is performed, i.e., $d^k = d^k_{AS}$.

7.1.7 Convergence Analysis

In this section, we analyze the convergence properties of the non-monotone Frank-Wolfe Algorithm 18 with away-steps. In order to do that, we need to prove some preliminary results. We first prove, in the following Lemma, that the sequence $\{\bar{f}^k\}_k$ converges.

Lemma 7.4. *Suppose that Algorithm 18 produces an infinite sequence $\{x^k\}_k$. Then*

- *$x^k \in \mathcal{L}(x^0) \cap S$ for all k, and*
- *$\{\bar{f}^k\}_k$ is non-increasing and converges to a value $\bar{f}$.*

Proof. From the non-monotone Armijo line search we have that

$$\begin{aligned} \bar{f}^{k+1} &= \max_{0 \le i \le \min\{nm, k+1\}} f(x^{k+1-i}) \\ &\le \max\{\bar{f}^k, f(x^{k+1})\}. \end{aligned}$$

Since $f(x^{k+1}) < \bar{f}^k$ from the non-monotone line search, then

$$\bar{f}^{k+1} \le \bar{f}^k$$

which proves that $\{\bar f^k\}_k$ is non increasing. Now, taking into account the fact that $f(x^k) \le f(x^0)$ for all k, we get that the sequence $\{x^k\}_k$ belongs to the set $\mathcal{L}(x^0) \cap S$. Therefore, we get that the non increasing sequence $\{\bar f^k\}_k$ is bounded from below, and converges to $\bar f$. □

Now, we prove that the sequence $\{f(x^k)\}_k$ converges to the same limit as the sequence $\{\bar f^k\}_k$.

Lemma 7.5. *Suppose that Algorithm 18 produces an infinite sequence $\{x^k\}_k$. Then*

$$\lim_{k\to+\infty} f(x^k) = \lim_{k\to+\infty} \bar f^k = \bar f. \tag{7.14}$$

Proof. Let $\{x^{t^k}\}_k \subseteq \{x^k\}_k \subset \mathcal{L}(x^0) \cap S$ be the subsequence of points such that $\bar f^k = f(x^{t^k})$ and $k - \min(k, nm) \le t^k \le k$.

We prove by induction that for any fixed integer $i \ge 1$ we have

$$\lim_{k\to+\infty} f(x^{t^k - i}) = \lim_{k\to+\infty} f(x^{t^k}) = \lim_{k\to+\infty} \bar f^k = \bar f. \tag{7.15}$$

Suppose at first $i = 1$.

$$\begin{aligned}\bar f^k &= f(x^{t^k}) = f(x^{t^k-1} + \alpha^{t^k-1} d^{t^k-1})\\ &\le \bar f^{t^k-1} + \gamma_1 \alpha^{t^k-1} \nabla f(x^{t^k-1})^\top d^{t^k-1} - \gamma_2 (\alpha^{t^k-1})^2 \|d^{t^k-1}\|^2\\ &\le \bar f^{t^k-1} - \gamma_2 (\alpha^{t^k-1})^2 \|d^{t^k-1}\|^2,\end{aligned}$$

where the second inequality holds since $\nabla f(x^{t^k-1})^\top d^{t^k-1} < 0$. This means that

$$\bar f^k - \bar f^{t^k-1} \le -\gamma_2 (\alpha^{t^k-1})^2 \|d^{t^k-1}\|^2.$$

From Lemma 7.4 the left hand side converges to zero and we obtain

$$\lim_{k\to+\infty} (\alpha^{t^k-1})^2 \|d^{t^k-1}\|^2 = 0,$$

that means $\lim_{k\to+\infty} \|x^{t^k} - x^{t^k-1}\| = 0$. Due to the uniform continuity of $f(x)$ over $\mathcal{L}(x^0) \cap S$ equation (7.15) holds for $i = 1$.

We now assume that (7.15) holds for $i \ge 1$ and we prove that it holds for index $i + 1$.

We have

$$f(x^{t^k-i}) \le \bar f^{t^k-i-1} + \gamma_1 \alpha^{t^k-i-1} \nabla f(x^{t^k-i-1})^\top d^{t^k-i-1} - \gamma_2 (\alpha^{t^k-i-1})^2 \|d^{t^k-i-1}\|^2$$

and reasoning as before yields

$$f(x^{t^k-i}) - \bar{f}^{t^k-i-1} \leq -\gamma_2(\alpha^{t^k-i-1})^2 \|d^{t^k-i-1}\|^2. \tag{7.16}$$

The left hand side of (7.16) tends to zero since (7.15) holds for i, i.e., $f(x^{t^k-i})$ converges to $\bar{f}$. Then,

$$\lim_{k\to+\infty} (\alpha^{t^k-i-1})^2 \|d^{t^k-i-1}\|^2 = 0,$$

that means $\lim_{k\to+\infty} \|x^{t^k-i} - x^{t^k-i-1}\| = 0$. Again, uniform continuity of $f(x)$ over $\mathcal{L}(x^0) \cap S$ yields (7.15) for index $i+1$.

Let $T^k = t^{k+nm+1}$. In order to conclude the proof we notice that for any index k we can write

$$f(x^k) = f(x^{T^k}) - \sum_{i=0}^{T^k-k-1} (f(x^{T^k-i}) - f(x^{T^k-i-1})).$$

Therefore, taking the limit for $k \to +\infty$ we obtain (7.14). $\square$

The next result shows that the sequence $\{\nabla f(x^k)^\top d^k\}_k$ converges to zero.

Lemma 7.6. *Let $\{x^k\}_k \subset \mathcal{L}(x^0) \cap S$ be the sequence of points produced by Algorithm 18. Then*

$$\lim_{k\to+\infty} \nabla f(x^k)^\top d^k = 0. \tag{7.17}$$

Proof. First of all we notice that $\nabla f(x^k)^\top d^k < 0$ for all k. Let α^k be the step size computed by Algorithm 18 at iteration k. Then,

$$\bar{f}^k - f(x^k + \alpha^k d^k) \geq \gamma_1 \, \alpha^k \, |\nabla f(x^k)^\top d^k| + \gamma_2 \, (\alpha^k)^2 \, \|d^k\|^2 \geq \gamma_1 \, \alpha^k \, |\nabla f(x^k)^\top d^k|.$$

From Lemma 7.5, since $\lim_{k\to+\infty}(\bar{f}^k - f(x^k + \alpha^k d^k)) = 0$, we have

$$\lim_{k\to+\infty} \gamma_1 \, \alpha^k \, |\nabla f(x^k)^\top d^k| = 0. \tag{7.18}$$

Suppose by contradiction that (7.17) does not hold. This implies that a subsequence $\{\nabla f(x^k)^\top d^k\}_k$, which we relabel as $\{\nabla f(x^k)^\top d^k\}$, exists such that

$$\lim_{k\to+\infty} \nabla f(x^k)^\top d^k = -\eta < 0.$$

Then, by (7.18) we have that $\lim_{k\to+\infty} \alpha^k = 0$ must hold.

Let $D > 0$ be the diameter of S. Since $\|d^k\| \leq \|x^k\| + \|\hat{x}^k\| \leq 2D$ we have that the sequence $\{d^k\}_k$ is bounded. The sequence $\{x^k\}_k \subset \mathcal{L}(x^0) \cap S$ is also bounded, so that appropriate subsequences, which we relabel as $\{d^k\}_k$ and $\{x^k\}_k$, exist such that

$$\lim_{k\to+\infty} x^k = \bar{x}; \qquad \lim_{k\to+\infty} d^k = \bar{d}.$$

From the continuity of the gradient in $\mathcal{L}(x^0) \cap S$ we have:

$$\nabla f(\bar{x})^\top \bar{d} = \lim_{k\to+\infty} \nabla f(x^k)^\top d^k = -\eta < 0.$$

Since $\alpha_{max} \geq \beta > 0$ and the sequence α^k is converging to zero, a value $\bar{k} \in \mathbb{N}$ exists such that $\alpha^k < \alpha_{max}$, for $k \geq \bar{k}$. In other words, for $k \geq \bar{k}$ the step size α^k cannot be set to the maximum step size and, taking into account the non-monotone Armijo line search, we can write

$$f\Big(x^k + \frac{\alpha^k}{\delta} d^k\Big) > \bar{f}^k + \gamma_1 \frac{\alpha^k}{\delta} \nabla f(x^k)^\top d^k - \gamma_2 \Big(\frac{\alpha^k}{\delta}\Big)^2 \|d^k\|^2.$$

Hence, due to the fact that $\bar{f}^k \geq f(x^k)$, we get

$$f\Big(x^k + \frac{\alpha^k}{\delta} d^k\Big) - f(x^k) > \gamma_1 \frac{\alpha^k}{\delta} \nabla f(x^k)^\top d^k - \gamma_2 \Big(\frac{\alpha^k}{\delta}\Big)^2 \|d^k\|^2. \tag{7.19}$$

Since f is continuously differentiable in $\mathcal{L}(x^0) \cap S$, we can apply the Mean Value Theorem and we have that $s^k \in [0, 1]$ exists such that

$$f\Big(x^k + \frac{\alpha^k}{\delta} d^k\Big) = f(x^k) + \frac{\alpha^k}{\delta} \nabla f\Big(x^k + s^k \frac{\alpha^k}{\delta} d^k\Big)^\top d^k. \tag{7.20}$$

In particular we have $\lim_{k\to+\infty} x^k + s^k \frac{\alpha^k}{\delta} d^k = \bar{x}$. By substituting (7.20) within (7.19) we have

$$\nabla f\Big(x^k + s^k \frac{\alpha^k}{\delta} d^k\Big)^\top d^k > \gamma_1 \nabla f(x^k)^\top d^k - \gamma_2 \frac{\alpha^k}{\delta} \|d^k\|^2.$$

Considering the limit on both sides we get

$$-\eta = \nabla f(\bar{x})^\top \bar{d} > \gamma_1 \nabla f(\bar{x})^\top \bar{d} = -\gamma_1 \eta$$

and we get a contradiction since $\gamma_1 \in (0, \frac{1}{2})$. □

Finally, we can prove the main convergence result related to Algorithm 18. We notice that due to the use of the line search, there is no need to make any particular assumption on the gradient of the objective function (like, e.g., Lipschitz assumption) for proving convergence.

Proposition 7.7. *Let $\{x^k\}_k \subseteq \mathcal{L}(x^0) \cap S$ be the sequence of points produced by Algorithm 18. Then, either an integer $k \geq 0$ exists such that x^k is an optimal solution for Problem* (7.3)*, or the sequence $\{x^k\}_k$ is infinite and every limit point $x^\star$ of the sequence is an optimal solution for Problem* (7.4)*.*

Proof. If Algorithm 18 does not stop in a finite number of iterations at an optimal solution, from Proposition 7.6 we have that

$$\lim_{k\to+\infty} \nabla f(x^k)^\top d^k = 0.$$

Since the set $\mathcal{L}(x^0) \cap S$ is compact and the sequence $\{d^k\}_k$ is bounded, we have that appropriate subsequences, which we relabel as $\{d^k\}_k$ and $\{x^k\}_k$, exist such that

$$\lim_{k\to+\infty} x^k = x^\star; \qquad \lim_{k\to+\infty} d^k = d^\star.$$

Therefore

$$\nabla f(x^\star)^\top d^\star = \lim_{k\to+\infty} \nabla f(x^k)^\top d^k = 0.$$

From the definition of d^k we have

$$\nabla f(x^k)^\top d^k \le \nabla f(x^k)^\top (x - x^k) \quad \forall\, x \in S.$$

Taking the limit for $k \to +\infty$ we have

$$0 = \nabla f(x^\star)^\top d^\star \le \nabla f(x^\star)^\top (x - x^\star) \quad \forall\, x \in S,$$

that is

$$\nabla f(x^\star)^\top (x - x^\star) \ge 0 \quad \forall\, x \in S$$

and $x^\star$ is an optimal solution for Problem (7.4). □

7.1.8 Dual Bound Computation

Algorithm 18 is used within a branch-and-bound algorithm presented in Section 7.2 to compute valid dual bounds. To speed up the enumeration process, we take advantage of the following surrogate duality gap [115] for Problem (7.4):

$$g(x) := \max_{z\in S} \nabla f(x)^\top (x - z),$$

defined for all $x \in S \setminus \{0\}$. Then, convexity of f implies that $g(x)$ is an upper bound for the gap between $f(x)$ and $f(x^\star)$:

$$f(x) - f(x^\star) \le \nabla f(x)^\top (x - x^\star) \le g(x).$$

In particular, when considering the search direction computed in Algorithm 18, we have that $g(x^k) = -\nabla f(x^k)^\top d^k$, yielding

$$f(x^k) + \nabla f(x^k)^\top d^k \le f(x^*).$$

Therefore, we get a dual bound for free in each iteration of Algorithm 18 by the computation of the step direction d^k. This implies the possibility of an early pruning, already successfully used in `FAST-QPA-BB` and explained in Section 5.4.2. This premature termination of the iteration process leads to a faster overall enumeration. Due to the non-monotone Armijo line search we choose to update the value of the dual bound only when there is an improvement in the objective function.

7.2 A Branch-and-Bound Algorithm for GCBP

The branch-and-bound algorithm follows the successfully adapted ideas of Chapter 4 and Chapter 5 with the aim of a fast enumeration of the search tree. In the following, we describe the branch-and-bound algorithm by briefly summarizing its main components.

Upper Bound As an initial upper bound in the branching tree, we use a simple greedy heuristic, adapted from Julstrom [117] for the quadratic knapsack problem. Analogously to the notation used in the theory of knapsack problems the profit ratio p_i of an item i is defined as the sum of all profits that are gained by putting item i into the knapsack of capacity b, divided by its weight a_i. Transferred to our application, we have $p_i := \Big(h\Big(\sqrt{m_{ii} + 2\sum_{j\neq i} m_{ij}}\Big) - r_i\Big)/a_i$, for all $i = 1, \ldots, n$. The heuristic sorts all items $1, \ldots, n$ in a non-decreasing order and successively fills the knapsack with the best items until the capacity of the knapsack is reached. Since the original algorithm proposed by Julstrom is tailored for a binary problem, we adapt it by allowing multiple copies of each item. The overall running time of the heuristic is $O(n^2)$.

During the branch-and-bound enumeration, we do not use any heuristics for improving the primal bound, since the quick enumeration using a depth-first search yields quick updates of the bound. Once all integer variables have been fixed, we compute the optimal solution of the subproblem in the reduced continuous subspace. If the computed point is feasible, it yields a valid upper bound for the original problem.

Branching We use the same branching scheme as explained in Section 4.2, i.e., we branch by successively fixing a variable in increasing distance to its value in the fractional solution of the current node relaxation. Furthermore, the branching order of the variables at every level is predetermined.

Incremental Computations An advantage of branching by fixing variables is that the subproblems in the enumeration process of the search tree have the very same structure, just in a reduced dimension. Let $\ell \in \{0, \ldots, n_1 - 1\}$ be the current depth in the branch-and-bound tree. By moving the part resulting from fixing ℓ variables into the linear and constant part, respectively, the problem reduces to the minimization of

$$\bar{f} : \mathbb{Z}^{n_1-\ell} \times \mathbb{R}^{n-n_1} \to \mathbb{R},\ x \mapsto \sqrt{x^\top M_\ell x + c_\ell^\top x + d_\ell} - r_\ell x - e_\ell \tag{7.21}$$

over the feasible region $S_\ell = \{x \in \mathbb{Z}^{n_1-\ell} \times \mathbb{R}^{n-n_1} \mid a_\ell^\top x \leq b_\ell,\ x \geq 0\}$. We notice that in the case where there is at least one variable that is fixed to a nonzero value, the objective function is differentiable everywhere in S_ℓ, since $d_\ell > 0$.

The matrix M_ℓ is obtained by deleting the corresponding ℓ rows and columns of M and c_ℓ, d_ℓ and e_ℓ are adapted appropriately by

$$(c_\ell)_{j-\ell} := c_j + 2\sum_{i=1}^{\ell} \sigma_{ij} s_i, \text{ for all } j = \ell+1, \dots, n$$

and

$$d_\ell := d + \sum_{i=1}^{\ell} c_i s_i + \sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \sigma_{ij} s_i s_j, \quad e_\ell := e + \sum_{i=1}^{\ell} r_i s_i,$$

where $s = (s_1, \dots, s_\ell) \in \mathbb{Z}^\ell$ is the current fixing at depth ℓ. Similarly a_ℓ is obtained by deleting the corresponding entries of a and the reduced right hand side b_ℓ is updated accordingly by $b_\ell = b - \sum_{i=1}^{\ell} s_i a_i$.

Preprocessing To check whether zero is the minimizer of the node relaxation at any level ℓ of the enumeration tree, we have derived the necessary and sufficient condition (7.5). We point out that checking this condition can be performed completely in a preprocessing phase, since the reduced matrices $Q_\ell^{\frac{1}{2}}$ and reduced linear coefficient vectors μ_ℓ do not depend on the specific fixings. Therefore the problem data of Problem (7.8) is known in advance for every level ℓ and it can be solved before starting the enumeration.

Warmstart With the aim of speeding up our branch-and-bound scheme, we use a warmstart procedure by taking over information from the parent node. The branching order of the variables at every level ℓ is $x_1, \dots, x_{n_1}$. Let $x^\star \in \mathbb{R}^{n-\ell}$ be the optimal solution of the subproblem after fixing the first $\ell \geq 1$ variables to their integer values. Assume we fix the next component $x_{\ell+1}$. We then consider the point $\tilde{x} := (x_2^\star, \dots, x_{n-\ell}^\star) \in \mathbb{R}^{n-\ell-1}$. If $\tilde{x}$ is feasible for the current node relaxation, we take it as a starting point for Algorithm 18, otherwise we take one of the following feasible points:

($\tilde{x} \vee e_1$): $e_1 = (1, 0, \dots, 0) \in \mathbb{R}^{n-\ell-1}$,

($\tilde{x} \vee e_p$): $e_p \in \mathbb{R}^{n-\ell-1}$, that is the projection of $\tilde{x}$ onto the set

$$\tilde{S} = \{x \in \mathbb{R}^{n-\ell-1} \mid e^\top x = 1, \ x \geq 0\}$$

or

($\tilde{x} \vee e_{\hat{\imath}}$): $e_{\hat{\imath}}$, where

$$\hat{\imath} := \operatorname{argmin} \left\{ \sqrt{q_{ii} + \sum_{j \neq i}^{n-\ell-1} 2q_{ij}} - \mu_i \mid i = 1, \dots, n-\ell-1 \right\}.$$

The feasible point e_p is computed by solving the nonlinear optimization problem

$$\min \left\{ \frac{1}{2}||x - \tilde{x}||^2 \mid \sum_{i=1}^{n} x_i = 1,\ x \geq 0 \right\}$$

using an algorithm originally proposed by Held et al. [109], that was recently rediscovered by Duchi et al. [73]. For the latter version, sketched in Algorithm 20, the overall complexity has been proven to be $O(n^2)$. Since the projection of a vector onto the so-called l_1-ball has many applications in the areas of statistics, operations research and machine learning, improved versions of this algorithm have been proposed, most of them using different sophisticated data structures and sorting techniques. For a recent survey on algorithms for the projection onto the l_1-ball, we refer to Condat [53].

Algorithm 20: Projection of $\tilde{x}$ onto the set $\tilde{S}$

```
input  : a vector $\tilde{x} \in \mathbb{R}^n$
output: the projection $x^\star$ of $\tilde{x}$ onto $\tilde{S}$
Sort $\tilde{x}$ into $\bar{x}$: $\bar{x}_1 \geq \bar{x}_2 \geq \ldots \geq \bar{x}_n$.
Find $\rho(\bar{x}) = \max\left\{ j \in \{1, \ldots, n\} \mid \bar{x}_j - \frac{1}{j}\left(\sum_{i=1}^{j} \bar{x}_i - 1\right) > 0 \right\}$.
Define $\lambda^\star = \left(\sum_{i=1}^{\rho} \bar{x}_i - 1\right)/\rho$.
Set $x_i^\star = \max\{\tilde{x}_i - \lambda^\star, 0\}$.
```

The unit vector $e_{\hat{\imath}}$ is chosen by adapting ideas of the greedy heuristic by Julstrom [117]. It represents the vertex of S where the simultaneous potential risk increase by setting $x_{\hat{\imath}} = 1$ is minimized.

7.3 Experimental Results

In order to investigate the potential of our algorithm `FW-BB` applied to the capital budgeting problem 7.2, we produced an implementation in C++/Fortran 90. As benchmark data set, we used real-world capital market indices from the Standard & Poor's 500 index (S&P 500) that were used and made public by Cesarone, Scozzari and Tardella [51]. This data set was used for solving a *Limited Asset Markowitz (LAM)* model. For each of the 500 stocks the authors obtained 265 weekly price data, adjusted for dividends, from Yahoo Finance for the period from March 2003 to March 2008. Stocks with more than two consecutive missing values were disregarded. The missing values of the remaining stocks were interpolated, yielding 476 stocks in total. Logarithmic weekly returns, expected returns and covariance matrices based on the data for the period March 2003 to March 2007 were computed. By choosing stocks at random from the 476 available, we built

Table 7.1: Results for the pure integer instances with $h(t) = \sqrt{\frac{1-0.91}{0.91}}\, t$ and $b = b_3$, running the non-monotone and monotone version of `FW-BB`.

	`NM-FW-BB`				`M-FW-BB`			
n	#	time	nodes	it	#	time	nodes	it
50	9	12.51	7.07e+05	17.00	6	137.90	2.21e+05	91.50
75	10	582.89	5.20e+06	35.80	6	118.38	3.95e+05	78.33
100	9	359.06	3.33e+06	32.67	5	644.72	4.62e+06	57.20

portfolio optimization instances of different sizes. Namely, we built 10 problems each with 50, 75 and 100 stocks for the pure integer case (i.e., $n_1 = n$) and 10 problems each with 100, 150 and 200 for the mixed-integer case, for which we considered $n_1 = \lfloor n/2 \rfloor$. We considered 3 different values for the budget b of the investor, namely $b \in \{b_1 := 1 \cdot \sum_{i=1}^n a_i, b_2 := 10 \cdot \sum_{i=1}^n a_i, b_3 := 100 \cdot \sum_{i=1}^n a_i\}$, having a total of 90 instances each for the pure integer case and for the mixed integer one. All experiments were carried out on Intel Xeon processors running at 2.60 GHz. All running times are measured in cpu-seconds and the time limit is 1 cpu-hour. In the following, we first present a numerical experience related to our algorithm `FW-BB`: we explore the benefits of using the non-monotone line search and using warmstart alternatives. Then, we present a comparison of `FW-BB` with the MIQP and SOCP solver of `CPLEX 12.6`, for the two cases $h(t) = \Omega t$ and $h(t) = t^2$, respectively. In the latter case, we also used the branch-and-bound solver `B-BB` of the open-source optimization software package `Bonmin 1.8.1` for comparison. Finally, we report some numerical tests on a different risk-adjusted function.

7.3.1 Non-monotone Line Search and Warmstarts

The Algorithm 18 that we have devised in Section 7.1 uses a non-monotone line search and in our implementation of `FW-BB` we set $nm = 1$. In order to show the benefits of the non-monotone version of `FW-BB` we report in Table 7.1 the comparison between the non-monotone version (`NM-FW-BB`) and the monotone one (`M-FW-BB`). The comparison is made on the pure integer instances with $h(t) = \Omega t$ and as a budget constraint we chose $a^\top x \leq b_3$, where b_3. In Table 7.1 we report, for each dimension, the number of instances solved within the time limit (#), the average running times (time), the average numbers of the branch-and-bound nodes (nodes) and the average numbers of iterations of FW in each node of the enumeration tree (it). By considering the non-monotone line search, `FW-BB` is able to solve more instances and the average number of iterations of Algorithm 18 is almost halved. In order to investigate the benefits of the warmstart choices $(\tilde{x} \vee e_1)$, $(\tilde{x} \vee e_p)$, $(\tilde{x} \vee e_{\hat{\imath}})$ we run the different versions of `FW-BB` on the pure and

Table 7.2: Results for different starting point choices for the pure integer instances with $h(x) = \sqrt{\frac{1-0.91}{0.91}}\ x$ and $b = b_3$, running `FW-BB`.

	e_1		$e_{\hat{\imath}}$		$\tilde{x} \vee e_1$		$\tilde{x} \vee e_p$		$\tilde{x} \vee e_{\hat{\imath}}$	
n	#	time	#	time	#	time	#	time	#	time
50	9	92.93	9	81.83	9	20.38	7	2.87	9	16.30
75	7	116.10	7	122.12	9	346.51	5	18.53	10	692.32
100	9	225.19	9	226.23	9	518.65	6	73.85	9	416.47

Table 7.3: Results for different starting point choices for the mixed-integer instances with $h(x) = \sqrt{\frac{1-0.91}{0.91}}\ x$ and $b = b_3$, running `FW-BB`.

	e_1		$e_{\hat{\imath}}$		$\tilde{x} \vee e_1$		$\tilde{x} \vee e_p$		$\tilde{x} \vee e_{\hat{\imath}}$	
n	#	time	#	time	#	time	#	time	#	time
100	8	1.95	9	23.81	10	38.25	10	3.03	10	5.06
150	9	7.14	9	9.92	9	14.22	10	13.32	9	15.17
200	5	101.29	5	123.24	8	403.14	10	169.84	9	411.50

mixed-integer instances with $h(t) = \Omega t$ and as a budget constraint $a^\top x \leq b_3$. We compare the three warmstart possibilities presented to the following alternatives:

(e_1): choose at every node $e_1 = (1, 0, \ldots, 0) \in \mathbb{R}^{n-\ell-1}$ or

($e_{\hat{\imath}}$): choose at every node $e_{\hat{\imath}} \in \mathbb{R}^{n-\ell-1}$.

In Table 7.2 we show the results related to the five different starting point choices for the pure integer instances. We can observe that, by choosing $(\tilde{x} \vee e_{\hat{\imath}})$, `FW-BB` is able to solve the most instances within the time limit. The alternatives that do not take into account the parent node are better for instances with $n = 100$. By choosing $(\tilde{x} \vee e_p)$, `FW-BB` solve the lowest number of instances, but it is faster with respect to the average running times for the instances solved. In Table 7.3 we show the results related to the five different starting point choices for the mixed-integer instances. We can observe that the only choice that enables `FW-BB` to solve all the instances within the time limit is $(\tilde{x} \vee e_p)$. Choosing $(\tilde{x} \vee e_{\hat{\imath}})$ looks like the second best choice in terms of performance. We also observe that choosing $(\tilde{x} \vee e_1)$, is better than considering e_1 or $e_{\hat{\imath}}$ as starting points, highlighting the benefits from using warmstarts.

7.3.2 Comparison to CPLEX 12.6 and Bonmin 1.8.1

In this section we present a numerical comparison on instances for $h(t) = \Omega t$ and $h(t) = t^2$. We compare `FW-BB` with the MIQP and the SOCP solver of `CPLEX 12.6`, respectively. In case $h(t) = t^2$, we additionally tested the branch-and-bound solver `B-BB` of `Bonmin 1.8.1`. Concerning `FW-BB`, we consider the two non-monotone versions, `FW-BB-P` and `FW-BB-G`, using $(\tilde{x} \vee e_p)$ and $(\tilde{x} \vee e_{\hat{\imath}})$, respectively. We use an absolute optimality tolerance of 10^{-10} for all algorithms.

Comparison on instances with $h(t) = \Omega t$. In order to compare `FW-BB` with `CPLEX 12.6`, we modeled GCBP as an equivalent mixed-integer second-order cone program (SOCP):

$$z := -\min\left\{y - r^\top x : a^\top x \le b, \Omega\sqrt{x^\top M x} \le y, x \ge 0, x_1, \ldots, x_{n_1} \in \mathbb{Z}_+, y \in \mathbb{R}\right\}. \tag{7.22}$$

We chose $\Omega = \sqrt{(1-\varepsilon)/\varepsilon}$, where $\varepsilon \in \{0.91, 0.95, 0.99\}$. The value of ε controls the amount of risk the investor is willing to take. In theory, ε can take any value in (0,1], where a small value implies a bigger weight on the risk-term and $\varepsilon = 1$ means that the risk is not taken into account. Numerical tests on single instances showed that any value of ε in (0,0.9] gives the trivial optimal solution zero, i.e., not investing anything is the optimal decision for the investor. Therefore, we restricted our experiments to the three values of ε mentioned above.

The performance of the considered algorithms for the pure integer and the mixed-integer instances can be found in Table 7.4 and Table 7.5, respectively. The tables include the following data: numbers of instances solved within the time limit, average running times, average numbers of branch-and-bound nodes. All averages are taken over the set of instances solved within the time limit. The different tables show the computational results for the three different values of ε and b. From the results we can see that `FW-BB` suffers with the increasing of the right hand side b. We can observe that, in the pure integer case, `FW-BB-G` is able to solve the most instances within the time limit, while for the mixed-integer case, it is `FW-BB-P` that is able to solve the largest number of instances. When the number of solved instances is the same, both version of `FW-BB` outperform the SOCP solver of `CPLEX 12.6`. We also observe that `FW-BB-G` is the best in the pure integer case, while `FW-BB-P` is the fastest in the mixed-integer case.

In our experiments, we noticed that in some cases `FW-BB` and `CPLEX` provide slightly different minimizers, yielding slightly different optimal objective function values. While for certain instances the optimal solution of `FW-BB` is slightly superior to `CPLEX`, for other instances it is the other way round. We observed a relative difference from the best solution of the order of 10^{-5} and 10^{-3}, respectively for the pure and the mixed-integer case.

Table 7.4: Results for the pure integer instances with $h(t) = \sqrt{\frac{1-\varepsilon}{\varepsilon}}\; t$ and $\varepsilon \in \{0.91, 0.95, 0.99\}$, running `FW-BB-P`, `FW-BB-P` and `CPLEX 12.6`.

inst			`FW-BB-P`			`FW-BB-G`			`CPLEX 12.6`		
n	ε	b	#	time	nodes	#	time	nodes	#	time	nodes
50	0.91	b_1	10	0.10	5.9e+03	10	0.14	6.0e+03	10	3.78	1.4e+04
50	0.91	b_2	10	20.24	6.7e+05	10	4.01	2.1e+05	10	720.33	9.2e+05
50	0.91	b_3	7	3.30	1.3e+05	9	15.51	7.1e+05	7	532.02	6.4e+05
50	0.95	b_1	10	0.08	5.7e+03	10	0.08	5.8e+03	10	0.68	4.0e+03
50	0.95	b_2	10	5.25	5.4e+05	10	5.91	5.1e+05	10	501.38	1.3e+06
50	0.95	b_3	8	104.49	8.8e+06	10	15.50	5.5e+05	9	483.39	9.5e+05
50	0.99	b_1	10	0.03	5.3e+03	10	0.02	5.3e+03	10	0.03	2.4e+02
50	0.99	b_2	10	10.62	2.6e+06	10	10.39	2.6e+06	9	0.53	5.5e+03
50	0.99	b_3	10	7.71	1.5e+06	10	8.60	1.5e+06	10	14.37	1.7e+05
75	0.91	b_1	10	2.13	4.4e+04	10	2.45	4.3e+04	9	270.65	1.8e+05
75	0.91	b_2	9	342.27	1.5e+07	10	100.53	3.4e+06	6	1367.35	8.7e+05
75	0.91	b_3	5	24.30	5.3e+05	9	582.18	5.3e+06	6	1064.61	1.4e+06
75	0.95	b_1	10	0.36	1.9e+04	10	0.34	1.9e+04	10	16.17	3.4e+04
75	0.95	b_2	10	3.79	2.4e+05	10	8.95	2.6e+05	9	176.86	3.3e+05
75	0.95	b_3	7	97.13	3.7e+05	10	323.10	1.1e+06	7	632.53	7.6e+05
75	0.99	b_1	10	0.40	2.7e+04	10	0.38	2.7e+04	10	1.37	5.8e+03
75	0.99	b_2	10	2.77	2.5e+05	10	2.68	2.5e+05	10	8.29	8.6e+04
75	0.99	b_3	10	1.41	2.2e+04	10	1.45	2.2e+04	10	0.52	4.4e+03
100	0.91	b_1	10	10.98	2.5e+05	10	14.03	2.5e+05	6	754.15	5.6e+05
100	0.91	b_2	8	16.59	4.7e+05	10	59.60	9.3e+05	4	543.38	4.3e+05
100	0.91	b_3	6	94.64	3.7e+06	9	566.18	3.3e+06	4	885.57	9.4e+05
100	0.95	b_1	10	6.09	1.1e+05	10	6.59	1.1e+05	9	157.28	1.9e+05
100	0.95	b_2	10	18.17	1.6e+06	10	27.56	1.8e+06	10	423.91	4.7e+05
100	0.95	b_3	6	85.64	5.6e+05	8	505.27	3.5e+06	5	130.69	1.4e+05
100	0.99	b_1	9	0.95	7.8e+04	10	8.65	1.2e+06	9	1.48	1.1e+04
100	0.99	b_2	10	4.28	2.2e+05	10	4.24	2.2e+05	10	1.70	1.7e+04
100	0.99	b_3	9	11.30	5.2e+05	9	11.70	5.2e+05	10	34.71	1.4e+05

Comparison on instances with $h(t) = t^2$. In this case our Problem GCBP reduces to a convex quadratic mixed-integer problem, and the objective function is differentiable everywhere in the feasible set. On Table 7.6 and Table 7.7 we report the comparison among `FW-BB-P`, `FW-BB-G`, the MIQP solver of `CPLEX 12.6` and the branch-and-bound solver of `Bonmin 1.8.1`. We considered $\Omega = 1$. All the algorithms were able to solve all the instances very quickly. The MIQP solver of `CPLEX 12.6` shows the best cpu-times, although `FW-BB` is also very fast even if it enumerates a higher number of nodes. It is not surprising that `Bonmin` is not competitive, since it is not tailored for quadratic objective functions.

Table 7.5: Results for the mixed-integer instances with $n_1 = \lfloor \frac{n}{2} \rfloor$ and $h(t) = \sqrt{\frac{1-\varepsilon}{\varepsilon}}\, t$ and $\varepsilon \in \{0.91, 0.95, 0.99\}$, running `FW-BB-P`, `FW-BB-P` and `CPLEX 12.6`.

inst			`FW-BB-P`			`FW-BB-G`			`CPLEX 12.6`		
n	ε	b	#	time	nodes	#	time	nodes	#	time	nodes
100	0.91	b_1	10	0.13	5.3e+02	10	0.96	1.6e+03	10	17.00	3.8e+03
100	0.91	b_2	10	0.22	7.7e+02	10	0.37	7.3e+02	10	279.15	7.9e+03
100	0.91	b_3	10	2.63	4.2e+02	10	4.28	7.0e+02	3	51.01	2.8e+03
100	0.95	b_1	10	0.07	2.6e+02	10	0.09	2.6e+02	10	1.89	4.7e+02
100	0.95	b_2	10	0.21	3.0e+02	10	0.37	3.2e+02	10	59.10	3.0e+03
100	0.95	b_3	10	3.35	5.3e+02	10	4.08	5.9e+02	5	364.90	4.4e+03
100	0.99	b_1	10	0.04	1.8e+02	10	0.05	1.8e+02	10	0.15	3.7e+01
100	0.99	b_2	10	0.20	7.3e+02	10	0.21	7.6e+02	10	0.70	1.9e+02
100	0.99	b_3	10	80.69	7.2e+03	10	168.03	2.7e+04	9	503.62	1.0e+04
150	0.91	b_1	10	1.06	2.4e+03	10	8.21	1.1e+04	10	52.53	3.2e+03
150	0.91	b_2	9	1.27	1.6e+03	10	294.82	4.9e+04	6	707.94	6.7e+03
150	0.91	b_3	10	10.09	1.1e+03	9	12.80	1.3e+03	5	47.32	1.8e+03
150	0.95	b_1	10	0.41	1.1e+03	10	0.54	1.1e+03	10	11.49	1.0e+03
150	0.95	b_2	10	1.19	1.3e+03	10	21.34	5.8e+03	8	225.78	3.0e+03
150	0.95	b_3	10	57.69	7.7e+03	10	142.30	1.2e+04	5	834.79	6.2e+03
150	0.99	b_1	10	0.15	2.5e+02	10	0.18	2.6e+02	10	35.08	5.8e+02
150	0.99	b_2	10	1.04	4.7e+02	10	1.54	6.2e+02	10	6.69	6.1e+02
150	0.99	b_3	10	10.00	1.3e+03	9	10.78	1.4e+03	9	422.15	3.5e+03
200	0.91	b_1	10	1.63	3.6e+03	10	8.94	1.2e+04	10	465.62	9.5e+03
200	0.91	b_2	9	11.92	5.9e+03	8	65.36	1.8e+04	3	879.46	9.1e+03
200	0.91	b_3	10	142.27	1.3e+04	9	349.51	2.7e+04	3	204.92	3.8e+03
200	0.95	b_1	10	0.86	1.3e+03	10	1.30	1.4e+03	10	75.50	3.5e+03
200	0.95	b_2	10	3.56	2.3e+03	10	5.94	9.9e+02	5	44.64	1.6e+03
200	0.95	b_3	9	146.75	1.4e+04	6	16.54	2.4e+03	7	38.77	2.4e+03
200	0.99	b_1	10	2.63	1.6e+04	10	2.65	1.6e+04	10	2.44	2.8e+03
200	0.99	b_2	10	2.15	4.9e+02	10	5.25	8.3e+02	9	277.08	2.6e+03
200	0.99	b_3	10	49.84	6.1e+03	10	168.41	2.2e+04	9	183.80	2.1e+03

Table 7.6: Results for the pure integer instances with $h(t) = t^2$, running `FW-BB-P`, `FW-BB-G`, `CPLEX 12.6` and `B-BB 1.8.1`.

inst		`FW-BB-P`		`FW-BB-G`		`CPLEX 12.6`		`B-BB 1.8.1`	
n	b	time	nodes	time	nodes	time	nodes	time	nodes
50	b_1	0.02	1.7e+03	0.01	1.7e+03	0.02	3.3e+01	2.05	72
50	b_2	0.03	1.9e+03	0.01	1.9e+03	0.01	6.1e+01	3.16	119
50	b_3	0.05	1.9e+03	0.05	1.9e+03	0.02	5.8e+01	3.70	151
75	b_1	0.12	9.4e+03	0.12	9.4e+03	0.03	7.5e+01	7.31	174
75	b_2	0.13	5.6e+03	0.14	5.6e+03	0.03	6.1+e01	5.41	98
75	b_3	0.22	5.4e+03	0.22	5.4e+03	0.03	7.3e+01	6.29	129
100	b_1	0.24	9.6e+03	0.24	9.4e+03	0.05	9.8e+01	14.44	195
100	b_2	0.28	8.3e+03	0.29	8.3e+03	0.05	8.8e+01	15.93	206
100	b_3	1.27	3.9e+04	1.27	3.9e+04	0.04	6.6e+01	13.04	143

Table 7.7: Results for the mixed-integer instances with $n_1 = \lfloor \frac{n}{2} \rfloor$ and $h(t) = t^2$, running `FW-BB-P`, `FW-BB-G`, `CPLEX 12.6` and `B-BB 1.8.1`.

inst		`FW-BB-P`		`FW-BB-G`		`CPLEX 12.6`		`B-BB 1.8.1`	
n	b	time	nodes	time	nodes	time	nodes	time	nodes
100	b_1	0.07	4.1e+02	0.07	4.1e+02	0.04	1.1e+01	2.69	17
100	b_2	0.08	6.4e+02	0.09	6.4e+02	0.03	1.6e+01	3.01	28
100	b_3	0.33	9.3e+02	0.33	9.3e+02	0.03	2.6e+01	4.28	44
150	b_1	0.21	7.6e+02	0.22	7.6e+02	0.06	1.9e+01	13.96	53
150	b_2	0.29	9.7e+02	0.29	9.6e+02	0.06	2.1e+01	23.57	69
150	b_3	0.33	8.5e+02	0.34	8.5e+02	0.06	9.9e+00	17.92	30
200	b_1	0.79	3.3e+03	0.79	3.3e+03	0.11	2.1e+01	75.33	173
200	b_2	1.60	5.2e+03	1.65	5.2e+03	0.11	1.9e+01	80.79	101
200	b_3	0.70	1.5e+03	0.72	1.5e+03	0.12	2.4e+01	90.56	122

7.3.3 Using a Different Risk-adjusted Function

As a further experiment, we tested the pure integer instances considering a different risk-adjusted function $h_{exp} : \mathbb{R}^+ \to \mathbb{R}$, namely

$$h_{exp}(t) = \begin{cases} 0 & t \leq \gamma \\ \exp(t-\gamma) - (t-\gamma+1) & t > \gamma, \end{cases}$$

so that the investor's risk-aversion increases exponentially in the magnitude of the risk.

In Table 7.8, we report the results of `FW-BB-G`, considering three different values for the parameter γ, namely $\gamma \in \{0, 1, 10\}$. We observe that for $\gamma = 0$ and $\gamma = 1$, `FW-BB-G` is able to solve all the instances within the time limit and that the bigger the γ the more difficult are the instances for `FW-BB-G`.

In order to have an idea of how the risk-adjusted function influences the results in terms of objective function value, we report in Table 7.9 the results obtained on a pure integer instance of dimension $n = 50$, under the constraint $a^\top x \leq b_1$. In the table we report, for each risk-adjusted function $h(t)$, depending on a specific risk parameter (risk-par), the value of the objective function obtained ($f^\star$), the value of the return term in the objective function ($r^\top x^\star$), the number of the non-zero entries in the optimal solution ($|\text{supp}(x)|$) and the infinity norm of the optimal solution ($\|x^\star\|_\infty$). We notice that the lowest value for the objective function value and the return is obtained by considering $h(t) = t^2$. Concerning the linear risk-adjusted function $h(t) = \Omega\, t$, we can observe that we get higher returns for higher values of ϵ. For what concerns the risk-adjusted function $h_{exp}(t)$, we can observe that we get higher returns the higher is the value of γ. Both of these behaviors suggest that the more risk the investor is willing to take, the better can be the

Table 7.8: Results for the pure integer instances with $h(t) = \tilde{h}(t)$ and $\gamma \in \{0, 1, 10\}$, running `FW-BB-G`.

inst		$\gamma = 0$			$\gamma = 1$			$\gamma = 10$		
n	b	#	time	nodes	#	time	nodes	#	time	nodes
50	b_1	10	0.05	2.2e+03	10	0.64	1.3e+04	10	0.04	2.1e+03
50	b_2	10	0.04	1.0e+03	10	0.20	4.0e+03	8	33.25	6.2e+05
50	b_3	10	0.10	1.3e+03	10	2.31	2.0e+04	7	211.84	1.6e+05
75	b_1	10	0.20	4.5e+03	10	5.31	1.0e+05	9	37.38	4.8e+04
75	b_2	10	0.13	2.1e+03	10	1.38	2.2e+04	3	21.66	3.4e+05
75	b_3	10	0.47	5.9e+03	10	5.83	1.9e+04	5	441.38	1.9e+05
100	b_1	10	0.37	6.9e+03	10	17.72	3.0e+05	7	228.46	2.5e+06
100	b_2	10	0.54	5.4e+03	10	4.14	2.0e+04	6	176.83	7.1e+04
100	b_3	10	3.06	1.4e+04	10	32.37	5.0e+04	5	1352.24	5.3e+05

Table 7.9: Results for one instance with $n = 50$, $b = b_3$ and different risk-functions, running `FW-BB-G`.

model		solution characteristics			
$h(t)$	risk-par	$f(x^\star)$	$r^\top x^\star$	$\|\|x^\star\|\|_\infty$	$\|\text{supp}(x^\star)\|$
Ωt	$\varepsilon = 0.91$	-0.05684	-0.3438	14	10
Ωt	$\varepsilon = 0.95$	-0.15883	-0.5471	28	9
Ωt	$\varepsilon = 0.99$	-1.76778	-4.8224	613	2
Ωt^2	$\Omega = 1$	-0.03672	-0.0784	2	9
$\tilde{h}(t)$	$\gamma = 0$	-0.06768	-0.1311	4	11
$\tilde{h}(t)$	$\gamma = 1$	-0.40489	-0.4341	20	10
$\tilde{h}(t)$	$\gamma = 10$	-2.03201	-2.0429	158	3

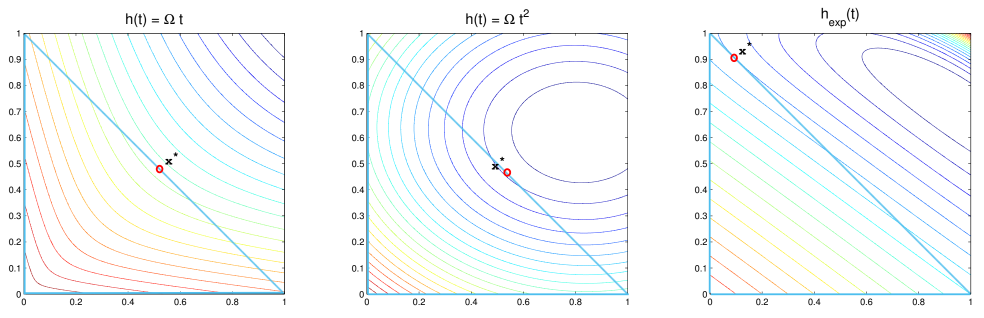

Figure 7.2: Contour plots of $f = h(\sqrt{x^\top M x}) - r^\top x$ for different risk-adjusted functions.

return he gets. Furthermore, we observe a smaller number of non-zero components in the optimal solution $x^\star$, as well as a higher value of the infinity norm of $x^\star$ if the investor is becoming less risk-averse. A possible explanation is that allowing more risk results in a less diversified portfolio. In Figure 7.2, we show the contour plots for different risk-adjusted functions $h(t)$.

7.4 Conclusion

In this chapter we have designed a new branch-and-bound algorithm for convex mixed-integer minimization problems. Its effectiveness relies on the rapid enumeration of the search tree and combines ideas of early pruning, warmstart features and quick incremental computations. The branch-and-bound algorithm has been tested on a set of real-world instances for the capital budgeting problem. Different risk-adjusted functions have been modeled and considered. Experimental results show that the approach proposed significantly outperforms the SOCP solver of `CPLEX 12.6` for instances where a linear risk-adjusted function is considered. In particular, the results also confirm the remarkable benefits of the non-monotone Frank-Wolfe version to solve the node relaxations.

Summary and Outlook

This thesis deals with exact algorithms for convex mixed-integer nonlinear optimization problems. We proposed three branch-and-bound algorithms that all follow and extend the same ideas of the basic branch-and-bound scheme proposed by Buchheim et al. for the solution of box-constrained convex mixed-integer programming problems. The latter algorithm was designed to give a quick enumeration of the nodes rather than computing strong dual bounds. To process each node as fast as possible, we focused on special tailored algorithms from nonlinear programming to compute the dual bounds effectively. In many cases a reengineering of the basic algorithms leads to good overall performance of the branch-and-bound algorithms.

After introducing the basics of nonlinear and integer programming, we summarized the most relevant algorithmic and theoretical advances in the form of an overview of existing solution methods and software for convex mixed-integer nonlinear programming problems in Chapter 3. Based on the branch-and-bound scheme for convex quadratic mixed-integer programming problems in Chapter 4, we proposed the branch-and-bound algorithm `FAST-MIQPA-BB` using a specialized active set method for the solution of convex mixed-integer quadratic minimization problems with linear inequalities in Chapter 5. While common QP-based branch-and-bound algorithms tackle this problem by simply solving the QP relaxation in the nodes, we addressed the problem of computing dual bounds by solving the dual problems of the QP relaxations. This approach has the big advantage that valid lower bounds are automatically generated as long as feasibility is obtained, even if the problem is solved only approximately. In many cases an approximate solution was already sufficient to prune the node such that the average running per node reduced considerably. The dual problem was solved by the specialized active set algorithm `FAST-QPA` that fully exploited the structure of the underlying quadratic program. As a third key component, we focused on warmstart procedures to decrease the average running times per node. Exploiting the fact that the node relaxations are highly related to each other in the sense that they have the same form and differ only slightly in the problem data, we took over the node solutions from the parent node as a starting point for our active set algorithm. Our experiments on randomly generated instances showed that our algorithm is very competitive to the state-of-the art MIQP solver of `CPLEX 12.6` in case the number of linear inequalities is moderate.

The second branch-and-bound algorithm was used within the quadratic outer approximation scheme of Chapter 6 to solve box-constrained convex mixed-integer minimization problems. We introduced an extension of the classical linear outer approximation that uses quadratic underestimators instead of linearizations. In every iteration a modified version of our branch-and-bound scheme is used to solve a box-constrained convex mixed-integer quadratic problem within the quadratic outer approximation scheme. The surrogate problems have been tackled by a decomposition approach that reduced the convex linearly constrained mixed-integer nonlinear programming problem to a series of unconstrained problems of the same form. It turned out that the quadratic underestimators have the potential to approximate the original objective function much better than the standard supporting hyperplanes, yielding a faster convergence. The disadvantage of the additional computational effort of minimizing a piecewise convex quadratic function instead of a piecewise linear function was faced by requiring the quadratic underestimators to have the same Hessian. This allowed us to reduce the problem to a sequence of convex mixed-integer quadratic minimization problems that can be solved effectively by our common branch-and-bound framework. For a class of exponential functions, it turned out that the usage of quadratic outer approximation is useful. In numerical experiments we showed that the scheme is superior to `B-BB`, one of the state-of-the art branch-and-bound solvers for general convex mixed-integer problems.

In Chapter 7, we showed the potential of quick branch-and-bound schemes to solve real-world applications. We considered a generalized version of the classic mean-variance capital budgeting problem and proposed a modified Frank-Wolfe type algorithm to compute the dual bounds. Using several improvements, we were able to quickly find optimal solutions. Applying it to real-world capital market indices, taken from historical Standard and Poor's indices, we could solve pure integer instances with up to 100 variables within a time limit of one hour. Comparing it to `CPLEX 12.6` and `Bonmin 1.8.1` confirmed the effectiveness of our approach.

Based on the results of this thesis, we want to suggest possible research directions. First of all, it is worth to analyze to which classes of general convex mixed-integer programming problems the dual bound computation approach of `FAST-QPA-BB` can be adapted. This is not straightforward because of mainly two reasons. The first one is that the computation of the dual relaxations requires a suitable algorithm. Since the dual problem is a bound-constrained nonlinear problem (see Section 2.3), one possibility is to use the tailored feasible active set method by De Santis et al. [65]. This algorithm is a generalization of `FAST-QPA` and seems therefore promising to provide good results. The more difficult reason is due to the crucial task of recovering the primal solution from the dual. In fact, this is not always possible by a closed formula and in general requires the solution of a system of nonlinear equations. In case of MIQP this was straightforward since the stationarity condition of the KKT-system is a system of linear equations.

Moreover, our elementary work on the quadratic outer approximation scheme leads to several natural and possible improvements. First of all, we need to figure out how the quadratic underestimator can be chosen automatically to use it in practice. One possibility is to study results on the estimation of the eigenvalues of the hessian. For example in case Q is an interval matrix one might use techniques inspired by the ideas of Adjiman et al. used in the design of the solver α-BB [4, 5] to automatically generate valid convex underestimators. Besides this key questions, there are two main directions to study. It would be interesting to generalize our quadratic outer approximation scheme to general constrained convex mixed-integer optimization problems by also outer approximating the feasible region by supporting quadratic functions instead of hyperplanes. This would lead to another challenging task of designing an algorithm that can handle the underlying mixed-integer quadratically constrained quadratic problems for solving the subproblems. Alternatively, a penalty approach might be used to reduce the problem to an unconstrained problem since a convex differentiable penalty term would not affect the validity of the quadratic underestimator. A first generalization could be the investigation of linear inequalities. Another idea is to study ways of discarding underestimators in the case they are not needed in the current iteration, since the number of active underestimators plays a crucial role for the overall running time. Finally, a novel approach would be to use non-convex quadratic underestimators. The resulting surrogate problem then becomes a non-convex quadratic mixed-integer problem that can be handled for example by the exact algorithm of Buchheim et al. [44].

Concerning the results of Chapter 7, it seems promising to enhance the Frank-Wolfe method to handle general nonlinear convex objective functions. Thus, we would get a powerful algorithm for handling convex mixed-integer nonlinear programming problems. Finally it would be interesting to apply our branch-and-bound scheme to further applications, especially real-world problems that can be modeled as convex MINLP problems with one linear constraint.

References

[1] K. Abhishek, S. Leyffer, and J.T. Linderoth. FilMINT: An outer approximation-based solver for convex mixed-integer nonlinear programs. *INFORMS Journal on Computing*, 22(4):555–567, 2010.

[2] T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[3] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[4] C.S. Adjiman, S. Dallwig, C.A. Floudas, and A Neumaier. A global optimization method, alpha BB, for general twice-differentiable constrained NLPs - I. Theoretical advances. *Computers & Chemical Egenineering*, 22: 1137–1158, 1998.

[5] C.S. Adjiman, S. Dallwig, C.A. Floudas, and A Neumaier. A global optimization method, alpha BB, for general twice-differentiable constrained NLPs - II. Implementation and computational results. *Computers & Chemical Egenineering*, 22:1159–1179, 1998.

[6] S.C. Agrawal. On mixed integer quadratic programs. *Naval Research Logistics Quarterly*, 21(2):289–297, 1974.

[7] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, 2002.

[8] F.A. Al-Khayyal and C. Larsen. Global optimization of a quadratic function subject to a bounded mixed integer consraint set. *Annals of Operations Research*, 25(1):169–180, 1990.

[9] W. Alt. *Nichtlineare Optimierung: Eine Einführung in Theorie, Verfahren und Anwendungen.* Vieweg+Teubner Verlag, 2011.

[10] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. On the solution of traveling salesman problems. *Documenta Mathematica*, 3:645–656, 1998.

[11] A. Atamtürk and V. Narayanan. Polymatroids and mean-risk minimization in discrete optimization. *Operations Research Letters*, 36(5):618–622, 2008.

[12] E. Balas. Intersection cuts–A new type of cutting planes for integer programming. *Operations Research*, 19(1):19–39, 1971.

[13] E. Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5: 3–51, 1979.

[14] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58(1–3): 295–324, 1993.

[15] F. Baumann, S. Berckey, and C. Buchheim. Exact algorithms for combinatorial optimization problems with submodular objective functions. In Michael Jnger and Gerhard Reinelt, editors, *Facets of Combinatorial Optimization*, pages 271–294. Springer Berlin Heidelberg, 2013.

[16] F. Baumann, C. Buchheim, and A. Ilyina. Lagrangean decomposition for mean-variance combinatorial optimization. In *Combinatorial Optimization - Third International Symposium, ISCO 2014*, pages 62–74, 2014.

[17] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming - Theory and Algorithms.* Wiley, 1993.

[18] E.M.L. Beale. On quadratic programming. *Naval Research Logistics*, 3(6): 227–243, 1959.

[19] P. Belotti, C. Kirches, S. Leyffer, J.T. Linderoth, J. Luedtke, and A. Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

[20] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications.* Society for Industrial and Applied Mathematics, 2001.

[21] M. Benichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

[22] T. Berthold, A. Gleixner, S. Heinz, T. Koch, and S. Vigerske. Extending SCIP for solving MIQCPs. In *European Workshop on Mixed Integer Nonlinear Programming (EWMINLP)*, pages 181–196, 2010.

[23] T. Berthold, G. Gamrath, A. Gleixner, S. Heinz, T. Koch, and Y. Shinano. Solving mixed integer linear and nonlinear problems using the SCIP optimization suite. Technical report, ZIB, 2012.

[24] D.P. Bertsekas. Projected Newton methods for optimization problems with simple constraints. *SIAM Journal on Control and Optimization*, 20(2): 221–246, 1982.

[25] D.P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods.* Academic Press, 1982.

[26] D.P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 1999.

[27] D. Bertsimas and I. Popescu. Optimal inequalities in probability theory: A convex optimization approach. *SIAM Journal on Optimization*, 15(3): 780–804, 2005.

[28] D. Bertsimas and R. Shioda. Algorithm for cardinality-constrained quadratic optimization. *Computational Optimization and Applications*, 43 (1):1–22, 2009.

[29] D. Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74(2):121–140, 1996.

[30] J.R. Birge and F. Louveaux. *Introduction to Stochastic Programming.* Springer, 1997.

[31] R.E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

[32] P.T. Boggs and J.W. Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1 1995.

[33] P. Bonami and J.P.M. Gonçalves. Heuristics for convex mixed integer nonlinear programs. *Computational Optimization and Applications*, 51(2):729–747, 2012.

[34] P. Bonami, L.T. Biegler, A.R. Conn, G. Cornuéjols, I.E. Grossmann, C.D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008.

[35] P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119(2): 331–352, 2009.

[36] P. Bonami, M. Kilinç, and J.T. Linderoth. *Algorithms and Software for Solving Convex Mixed Integer Nonlinear Programs*, volume 154 of *IMA Volumes in Mathematics and its Applications: Mixed Integer Nonlinear Programming*, chapter Part I: Convex MINLP, pages 1–39. Springer, 2012.

[37] bonmin. Bonmin 1.6.0: Basic open-source nonlinear mixed integer programming, 2015. www.coin-or.org/Bonmin.

[38] R.R. Boorstyn and H. Frank. Large-scale network topological optimization. *IEEE Transactions on Communications*, 25(1):29–47, 1977.

[39] B. Borchers and J.E. Mitchell. An improved branch and bound algorithm for mixed integer nonlinear programs. *Computers and Operations Research*, 21(4):359–367, 1994.

[40] J. Boutros, N. Gresset, L. Brunel, and M.P.C. Fossorier. Soft-input soft-output lattice sphere decoder for linear channels. In *Global Telecommunications Conference, GLOBECOM 2003*, pages 1583–1587, 2003.

[41] C. Buchheim and L. Trieu. Quadratic outer approximation for convex integer programming with box constraints. In *Experimental Algorithms, 12th International Symposium, SEA 2013*, pages 224–235, 2013.

[42] C. Buchheim and L. Trieu. Active set methods with reoptimization for convex quadratic integer programming. In *Combinatorial Optimization - Third International Symposium, ISCO 2014*, pages 125–136, 2014.

[43] C. Buchheim, A. Caprara, and A. Lodi. An effective branch-and-bound algorithm for convex quadratic integer programming. *Mathematical Programming*, 135(1-2):369–395, 2012.

[44] C. Buchheim, M. De Santis, L. Palagi, and M. Piacentini. An exact algorithm for nonconvex quadratic integer minimization using ellipsoidal relaxations. *SIAM Journal on Optimization*, 23(3):1867–1889, 2013.

[45] S. Burer and A.N. Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012.

[46] M.R. Bussieck and S. Vigerske. *MINLP Solver Software*. John Wiley & Sons, Inc., 2010.

[47] R.H. Byrd, J. Nocedal, and R.A. Waltz. Knitro: An integrated package for nonlinear optimization. In G. Di Pillo and M. Roma, editors, *Large-Scale Nonlinear Optimization*, volume 83, pages 35–59. Springer US, 2006.

[48] S. Callegari, F. Bizzarri, R. Rovatti, and G. Setti. On the approximate solution of a class of large discrete quadratic programming problems by $\delta\sigma$ modulation: The case of circulant quadratic forms. *IEEE Transactions on Signal Processing*, 58(12):6126–6139, 2010.

[49] A. Caprara and M. Fischetti. $\{0, 1/2\}$-chvátal-gomory cuts. *Mathematical Programming*, 74(3):221–235, 1996.

[50] I. Castillo, J. Westerlund, S. Emet, and T. Westerlund. Optimization of block layout design problems with unequal areas: A comparison of MILP and MINLP optimization methods. *Computers & Chemical Engineering*, 30(1):54–69, 2005.

[51] F. Cesarone, A. Scozzari, and F. Tardella. Efficient algorithms for mean-variance portfolio optimization with hard real-world constraints. *Giornale dell'Istituto Italiano degli Attuari*, 72:37–56, 2009.

[52] X-W. Chang and Q. Han. Solving box-constrained integer least squares problems. *IEEE Transactions on Wireless Communications*, 7(1):277–287, 2008.

[53] L. Condat. Fast projection onto the simplex and the l1 ball. Technical report, GIPSA-lab, University of Grenoble, 2014.

[54] W. Cook, R. Kannan, and A. Schrijver. Chvátal closures for mixed integer programming problems. *Mathematical Programming*, 47(1–3):155–174, 1990.

[55] G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112(1):3–44, 2008.

[56] G. Cornuéjols and R. Tütüncü. *Optimization methods in finance.* Cambridge University Press, 2006.

[57] cplex. IBM ILOG CPLEX optimizer 12.6, 2015. www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[58] A. Czajkowski. Beschränktheit ganzzahliger konvex-quadratischer Optimierungsprobleme mit Eigenwert Null. *TU Dortmund, Diplomathesis*, 2011.

[59] R.J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8:250–255, 1965.

[60] C. D'Ambrosio and A. Lodi. Mixed integer nonlinear programming tools: an updated practical overview. *Annals of Operations Research*, 204(1):301–320, 2013.

[61] M.O. Damen, H. El Gamal, and G. Caire. On maximum-likelihood detection and the search for the closest lattice point. *IEEE Transactions on Information Theory*, 49(10):2389–2402, 2003.

[62] G. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.

[63] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2:393–410, 1954.

[64] M. D'Apuzzo and M. Marino. Parallel computational issues of an interior point method for solving large bound-constrained quadratic programming problems. *Parallel Computing*, 29(4):467–483, 2003.

[65] M. De Santis, G. Di Pillo, and S. Lucidi. An active set feasible method for large-scale minimization problems with bound constraints. *Computational Optimization and Applications*, 53(2):395–423, 2012.

[66] R.S. Dembo, S.C. Eisenstat, and T. Steinhaug. Inexact Newton methods. *SIAM Journal on Numerical Analysis*, 19:400–408, 1982.

[67] J.E. Dennis and R.B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice Hall, 1983.

[68] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[69] W.S. Dorn. A duality theorem for convex programs. *IBM J. Res. Dev.*, 4 (4):407–413, 1960.

[70] Z. Dostál. *Optimal Quadratic Programming Algorithms: With Applications to Variational Inequalities.* Springer Publishing Company, Incorporated, 2009.

[71] Z. Dostál and J. Schöberl. Minimizing quadratic functions subject to bound constraints with the rate of convergence and finite termination. *Computational Optimization and Applications*, 30(1):23–43, 2005.

[72] A.S. Drud. CONOPT–A large-scale GRG code. *ORSA Journal on Computing*, 6(2):207–216, 1994.

[73] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning, ICML 2008*, pages 272–279, 2008.

[74] J.C. Dunn. Convergence rates for conditional gradient sequences generated by implicit step length rules. *SIAM Journal on Control and Optimization*, 18(5):473–487, 1980.

[75] M.A. Duran and I.E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36 (3):307–339, 1986.

[76] A.M. Eliceche, S.M. Corvalán, and P. Martínez. Environmental life cycle impact as a tool for process optimisation of a utility plant. *Computers & Chemical Engineering*, 31(56):648–656, 2007.

[77] F. Facchinei and S. Lucidi. Quadratically and superlinearly convergent algorithms for the solution of inequality constrained minimization problems. *Journal of Optimization Theory and Applications*, 85(2):265–289, 1995.

[78] fico. FICO Xpress Optimization Suite, 2015. www.fico.com.

[79] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–471, 1985.

[80] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66(3):327–349, 1994.

[81] R. Fletcher and S. Leyffer. Numerical experience with lower bounds for miqp branch-and-bound. *SIAM Journal on Optimization*, 8(2):604–616, 1998.

[82] R. Fletcher and S. Leyffer. User manual for filterSQP. Technical report, University of Dundee, 1998.

[83] A. Flores-Tlacuahuac and L.T. Biegler. Simultaneous mixed-integer dynamic optimization for integrated design and control. *Computers & Chemical Engineering*, 31(56):588–600, 2007.

[84] A. Frangioni and C. Gentile. Perspective cuts for a class of convex 0-1 mixed integer programs. *Mathematical Programming*, 106(2):225–236, 2006.

[85] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[86] R.M. Freund and P. Grigas. New analysis and results for the Frank-Wolfe method. *Mathematical Programming*, pages 1–32, 2014.

[87] gams. GAMS: General algebraic modeling system, 2015. www.gams.com/.

[88] gams-solver-manuals. GAMS – the solver manuals, 2015. www.gams.com/dd/docs/allsolvers.pdf.

[89] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[90] C. Geiger and C. Kanzow. *Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben.* Springer-Verlag Berlin Heidelberg, 1999.

[91] C. Geiger and C. Kanzow. *Theorie und Numerik restringierter Optimierungsaufgaben.* Springer-Verlag Berlin Heidelberg, 2002.

[92] A. Geoffrion. Generalized Benders Decomposition. *Journal of Optimization*, 10:237–260, 1972.

[93] P. Gill, W. Murray, and M. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[94] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27(1):1–33, 1983.

[95] D. Goldfarb and S. Liu. An $O(n^3L)$ primal interior point algorithm for convex quadratic programming. *Mathematical Programming*, 49(1-3):325–340, 1990.

[96] R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Society*, 64:275–278, 1958.

[97] R.E. Gomory. An algorithm for the mixed integer problem. Technical report, The RAND Corporation, 1960.

[98] L. Grippo and M. Sciandrone. Nonmonotone globalization techniques for the barzilai-borwein gradient method. *Computational Optimization and Applications*, 23:143–169, 2002.

[99] L. Grippo and M. Sciandrone. Nonmonotone globalization techniques, for the Barzilai-Borwein gradient method. *Computational Optimization and Applications*, 23(2):143–169, 2002.

[100] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23(4): 707–716, 1986.

[101] L. Grippo, F. Lampariello, and S. Lucidi. A truncated Newton method with nonmonotone line search for unconstrained optimization. *Journal of Optimization Theory and Applications*, 60(3):401–419, 1989.

[102] I.E. Grossmann and N.V. Sahinidis. Special issue on mixed integer programming and its applications to engineering, part I. *Optimization and Engineering*, 3(4), 2002.

[103] I.E. Grossmann and N.V. Sahinidis. Special issue on mixed integer programming and its applications to engineering, part II. *Optimization and Engineering*, 4(1), 2002.

[104] J. Guélat and P. Marcotte. Some comments on Wolfe's "away step". *Mathematical Programming*, 35(1):110–119, 1986.

[105] O.K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546, 1985.

[106] gurobi. GUROBI Optimizer, 2015. www.gurobi.com.

[107] A. Hassibi and S. Boyd. Integer parameter estimation in linear models with applications to gps. *IEEE Transactions on Signal Processing*, 46(11): 2938–2952, 1998.

[108] M. Heinkenschloss, M. Ulbrich, and S. Ulbrich. Superlinear and quadratic convergence of affine-scaling interior-point Newton methods for problems with simple bounds without strict complementarity assumption. *Mathematical Programming*, 86(3):615–635, 1999.

[109] M. Held, P. Wolfe, and H. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974.

[110] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436, 1952.

[111] H. Hijazi, P. Bonami, and A. Ouorou. An outer-inner approximation for separable mixed-integer nonlinear programs. *INFORMS Journal on Computing*, 26(1):31–44, 2014.

[112] R. Hübner. *A level set approach to integer nonlinear optimization*. PhD thesis, University of Göttingen, 2013.

[113] P. Hungerländer and F. Rendl. A feasible active set method for convex problems with simple bounds. Technical report, Alpen-Adria-Universitaet Klagenfurt, 2013.

[114] H. Ishii and T. Matsutomi. Confidence regional method of stochastic spanning tree problem. *Mathematical and Computer Modelling*, 22(1012):77–82, 1995.

[115] M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, pages 427–435, 2013.

[116] R.C. Jeroslow. There cannot be any algorithm for integer programming with quadratic constraints. *Operations Research*, 21:221–224, 1973.

[117] B.A. Julstrom. Greedy, genetic, and greedy genetic algorithms for the quadratic knapsack problem. In *Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 607–614, 2005.

[118] R. Kannan and C. Momma. On the computational complexity of integer programming problems. In R. Henn, B. Korte, and W. Oettli, editors, *Optimization and Operations Research*, pages 161–172. Springer, 1978.

[119] S. Kapoor and P.M. Vaidya. Fast algorithms for convex quadratic programming and multicommodity flows. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, STOC 1986*, pages 147–159, 1986.

[120] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.

[121] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[122] J.E. Kelley. The cutting plane method for solving convex programs. *Journal of the SIAM*, 8:703–712, 1960.

[123] L.G. Khachiyan. Polynomial algorithms in linear programming. {*USSR*} *Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[124] K.C. Kiwiel. *Methods of descent for nondifferentiable optimization.* Springer-Verlag, 1985.

[125] V. Klee and G.J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*, pages 159–175. Academic Press, New York, 1972.

[126] G.R. Kocis and I.E. Grossmann. Relaxation strategy for the structural optimization of process flow sheets. *Industrial & Engineering Chemistry Research*, 26(9):1869–1880, 1987.

[127] G.R. Kocis and I.E. Grossmann. Computational experience with DICOPT solving MINLP problems in process systems engineering. *Computers & Chemical Engineering*, 13(3):307–315, 1989.

[128] M.K. Kozlov, S.P. Tarasov, and L.G. Khachiyan. The polynomial solvability of convex quadratic programming. *USSR Computational Mathematics and Mathematical Physics*, 20(5):223–228, 1980.

[129] K. Kunisch and F. Rendl. An infeasible active set method for quadratic problems with simple bounds. *SIAM Journal on Optimization*, 14(1):35–52, 2003.

[130] S. Lacoste-Julien and M. Jaggi. An affine invariant linear convergence analysis for Frank-Wolfe algorithms. *arXiv preprint arXiv:1312.7864*, 2013.

[131] C.D. Laird, L.T. Biegler, and B. Van Bloemen Waanders. A mixed integer approach for obtaining unique solutions in source inversion of drinking water networks. *Journal of Water Resources Planning and Management, Special Issue on Drinking Water Distribution Systems Security*, 132(1):242–251, 2006.

[132] A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[133] R. Lazimy. Mixed-integer quadratic programming. *Mathematical Programming*, 22(1):332–349, 1982.

[134] R. Lazimy. Improved algorithm for mixed-integer quadratic programs and a computational study. *Mathematical Programming*, 32(1):100–113, 1985.

[135] E.K. Lee and J.E. Mitchell. Computational experience of an interior point algorithm in a parallel branch-and-cut framework. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[136] M. Lejeune. A unified approach for cycle service levels. Technical report, George Washington University, 2009.

[137] S. Leyffer. *Deterministic Methods for Mixed Integer Nonlinear Programming.* PhD thesis, University of Dundee, Dundee, Scotland, UK, 1993.

[138] S. Leyffer. User manual for MINLP-BB. Technical report, University of Dundee, 1998.

[139] S. Leyffer and A. Mahajan. *Software For Nonlinearly Constrained Optimization.* John Wiley & Sons, Inc., 2010.

[140] D. Li, X. Sun, and J. Wang. Optimal lot solution to cardinality constrained mean-variance formulation for portfolio selection. *Mathematical Finance*, 16(1):83–101, 2006.

[141] J.T. Linderoth and A. Lodi. *MILP Software.* John Wiley & Sons, Inc., 2010.

[142] J.T. Linderoth and T.K. Ralphs. Noncommercial software for mixed-integer linear programming. Technical report, Department of Industrial and Systems Engineering, Lehigh University, 2004.

[143] J.T. Linderoth and M.W.P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[144] J.D.C. Little, K.G. Murty, D.W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11(6):972–989, 1963.

[145] A. Lodi. Mixed integer programming computation. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer Berlin Heidelberg, 2010.

[146] A. Lodi, K. Allemand, and T.M. Liebling. An evolutionary heuristic for quadratic 0-1 programming. *European Journal of Operational Research*, 119(3):662–670, 999.

[147] R. Lougee-Heimer. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, 2003.

[148] A. Mahajan and T. Munson. Exploiting second-order cone structure for global optimization. Technical report, Argonne National Laboratory, 2010.

[149] A. Mahajan, C. Kirches, and S. Leyffer. Exploiting second-order cone structure for global optimization. Technical report, Argonne National Laboratory, 2012.

[150] H. Marchand and L.A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49:363–371, 2001.

[151] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[152] S. Mehrotra and J. Sun. An algorithm for convex quadratic programming that requires $O(n^{3.5}L)$ arithmetic operations. *Mathematics of Operations Research*, 15(2):342–363, 19.

[153] MIQP instances. Mean-Risk Portfolio Optimization Instances, 2015. www.mathematik.tu-dortmund.de/lsv/instances/MIQP.tar.gz.

[154] J. Moré and G. Toraldo. Algorithms for bound constrained quadratic programming problems. *Numerische Mathematik*, 55(4):377–400, 1989.

[155] J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, 1(1): 93–113, 1991.

[156] mosek. MOSEK Optimization Software, 2015. mosek.com/products/mosek.

[157] W.H. Mow. Universal lattice decoding: principle and recent advances. *Wireless Communications and Mobile Computing*, 3(5):553–569, 2003.

[158] B. Murtagh and M. Saunders. MINOS 5.5 user's guide. Technical report, Department of Operations Research, Stanford University, 1998.

[159] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, 1988.

[160] G.L. Nemhauser, M.W.P. Savelsbergh, and G.C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15(1):47–58, 1994.

[161] Y. Nesterov and A. Nemirovski. *Interior-Point Polynomial Algorithms in Convex Programming.* Society for Industrial and Applied Mathematics, 1994.

[162] J. Nocedal and S.J. Wright. *Numerical Optimization.* Springer, 2000.

[163] L. Ozsen, C.R. Coullard, and M.S. Daskin. Capacitated warehouse location model with risk pooling. *Naval Research Logistics*, 55(4):295–312, 2008.

[164] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1): 1–7, 1987.

[165] P.M. Pardalos and S. Jha. Complexity of uniqueness and local search in quadratic 0-1 programming. *Operations Research Letters*, 11(2):119–123, 1992.

[166] P.M. Pardalos and S.A. Vavasis. Quadratic programming with one negative eigenvalue is NP-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

[167] R. Pörn and T. Westerlund. A cutting plane method for minimizing pseudo-convex functions in the mixed integer case. *Computers and Chemical Engineering*, 24(12):2655–2665, 2000.

[168] M.J.D. Powell. On the quadratic programming algorithm of Goldfarb and Idnani. In *Mathematical Programming Essays in Honor of George B. Dantzig Part II*, volume 25, pages 46–61. Springer Berlin Heidelberg, 1985.

[169] I. Quesada and I.E. Grossmann. An LP/NLP based branch-and-bound algorithm for convex MINLP. *Computers and Chemical Engineering*, 16: 937–947, 1992.

[170] A. Ruszczynski. *Nonlinear Optimization.* Princeton University Press, 2006.

[171] H.S. Ryoo and N.V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, 1996.

[172] N.V. Sahinidis. *BARON 12.1.0: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2013.

[173] S. Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262–279, 1974.

[174] C. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66:181–199, 1994.

[175] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Inc., 1986.

[176] C.A. Schweiger. *Process Synthesis, Design, and Control: Optimization with Dynamic Models and Discrete Decisions.* PhD thesis, Princeton University, 1999.

[177] scip. Solving Constraint Integer Programs, 2015. http://scip.zib.de.

[178] D.X. Shaw, S. Liu, and L. Kopman. Lagrangian relaxation procedure for cardinality-constrained portfolio optimization. *Optimization Methods Software*, 23(3):411–420, 2008.

[179] M.Z. Shen, C. Coullard, and M.S. Daskin. A joint location-inventory model. *Transportation Science*, 37(1):40–55, 2003.

[180] H.D. Sherali and W.P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems.* Springer, 2010.

[181] C. De Simone. The cut polytope and the boolean quadric polytope. *Discrete Mathematics*, 79(1):71– 75, 1990.

[182] M. Tawarmalani and N.V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming : Theory, Algorithms, Software, and Applications.* Kluwer Academic Publishers, 2002.

[183] M. Tawarmalani and N.V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249, 2005.

[184] P. Van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. Technical report, University of Amsterdam, Department of Mathematics, 1981.

[185] J.P. Vielma, S. Ahmed, and G.L. Nemhauser. A lifted linear programming branch-and-bound algorithm for mixed-integer conic quadratic programs. *INFORMS Journal on Computing*, 20(3):438–450, 2008.

[186] A. Wächter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[187] T. Westerlund and K. Lundqvist. Alpha-ECP, version 5.101. an interactive MINLP-solver based on the extended cutting plane method. Technical report, Process Design Laboratory, Abo Akademi University, 2005.

[188] T. Westerlund and F. Pettersson. A cutting plane method for solving convex MINLP problems. *Computers and Chemical Engineering*, 19:131–136, 1995.

[189] T. Westerlund and R. Pörn. Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. *Optimization and Engineering*, 3(3):253–280, 2002.

[190] R.B. Wilson. *A Simplicial Algorithm for Concave Programming.* PhD thesis, Graduate School of Business Administration, Havard University, 1963.

[191] L.A. Wolsey. *Integer Programming.* Wiley-Interscience, 1998.

[192] S.J. Wright. *Primal-Dual Interior-Point Methods.* Society for Industrial and Applied Mathematics, 1997.

[193] Y. Ye and E. Tse. An extension of Karmarkar's projective algorithm for convex quadratic programming. *Mathematical Programming*, 44(1-3):157–179, 1989.