Development of a Solver for Mixed-Integer Quadratically Constrained Quadratic Optimization

Submitted in partial fulfillment of the requirements
of the degree of

Doctor of Philosophy

by

Vora Mustafa Makbul
(Roll No. 184190001)

Supervisors:
Prof. Ashutosh Mahajan

Industrial Engineering and Operations Research
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
2023

Dedicated to Hussaina.

Thesis Approval

This thesis entitled **Development of a Solver for Mixed-Integer Quadratically Constrained Quadratic Optimization** by **Vora Mustafa Makbul** is approved for the degree of **Doctor of Philosophy**.

Examiners:

.....

.....

.....

.....

Supervisor:

.....

Chairperson:

.....

Date: .....

Place: .....

Declaration

I declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date:

_____

Vora Mustafa Makbul

Roll No. 184190001

Abstract

This thesis describes the development of `mglob`, a general purpose solver for Mixed Integer Quadratically Constrained Quadratic Optimization (MIQCQO) problems. MIQCQO problems appear in several important applications. MIQCQO is a nonconvex nonlinear problem, which makes it challenging to find a globally optimal solution. Typically, spatial Branch-and-Bound algorithm is used for solving MIQCQO problems, which `mglob` also uses. Minotaur framework is used for implementing `mglob`. Minotaur framework provides several routines for reading and storing optimization problems, doing tree search, solving relaxations, etc., that are useful for solvers like `mglob`. We describe some algorithmic components we have developed for `mglob` and how well they perform on benchmark instances.

First, we describe three presolving techniques implemented in `mglob`. Presolving is a preprocessing step done in many general purpose solvers for mathematical optimization. It simplifies the problem and collects important information about the problem that can be used to solve the problem faster.

(i) The first technique converts a given quadratic function into a Dictionary of Keys format of coefficients of the quadratic function (`qf`). We study the effectiveness of these representations on function evaluation and gradient evaluation. Our results show that `qf` takes significantly less time in almost all instances except for a few instances with a special structure.

(ii) We describe an algorithm to detect convexity for separable quadratic functions that help detect convexity faster. Since MIQCQO are nonconvex problems in general, identifying convex problems or convex constraints in the problems can be beneficial to design specific algorithms for these problems.

(iii) We implement Feasibility Based Bound Tightening (FBBT), a specialized FBBT algorithm for univariate quadratic expression, and Optimality Based Bound Tightening

(OBBT) in `mglob`. Tighter variable bounds provide stronger relaxation for MIQCQO problems, which helps solve the problem faster. We study the effects of these three bound tightening techniques on the performance of the solver. Our tests show that doing OBBT at the root node can tighten the bounds of the variables significantly, and we can solve more instances than when OBBT is switched off.

Next, we describe a novel method to generate cutting planes for quadratically constrained optimization problems. The method uses information from the simplex tableau of a linear relaxation of the problem in combination with McCormick estimators. The method is guaranteed to cut off a basic feasible solution of the linear relaxation that violates the quadratic constraints in the problem as long as finite bounds on all variables are available. These cutting planes are computationally cheap and do not require any special structure in the input problem. The cuts generated by the method are the well-known Reformulation Linearization Technique (RLT) cuts. The procedure produces a large number of violated cuts. Several variants for selecting good cuts are tested. Instead of adding many cuts, one can also add auxiliary variables and a few cuts. Computational testing on benchmark test instances shows that, on average, upto 30% of the gap from the optimal can be closed.

Lastly, we describe five branching strategies for spatial branching of MIQCQO problems. Once a node in a branch-and-bound tree is solved and the solution is not feasible to the problem, we decide to branch. Typically, several variables are available for branching, and tree size greatly depends on the variable selected for branching. In the first branching strategy, we develop a distance measure that can be used to estimate the violation of a point and describe a maximum violation branching strategy. We then describe strong branching for spatial branching. Then, we develop a new branching strategy similar to strong branching, called bt-strong branching, that tightens bounds on variables using FBBT based approaches before strong branching for every candidate. This strategy gives a better estimate of lower bound update and takes fewer nodes than other branching strategies. The fourth branching strategy is bt-estimate branching. In this strategy, we first do bound tightening and then estimate lower bound update based on reduced costs for variables whose bounds get updated. This strategy is fast because no linear programs are solved. Finally, we combine these strategies into a reliability branching type setup called bt-reliability branching. We observe that bt-reliability branching performs better than other branching strategies described.

In the end, we benchmark the current state of our solver against SCIP and Gurobi. We

also compare our solver against an older version of `mglob` to see the combined effects of all the techniques described here.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Mixed-Integer Quadratically Constrained Quadratic Optimization (MIQCQO) refers to a class of mathematical optimization problems where the maximum degree of objective function and constraints is two. A general MIQCQO problem is written as

$$\begin{aligned} \min_{x} \quad & x^T Q^0 x + c_0^T x & & \\ \text{subject to} \quad & x^T Q^k x + c_k^T x \leq b_k & & k = 1, \ldots, m, \\ & \underline{x} \leq x \leq \bar{x}, & & \\ & x_i \in \mathbb{Z} & & \forall\ i \in I, \\ & x_i \in \mathbb{R} & & \forall\ i \in \{1, \ldots, n\} \setminus I, \end{aligned} \tag{Q}$$

where $Q^k = \left(q_{ij}^k\right)$ is a given $n \times n$ symmetric matrix, $c_k \in \mathbb{R}^n$, for $k = 0, \ldots, m, b_k \in \mathbb{R}$, for $k = 1, \ldots, m, \underline{x}, \bar{x} \in \mathbb{R}^n, I$ is the indicator set for the integer variables, and m, n are finite whole numbers. A vector x satisfying all the given quadratic constraints ($x^T Q^k x + c_k^T x \leq b_k$, for $k = 1, \ldots, m$), the bound constraints ($\underline{x} \leq x \leq \bar{x}$), and the integrality constraints ($x_i \in \mathbb{Z}\ \forall\ i \in I$) is said to be feasible to (Q). The problem is to find among all feasible vectors, the one that minimizes the quadratic function $x^T Q^0 x + c_0^T x$. Such a vector $\hat{x}$ is said to be the optimal solution to (Q),

and $\hat{x}^T Q^0 \hat{x} + c_0^T \hat{x}$ is the optimal value of (Q). This thesis describes some computational and algorithmic techniques to solve MIQCQO. MIQCQO is used to model decision problems in several scientific, engineering and business domains. Some of these are briefly discussed next.

In a pooling problem input material of different qualities from multiple streams (usually crude oil procured from multiple sources) are mixed in several pools. It is one example of MIQCQO from the domain of Chemical Engineering. The output from these pools are blended together to form the end products. We describe a simplified example of pooling problem taken from [88] to illustrate how MIQCQO problems are modeled.

Example 1.1. Consider three supply sources $a, b,$ and c, a single pool, and two output products $1, 2,$ as shown in Figure 1.1. Supplies from a, and b are mixed in the pool, and supply c directly feeds to the output products. The sulfur qualities in the supply at a, b, c are $3\%, 2\%, 1\%$ respectively, while the per unit costs are $\$6, \$16, \$10$ respectively. Maximum permissible sulfur qualities at $1, 2$ are $2.5\%, 1.5\%$, and the demands are $100, 200$ units respectively. The per unit profit for product 1 is \$9 and for product 2 is \$15. We want to decide the quantity of supply from the sources, the quality at the pool after mixing, and the quantity of flow from the pool to the output products. We denote the quantity from source a to the pool as x_{ap} and the quantity from source b to pool as x_{bp}. The quantity from source c to the output $1, 2$ is denoted by x_{c1}, x_{c2} respectively. Similarly, the quantity from the pool to $1, 2$ is denoted as x_{p1}, x_{p2} respectively. The pool quality is denoted as q. Using these variables, the problem is formulated below.

$$\begin{aligned}
\min \quad & 6x_{ap} + 16x_{bp} + 10(x_{c1} + x_{c2}) - 9(x_{p1} + x_{c1}) - 15(x_{p2} + x_{c2}) && \text{(1.1a)} \\
\text{s. t.} \quad & x_{p1} + x_{c1} \leq 100, && \text{(1.1b)} \\
& x_{p2} + x_{c2} \leq 200, && \text{(1.1c)} \\
& x_{ap} + x_{bp} = x_{p1} + x_{p2}, && \text{(1.1d)} \\
& 3x_{ap} + x_{bp} = qx_{p1} + qx_{p2}, && \text{(1.1e)} \\
& qx_{p1} + 2x_{c1} \leq 2.5(x_{p1} + x_{c1}), && \text{(1.1f)} \\
& qx_{p2} + 2x_{c2} \leq 1.5(x_{p2} + x_{c2}), && \text{(1.1g)} \\
& x_{ap}, x_{bp}, x_{c1}, x_{c2}, x_{p1}, x_{p2}, q \geq 0. && \text{(1.1h)}
\end{aligned}$$

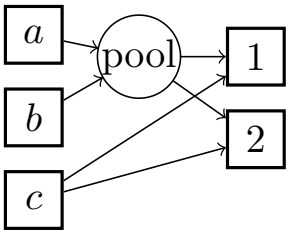


Figure 1.1: Schematic of the pooling problem described by Haverly [88]

Objective function (1.1a) minimizes the difference of the cost of raw materials and the profit from the output products. Constraints (1.1b) and (1.1c) model demand satisfaction. Constraint (1.1d) represents the mass balance for the pool and Constraint (1.1e) represents the mass balance for sulfur in the pool. Constraints (1.1f) and (1.1g) model the maximum permissible sulfur quality at the output. Finally, nonnegativity constraints (1.1h) are added. Constraints (1.1d), (1.1e), and (1.1f) are quadratic because of the presence of bilinear terms and hence this problem is a quadratically constrained problem. The optimal solution value of -400 is obtained when $q = 1, x_{bp} = x_{p2} = x_{c2} = 100, x_{ap} = x_{c1} = x_{p1} = 0$. Note that Q^0, Q^1, Q^2, Q^3 are zero, and there are no integer variables in this example.

In practice pooling problem may be much larger in scale because of multiple pools and variety of products that can be produced. This leads to many bilinear terms in the problem. Several alternative formulations and extensions have been proposed for the pooling problem, see [7, 11, 18, 26, 115, 119, 130].

Other applications in chemical engineering include crude oil scheduling [121], natural gas production [100, 101], distillation sequences [9], waste water treatment [48, 50], water network design [49, 139, 140].

Several applications of MIQCQO also appear in computational geometry problems like [20, 53, 95, 96, 99]. Another important application of MIQCQO is the trim loss problem in paper industry [64, 86, 87, 97]. MIQCQO has also been applied in supply chain management [62, 84, 89, 147, 148], optimal selection of breeding population [122], edge crossing minimization in bipartite graphs [41], graph partitioning [75], optimal power flow in electricity networks [13, 35, 51, 90], portfolio optimization in finance [30, 57, 70, 72, 143], etc.

Solution approaches to MIQCQO need to overcome two challenges: (a) nonconvexity of the quadratic functions in the objective or constraints, and (b) integer constraints on variables.

In the absence of these two, we get a convex quadratic optimization problem that is relatively much simpler to solve than MIQCQO. The algorithms for convex quadratic optimization are fundamentally different from those of MIQCQO, and are usually orders-of-magnitude faster. They are usually iterative in nature, moving from a candidate solution to the next that is 'better' in some sense, eventually converging to the optimal solution [71, 98, 113, 125]. On the other hand, search for a globally optimal solution of nonconvex MIQCQO relies on approximating the problem by solving a suitable convex approximation, and refining or subdividing it many times. A practically useful MIQCQO solver requires careful implementation and integration of several techniques for simplifying MIQCQOs, creating approximations, refining, searching etc. This thesis focuses on the development of `mglob`, an open-source solver developed within Minotaur (previously written as MINOTAUR) framework [111] for MIQCQO.

In this introductory chapter, we begin by describing the role convexity plays in solving optimization problems and highlight the difficulties when solving nonconvex optimization problems like the MIQCQO in Section 1.1. Next, we discuss two important optimization problems, namely, Linear Optimization (LO) in Section 1.2, and Mixed Integer Linear Optimization (MILO) in Section 1.3. Linear Optimization plays a key role in algorithms for MIQCQO. Algorithms and techniques used to solve an MIQCQO problem are inspired from MILO literature including Branch-and-Bound algorithm, cutting planes, presolving techniques, branching techniques, etc. Thus an overview of MILO has been presented here. In Section 1.6, we describe the spatial Branch-and-Bound algorithm which is used to solve MIQCQO problems. In Section 1.5 we describe some relaxation techniques for MIQCQO available in the literature. In Section 1.7, we briefly mention software for MIQCQO and related problems. In Section 1.8, we discuss several algorithmic components of Minotaur framework. We finally outline the remainder of the thesis and highlight our contributions in Section 1.9.

1.1 Convex vs Nonconvex Optimization

Convexity plays an important role in optimization, and many algorithms for nonconvex problems solve some convex optimization relaxations or approximations repeatedly. MIQCQO in the general form is a nonconvex problem.

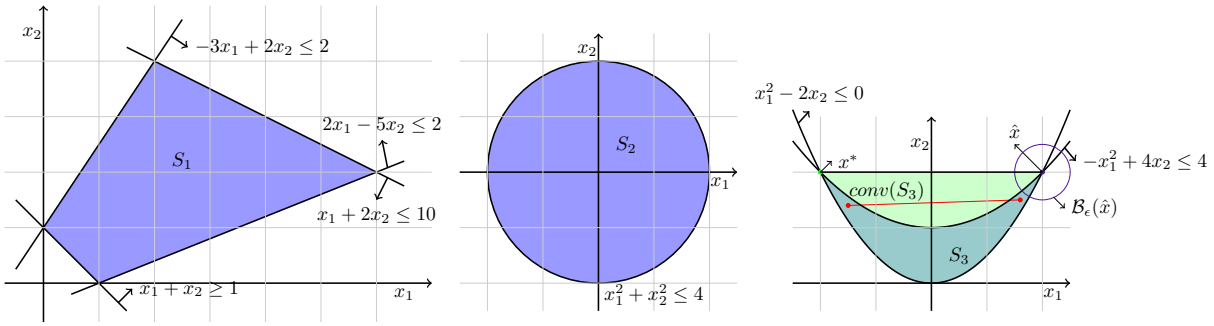


Figure 1.2: Examples of convex and nonconvex sets

Convex optimization problems are in general 'easy' to solve and several interior point algorithms exist which can provably reach the optimal solution within a small polynomial (in the size of the input) number of steps. More details about convex optimization problems can be found in [29, 40]. On the other hand nonconvex optimization problems are 'hard' problems. There are no known algorithms which can find an optimal solution within polynomial (in the size of input) number of steps. Formal definitions of computational complexity and hardness can be found in [73]

1.1.1 Convex Sets and Convex functions

A convex set C is a set such that given any two points $x, y \in C$ the line segment joining x, y is contained in C. More formally,

Definition 1.2. Convex set - A set $C \subseteq \mathbb{R}^n$ is a convex set if for any $x,\ y \in C$, and $\lambda \in [0,\ 1]$, we have $\lambda x + (1-\lambda)y \in C$.

Figure 1.2 shows some examples of convex and nonconvex sets. The *left* figure, a polyhedron, is a convex set defined by the intersection of four linear constraints, $S_1 = \{(x_1, x_2) \mid x_1 + x_2 \geq 1, -3x_1 + 2x_2 \leq 2, 2x_1 - 5x_2 \leq 2, x_1 + 2x_2 \leq 10\}$. The *middle* figure, a circle, is a convex set defined by a single quadratic constraint, $S_2 = \{(x_1, x_2) \mid x_1^2 + x_2^2 \leq 4\}$. The *right* figure, a crescent shape, is a nonconvex set defined by the intersection of two quadratic constraints, $S_3 = \{(x_1, x_2) \mid x_1^2 - 2x_2 \leq 0, -x_1^2 + 4x_2 \leq 4\}$. S_3 is a nonconvex set since the *red* line segment joining the two points marked in S_3 is not contained in S_3.

Definition 1.3. Convex hull - Given a set $S \subseteq \mathbb{R}^n$, a set $conv(S)$ is the convex hull of S if:

(i) $conv(S)$ is a convex set.

(ii) $S \subseteq conv(S)$.

(iii) If T is a convex set such that $S \subseteq T$ then $conv(S) \subseteq T$.

The convex hull of a set is a smallest convex set containing the set. For example, convex hull of set S_3 in Figure 1.2 is $conv(S_3) = \{(x_1, x_2) \mid x_1^2 - 2x_2 \leq 0, x_1 \leq 2\}$ shaded in green in Figure 1.2. Note that the convex hull of a convex set is the set itself.

A convex function is a function for which the line segment joining any two point on the graph of the function lies above the graph between those points.

Definition 1.4. Convex function - A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for any $x, y \in \mathbb{R}^n, \lambda \in [0,1]$ we have $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$.

Figure 1.3 shows two functions f, g. f is a convex function and as shown in the figure the green line segment joining the graph of the function overestimates the function. On the other hand, g is a nonconvex function since the red line segment joining the graph of the function does not overestimate at every point. We describe two important properties of differentiable convex

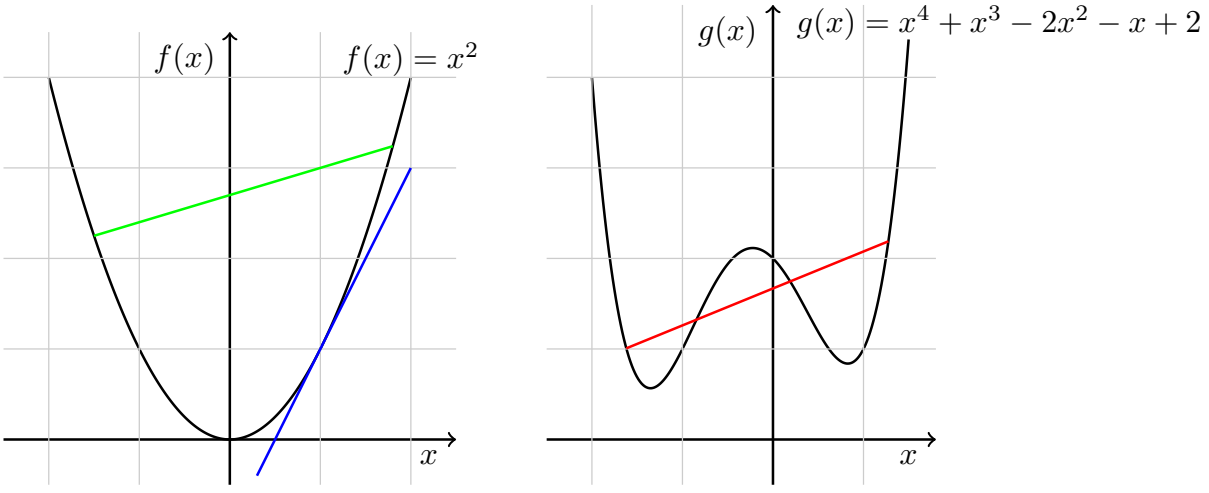


Figure 1.3: Examples of a convex (left) and a nonconvex (right) function

functions using the first and second order derivatives of the functions.

Property 1.5. First order condition [40] - If f is a convex function and f is differentiable then for any $\bar{x} \in \mathbb{R}^n$,

$$f(x) \geq f(\bar{x}) + \nabla f(\bar{x})^T (x - \bar{x}) \quad \forall\, x \in \mathbb{R}^n$$

This property shows that a tangent hyperplane to the graph of a convex function at any point is always below the graph of the function. In Figure 1.3, the blue hyperplane is a tangent to the graph of the function $f(x) = x^2$ and is always below the graph of the function.

Property 1.6. Second order condition [40] - Let f be a twice continuously differentiable function then f is a convex function if and only if its Hessian is a positive semidefinite at all points i. e., $\nabla^2 f(x) \succcurlyeq 0 \; \forall \; x \in \mathbb{R}^n$.

Definition 1.7. Convex constraint - Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function then for all $\beta \in \mathbb{R}$ the set of points satisfying the constraint $f(x) \leq \beta$ is a convex set, and the constraint is called a convex constraint.

If f is not a convex function it may still be possible that the feasible region defined by the constraint $f(x) \leq \beta$ is a convex set for some given β. For example, consider the function $f(x) = -e^x$, given any $\beta \in \mathbb{R}$ the region bounded by the constraint $f(x) \leq \beta$ is a convex set in $\mathbb{R}$ but this is still not a convex constraint since $-e^x$ is a not a convex function.

Now we define convex and nonconvex optimization problems. Suppose we have an optimization problem of the form,

$$\begin{aligned} \min_x \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq \beta_i \qquad \forall \, i \in \{1, \ldots, m\}, \\ & x \in \mathbb{R}^n, \end{aligned} \tag{1.2}$$

where the functions $f_0, \ldots, f_m$ and the constants $\beta_1, \ldots, \beta_m$ are given. If f_i, $i \in \{0, \ldots, m\}$ are all convex functions then the optimization problem is called a convex optimization problem. If any one of the f_i are not convex then it is a nonconvex optimization problem.

1.1.2 Local Minimum vs Global Minimum

Consider the optimization problem (1.2), and let the feasible region of the problem be $\mathscr{F} = \{x \in \mathbb{R}^n \mid f_i(x) \leq \beta_i \; \forall \; i \in \{1, \ldots, m\}\}$. A point x^* is a global minimum for the problem if $f_0(x^*) \leq f_0(x) \; \forall \; x \in \mathscr{F}$. On the other hand, a point $\hat{x}$ is local minimum of the problem if

$f_0(x^*) \leq f_0(x)\ \forall\ x \in \mathscr{F} \cap \mathscr{B}_\varepsilon(\hat{x})$ where $\mathscr{B}_\varepsilon(\hat{x})$ is the ball of radius ε centered at $\hat{x}$ i.e. $\mathscr{B}_\varepsilon(\hat{x}) = \{x \in \mathbb{R}^n \mid \|x-\hat{x}\| \leq \varepsilon\}$ for some $\varepsilon > 0$.

For a convex optimization problem, every local minimum is also a global minimum for the problem. For a nonconvex optimization problem, a local minimum may not be a global minimum for the problem.

Example 1.8. Consider the optimization problem $\min\{x_1 - 2x_2 \mid (x_1,x_2) \in S_3\}$ where S_3 is the nonconvex set shown in Figure 1.2. Consider the point $\hat{x} = (2,2)$ with objective value $f(\hat{x}) = -2$. There is no feasible point in the neighborhood of $\hat{x}$ for which the objective value can be decreased further i.e. $f(\hat{x}) < f(x) \forall\ x \in S_3 \cap \mathscr{B}_\varepsilon(\hat{x})$ with $\varepsilon = 0.01$. Thus $\hat{x}$ is a local minimum of the problem. The global minimum of the problem is the point $x^* = (-2,2)$ shown in green in Figure 1.2 with objective value $f(x^*) = -6$.

1.1.3 Separation

Another important distinction between convex sets and nonconvex sets comes from separating a point outside of the set. Let us consider a closed convex set C and a point $\hat{x} \notin C$, then the following theorem shows that we can construct a hyperplane that separates $\hat{x}$ from C.

Theorem 1.9. Separating Hyperplane Theorem[1] - Let C be a closed convex subset of $\mathbb{R}^n$ and let $\hat{x} \notin C$ then there exist a vector $a \in \mathbb{R}^n$ and a scalar $b \in \mathbb{R}$ such that $\forall\ x \in C, a^T x \leq b$ and $a^T\hat{x} > b$. That is, the hyperplane denoted by the constraint $a^T x = b$ separates $\hat{x}$ from C and is called a separating hyperplane.

Constructing a separating hyperplane for a convex optimization problem of the form (1.2) is straight forward. Let us consider a point $\hat{x}$ which is infeasible to the problem, then there exists an $i \in \{1,\ldots,m\}$ such $f_i(\hat{x}) > \beta_i$. Now since f_i is convex, using Property 1.5 we have for all $x, f_i(\hat{x}) + \nabla f_i(\hat{x})^T(x-\hat{x}) \leq f_i(x)$. Since we have $f_i(x) \leq \beta_i \forall\ x$ feasible to the problem, we must have

$$f_i(\hat{x}) + \nabla f_i(\hat{x})^T(x-\hat{x}) \leq \beta_i. \tag{1.3}$$

[1]This is a special case where we separate a convex set and a point, a more general version of the theorem separates two disjoint convex sets

Constraint (1.3) is not satisfied by $\hat{x}$ because $f_i(\hat{x}) > \beta_i$, and hence $f_i(\hat{x}) + \nabla f_i(\hat{x})^T (x - \hat{x}) = \beta_i$ defines a separating hyperplane. Let us illustrate it using an example.

Example 1.10. Consider the set

$$S_1 = \{x_1 x_2 \geq 4, 1 \leq x_1, x_2 \leq 5\}.$$

This set is actually a convex set but due to the presence of the nonconvex constraint $x_1 x_2 \geq 4$ it represents a feasible region of a nonconvex optimization problem. Now consider the set

$$S_2 = \{\sqrt{(x_1 - x_2)^2 + 16} \leq x_1 + x_2, 1 \leq x_1, x_2 \leq 5\}.$$

S_2 defines the same set as S_1 and all constraints in S_2 are convex. The point $\hat{x} = (1,2)^T$ is outside S_1, and S_2. For S_2, we can derive a hyperplane separating $\hat{x}$ from S_2. Property 1.5 gives the inequality $(\sqrt{17}+1)x_1 + (\sqrt{17}-1)x_2 \geq 16$ which separates $\hat{x}$ from S_2. Applying Property 1.5 to S_1 gives us the inequality $2x_1 + x_2 \geq 6$. This inequality is not valid for S_1 since the point $(1.4,3)^T$ in S_1 violates it. This shows that we cannot apply Property 1.5 when the functions are nonconvex.

1.2 Linear Optimization

An important class of convex optimization problems is that of Linear Optimization (LO) problems. Consider the optimization problem (1.2), if all $f_0, \ldots, f_m$ are linear functions then it is an LO problem. In its canonical form an LO problem is written as

$$\begin{aligned} \min \ & c^T x \\ & Ax \leq b, \\ & x \in \mathbb{R}^n, \end{aligned} \tag{1.4}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ are given and n, m are finite. Many nonconvex optimization problems including MIQCQO are solved by first creating and solving an LO problem. Since

we also use LO extensively, we describe the most relevant properties and methods briefly. A detailed treatment of LO can be found in [22, 31].

1.2.1 Structure and Properties of LO

In this section we describe the polyhedral structure of an LO problem and discuss the idea of corner point solution.

Property 1.11. [31] The feasible region $\{x \in \mathbb{R}^n \mid Ax \leq b\}$ of an LO problem is a polyhedron.

Definition 1.12. Corner Point [31] - Let $P := \{x \in \mathbb{R}^n \mid Ax \leq b\}$ be a polyhedron. A point x^* is corner point of P if there exists some $c \in \mathbb{R}^n$ such that $c^T x^* < c^T x \; \forall \; x \in P$.

Property 1.13. [31] For a polyhedron P with at least one corner point and a given $c \in \mathbb{R}^n$, if there exists a point $\hat{x}$ such that $c^T \hat{x} \leq c^T x \; \forall \; x \in P$ then there exists a corner point x^* with $c^T \hat{x} = c^T x^*$.

This property shows that for an LO problem whose optimal solution exists there must be a corner point which is also optimal.

Property 1.14. [31] If x^* is a corner point of $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ then there exists $n \times n$ invertible submatrix of A, say B, such that $Bx^* = b$.

The above property says that if x^* is a corner point then there are a set of n constraints from $Ax \leq b$ satisfied at equality. These constraints which are satisfied at equality are called active constraints. Since there can be only finitely many submatrices B, there are only finitely many corner points of a polyhedron. Now, using Property 1.13, this reduces the LO problem to that of enumeration and thus LO can always be solved in finite number of steps.

1.2.2 Simplex Method

There are broadly two categories of algorithms for solving an LO problem, namely, interior point methods and simplex method. There are several interior point methods, which in theory

are polynomial time algorithms, and can reliably solve LP problems fast. On the other hand simplex method is an exponential time algorithm in the worst case. In spite of the poor worst case complexity, simplex method is extensively used in many state-of-the-art LO solvers because of its practical speed, warm starting abilities, fast and reliable computations and a guarantee to provide a corner point solution. We now briefly describe the simplex method. A detailed description and important computational and practical aspects of the simplex method can be found in [142].

Simplex method considers the LO problem in its standard form

$$\begin{aligned} \min \ & c^T x \\ & Ax = b, \\ & x \geq 0, \end{aligned} \tag{1.5}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ are given. Note that an LO problem in canonical form can be easily converted to standard form by introducing additional nonnegtaive variables. It is assumed that A is a full rank matrix.

Definition 1.15. Basic Feasible Solution (BFS) - Consider the feasible region of the LO problem $\{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}$. We divide the matrix A into two submatrices B, N where $B \in \mathbb{R}^{m \times m}$ is invertible and, $N \in \mathbb{R}^{m \times (n-m)}$. Let x_B, x_N be the variables corresponding to the columns of B, N respectively. Then the solution $x_B = B^{-1}b$, $x_N = 0$ is known as basic solution and if $x_B = B^{-1}b \geq 0$ then it is a basic feasible solution.

Definition 1.16. Reduced Costs - Let $x = \begin{pmatrix} x_B \\ x_N \end{pmatrix}$ be a basic solution and $c = \begin{pmatrix} c_B \\ c_N \end{pmatrix}$ be the cost function then the vector $\bar{c} = c_N - c_B^T B^{-1} N$ is known as the vector of reduced costs.

Simplex method starts by selecting an initial BFS. There are several methods of obtaining an initial BFS, for example, the Phase I of the simplex method. Then it chooses an entering nonbasic variable with a negative reduced cost and a leaving basic variable such that feasibility is maintained. We then move to the next basic feasible solution with the new basis iteratively. We have reached an optimal solution when the vector of reduced costs is nonnegative.

There are several more details regarding an efficient implementation of the simplex method

for general purpose use. For example, exploiting sparsity of A, b, deciding the entering and leaving variable at every iterate (pivoting rules), handling degeneracy, getting a starting basic feasible solution etc.

After an LO problem is solved, we sometimes want to solve it again after modifying the vectors b and c, or after adding or removing some constraints. Variants of simplex method can be used to solve the modified problem starting from the last basic solution obtained previously to quickly find the solution to the new problem. This process is called warm-starting. Warm starting is a key enabler for developing LO based methods for solving other more difficult problems like Mixed-Integer Linear Optimization (MILO) and MIQCQO. We next discuss MILO.

1.3 Mixed Integer Linear Optimization

An MILO problem is an LO problem in which some or all variables are constrained to take integer values. In its general form, an MILO problem is represented as

$$\begin{aligned} \min\ & c^T x \\ & Ax \leq b, && (1.6) \\ & x_i \in \mathbb{Z} && i \in I, \\ & x_i \in \mathbb{R} && i \in \{1,\ldots,n\} \setminus I, \end{aligned}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $I \subseteq \{1,\ldots,n\}$ are given. MILO is used to model a diverse set of problems arising in practical situations like scheduling, routing, transportation etc. and important theoretical questions in graph theory, set theory, number theory, and many combinatorial problems.

MILO problem is an *NP*-hard [54] problem which means that all known algorithms to solve MILO require at least exponential number of steps in the size of the problem. Many state-of-the-art solvers for MILO use Branch-and-Cut algorithm augmented by several techniques including presolving, heuristics, conflict resolution etc.

We now discuss two important algorithms that are integrated together to solve an MILO problem because they are also extended to the case of MIQCQO. Both these algorithms start by creating a linear relaxation for the MILO problem and proceed by analyzing the optimal solution of the linear relaxation. The Branch-and-Bound algorithm is a divide and conquer type of algorithm that iteratively searches different parts of the feasible region by restricting the search space and solving LO subproblems at every iteration. On the other hand, cutting plane method solves a single linear relaxation but iteratively strengthens it to get closer to the integer optimal solution. In theory, both of these algorithms will reach optimal solution but in practice a combination of both is used.

1.3.1 Branch-and-Bound Algorithm

We now describe the Branch-and-Bound (B&B) algorithm using an example. Consider the two variable MILO problem below.

$$\begin{aligned} \min \quad & -x_1 - x_2 \\ \text{s.t.} \quad & 2x_1 - x_2 \leq 4, \\ & 4x_1 + 16x_2 \leq 71, \\ & -3x_1 + x_2 \leq 2, \\ & x_1, x_2 \geq 0, \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned} \tag{1.7}$$

We begin by solving the *natural linear relaxation* of (1.7) by relaxing the integrality constraints on both the variables. The optimal solution of the linear relaxation is $\hat{x} = (3.75, 3.5)$. Since x_1, x_2 do not take integral values at $\hat{x}$, we partition the problem in two subproblems. Noting that x_2 must be integral we create two subproblems one with $x_2 \leq 3$ and another with $x_2 \geq 4$. Thus we ensure that every feasible solution of the MILO is present in one of the subproblems. This process is known as *branching*. The objective value $\hat{z} = -7.25$ is a lower bound on the optimal objective value of the MILO problem. The upper bound on the optimal objective value currently is unknown (infinite). Once an integer feasible solution is found a finite upper bound will be

obtained as well. This is known as *bounding*. B&B algorithm iteratively solves subproblems by branching and bounds the objective value by updating lower and upper bounds. The algorithm stops when lower and upper bound are the same and we know that we have obtained an optimal solution to the problem.

The optimal solution for the left branch (node 1) with $x_2 \leq 3$ is $\hat{x} = (3.5, 3)$ and for the right branch (node 2) with $x_2 \geq 4$ is $\hat{x} = (1.75, 4)$. We further branch at node 1 on x_1 to obtain nodes 3 and 4. The optimal solution of node 3 $\hat{x} = (3, 3)$ is feasible to the MILP and we obtain an upper bound on the problem -6. This node is *pruned by feasibility*. Node 4 is infeasible and is *pruned by infeasibility*. We need not explore node 2 now since the objective value of the node is worse than the current upper bound. Thus node 2 is *pruned by bound*. The solution $x^* = (3, 3)$ is optimal to the MILO problem. The full B&B tree for the MILO problem in (1.7) is shown in Figure 1.4. In practice, a B&B tree can become large because of branching again and again.

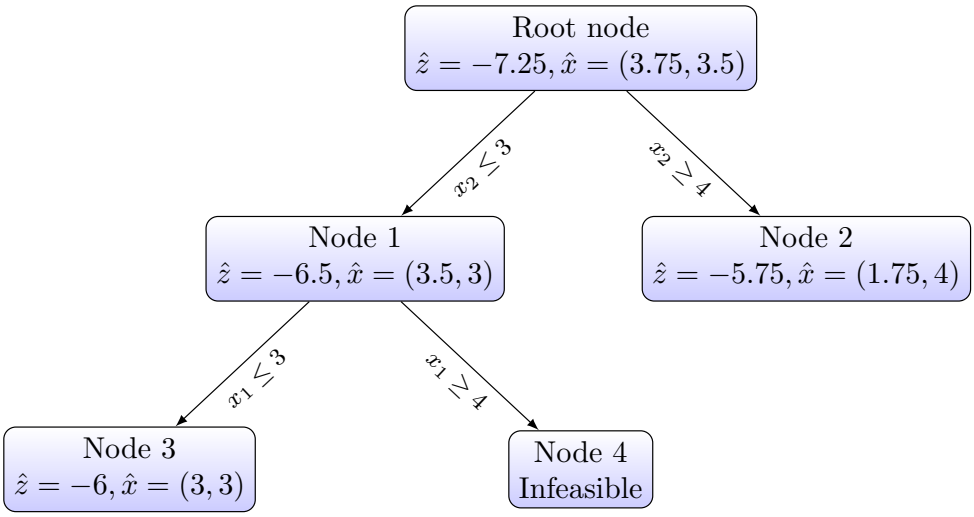


Figure 1.4: B&B tree for the MILO problem in (1.7)

1.3.2 Cutting Plane Algorithm

A cutting plane algorithm repeatedly solves a single LO problem by adding more and more constraints to the relaxation. This is in contrast with the B&B algorithm where multiple subproblems are solved. We begin by solving the natural linear relaxation of the problem. If the solution obtained is fractional then we solve a *separation* problem. This separation problem

returns a linear inequality that separates the current fractional solution from the feasible region of the MILO problem. We add this linear inequality as a constraint in the MILO problem and solve the relaxation again. This process is repeated until the optimal solution obtained is feasible to the MILO problem. The linear inequality obtained after solving the separation problem is also known as a *cut*.

There are several ways of obtaining a cut described in the literature. For a thorough discussion on different types of cuts see chapters 5, 6, and 7 of [52]. In branch-and-cut based solvers, several cuts are added at the root node to strengthen the relaxation initially and few cuts are added once branching is started. Two important algorithmic choices arise from any branch-and-cut based algorithm. Which cuts to add and what variable to branch on. Several empirical studies have considered these questions and have led to improvements in the performance of the solvers [4, 92, 141, 145].

1.4 Mixed Integer Quadratically Constrained Quadratic Optimization

We are now equipped to visit MIQCQO. Consider the problem (Q), if $Q^k = 0,\ k = 0,\ldots,m$ then (Q) reduces to (1.6). Thus MILO is a special case of MIQCQO and therefore, MIQCQO is as hard as MILO. Since MILO is *NP*-hard, we know that MIQCQO is also at least *NP*-hard. Solution methods and algorithms for solving MIQCQO problem are inspired from branch-and-bound and cutting plane algorithm described for MILO problem in the previous section. Although similar in principle there are some significant differences between algorithms for MIQCQO and MILO. First, creating a relaxation for MILO requires only relaxing the integrality while for MIQCQO specialized techniques are required for creating a relaxation. We describe some of these in Section 1.5. Second, Branch-and-bound for MILO problems terminates in finite time with exact global optimal solution while spatial branch-and-bound for MIQCQO terminates only in the limit and solution obtained in finite steps are optimal within some tolerance. We describe spatial branch-and-bound algorithm in Section 1.6. Lastly, Jeroslow [94] showed that there can not be an algorithm that provides an optimal solution to quadratic problems with unbounded integer variables. This result shows that finite bounds on

all variables appearing in quadratic terms is necessary for MIQCQO problems.

1.5 Relaxation Techniques for MIQCQO

There are several approaches to relax an MIQCQO problem. After relaxing the integrality constraints, an MIQCQO problem may still have nonconvexities from the quadratic constraints or the objective function can be nonconvex. As discussed in Section 1.1.2, we need special techniques to obtain convex relaxation of the nonconvex problem before we start the branch-and-bound process. In this section we will describe several relaxation techniques which obtain different types of convex sets as the relaxation to the MIQCQO problem.

1.5.1 McCormick Relaxation

McCormick relaxation [112] is a popular way of creating a linear relaxation of an MIQCQO problem. We first transform the problem (Q) by introducing auxiliary variables y_{ij} for each quadratic term $x_i x_j$ and adding new quadratic constraints each having only one quadratic term.

$$y_{ij} = x_i x_j \quad \forall i, j \in \{1, \ldots, n\},\ q_{ij}^k \neq 0 \text{ for some } k \in \{0, \ldots, m\}.$$

Example 1.17. Consider the following problem

$$\begin{aligned} \min \quad & x_1 x_2 - x_2 x_3 \\ \text{s. t.} \quad & x_1^2 + x_2 + x_3 \leq 1, \\ & -x_1 + 2x_2 + 2x_3 \leq 1, \\ & 0 \leq x_1, x_2, x_3 \leq 1. \end{aligned} \tag{1.8}$$

We add auxiliary variables y_{11}, y_{12}, y_{23} to obtain the following reformulation

$$\min \quad y_{12} - y_{23}$$

$$
\begin{aligned}
\text{s. t.} \quad & y_{11} + x_2 + x_3 \leq 1, \\
& -x_1 + 2x_2 + 2x_3 \leq 1, \\
& 0 \leq x_1, x_2, x_3 \leq 1, \\
& y_{11} = x_1^2, \\
& y_{12} = x_1 x_2, \\
& y_{23} = x_2 x_3.
\end{aligned} \tag{1.9}
$$

Note the Problems (1.8) are (1.9) are equivalent to each other.

We then relax these nonconvex constraints using the following inequalities

$$
\begin{aligned}
& y_{ij} \geq \underline{x}_i x_j + \underline{x}_j x_i - \underline{x}_i \underline{x}_j, \\
& y_{ij} \geq \overline{x}_i x_j + \overline{x}_j x_i - \overline{x}_i \overline{x}_j, \\
& y_{ij} \leq \overline{x}_i x_j + \underline{x}_j x_i - \overline{x}_i \underline{x}_j, \\
& y_{ij} \leq \underline{x}_i x_j + \overline{x}_j x_i - \underline{x}_i \overline{x}_j.
\end{aligned} \tag{1.10}
$$

to obtain the linear relaxation:

$$
\begin{aligned}
\min \quad & \sum_{(i,j) \in E_0} q_{ij}^0 y_{ij} + c_0^T x & & \\
\text{s.t.} \quad & \sum_{(i,j) \in E_k} q_{ij}^k y_{ij} + c_k^T x \leq b_k, & \forall \ k = 1, \ldots, m, & \\
& \left.\begin{aligned}
& y_{ij} \geq \underline{x}_i x_j + \underline{x}_j x_i - \underline{x}_i \underline{x}_j, \\
& y_{ij} \geq \overline{x}_i x_j + \overline{x}_j x_i - \overline{x}_i \overline{x}_j, \\
& y_{ij} \leq \overline{x}_i x_j + \underline{x}_j x_i - \overline{x}_i \underline{x}_j, \\
& y_{ij} \leq \underline{x}_i x_j + \overline{x}_j x_i - \underline{x}_i \overline{x}_j,
\end{aligned}\right\} & \forall \ (i,j) \in E, & \\
& \underline{x} \leq x \leq \overline{x}, & & \\
& x \in \mathbb{R}^n, & & \\
& y_{ij} \in \mathbb{R} & \forall \ (i,j) \in E. &
\end{aligned} \tag{1.11}
$$

Let E_k be the set of pairs (i, j) for which $q_{ij}^k \neq 0$, in other words the term $x_i x_j$ is present in the

corresponding quadratic function. Then $E = E_0 \cup E_1 \cup \ldots \cup E_m$ is the union of all the E_k sets, i.e. E is the set of pairs (i, j) for which the term $x_i x_j$ exists in the problem.

Example 1.18. Consider the reformulated Problem (1.9) in Example 1.17. McCormick relaxation of this problem is shown below.

$$
\begin{aligned}
\min \quad & y_{12} - y_{23} \\
\text{s. t.} \quad & y_{11} + x_2 + x_3 \leq 1, \\
& -x_1 + 2x_2 + 2x_3 \leq 1, \\
& 0 \leq x_1, x_2, x_3 \leq 1, \\
& \left.\begin{aligned} & y_{ij} \geq 0, \\ & y_{ij} \geq x_i + x_j - 1, \\ & y_{ij} \leq x_i, \\ & y_{ij} \leq x_j, \end{aligned}\right\} \quad \forall \ (i,j) \in \{(1,1),(1,2),(2,3)\}.
\end{aligned}
$$

The optimal solution to the above relaxation is $\hat{x} = (0.6, 0.4, 0.4), \hat{y} = (0.2, 0, 0.4)$. This relaxation solution is not feasible to the reformulated problem (1.9) since none of the quadratic constraints $y_{11} = x_1^2, y_{12} = x_1 x_2, y_{23} = x_2 x_3$ are satisfied. Also $\hat{x}$ is not feasible to the original problem (1.8) since the quadratic constraint $x_1^2 + x_2 + x_3 \leq 1$ is not satisfied.

1.5.2 Underestimators, Overestimators and αBB Relaxation

Another approach to relax (Q) is to replace the nonconvex functions by nonlinear convex underestimators and concave overestimators. An underestimator $\underline{f}$ of a function f has a function value less than the function value of f. That is

$$\underline{f}(x) \leq f(x) \quad \forall x \in \mathbb{R}^n.$$

If $\underline{f}$ is convex then it is a convex underestimator and if $\underline{f}$ is a maximal convex function then it is called a convex envelope.

Definition 1.19. Convex Envelope: [91] Let $\mathscr{S} \subset \mathbb{R}^n$ be a convex and compact set, and let $f : S \to \mathbb{R}$ be a lower semicontinuous function. A function $\underline{f} : \mathscr{S} \to \mathbb{R}$ is the convex envelope of f on $\mathscr{S}$ if:

(i) $\underline{f}$ is convex on $\mathscr{S}$

(ii) $\underline{f}(x) \leq f(x) \; \forall \, x \in \mathscr{S}$

(iii) there is no function $g : \mathscr{S} \to \mathbb{R}$ satisfying (i), (ii) and $g(\bar{x}) < \underline{f}(\bar{x})$ for some $\bar{x} \in \mathscr{S}$

Similarly a concave overestimator $\overline{f}$ of f is a concave function that has a function value more than that of f. Analogous to the definition of convex envelope we can define concave envelope $\overline{f}$ of f on $\mathscr{S}$ as the lowest overestimator of f on $\mathscr{S}$.

Relaxation using convex and concave envelopes of a function f gives us a convex relaxation of the epigraph of f on $\mathscr{S}$. There are few functions for which exact convex and concave envelopes are known. The following result gives us the exact convex and concave envelopes of the bilinear terms.

Theorem 1.20. [10] Let $\mathscr{S} = \{(x_1, x_2) \mid \underline{x} \leq x \leq \bar{x}\} \subset \mathbb{R}^2$ and let $f(x) = x_1 x_2$ then convex envelope $\underline{f}$ and concave envelope $\overline{f}$ of f on $\mathscr{S}$ are

$$\underline{f}(x) = \max\{\underline{x_2}x_1 + \underline{x_1}x_2 - \underline{x_1}\underline{x_2}, \overline{x_2}x_1 + \overline{x_1}x_2 - \overline{x_1}\overline{x_2}\},$$
$$\overline{f}(x) = \min\{\underline{x_2}x_1 + \overline{x_1}x_2 - \overline{x_1}\underline{x_2}, \overline{x_2}x_1 + \underline{x_1}x_2 - \underline{x_1}\overline{x_2}\}$$

The above theorem is limited to a single bilinear term ($x_1 x_2$). Applying this result to a general quadratic function by taking a sum of convex envelopes of individual terms does not give a convex envelope of the function. Meyer and Floudas [114] use some properties like edge concavity to find the convex envelopes of those functions in $\mathbb{R}^3$. Misener and Floudas [116] use these results to aggregate bilinear terms to get stronger convex relaxations of general quadratic functions.

A closely related approach to nonconvex optimization problems is the αBB underestimators [8, 15]. For a quadratic function $x^T Q^k x + c_k^T x$, a vector $\alpha^k \geq 0 \in \mathbb{R}^n$ is chosen such that $Q^k + \mathrm{Diag}(\alpha^k) \succcurlyeq 0$. Now, the function $x^T (Q^k + \mathrm{Diag}(\alpha^k))x + c_k^T x$

is a convex quadratic function (since Hessian is positive semidefinite) but does not underestimate the function since we have added $\sum_{i=1}^{n} \alpha_i^k x_i^2$. To obtain an underestimator, we subtract the secants for the univariate quadratic functions $\alpha_i^k x_i^2$. Thus, the function $x^T(Q^k + \text{Diag}(\alpha^k))x + c_k^T x - \sum_{i=1}^{n} \alpha_i^k((\underline{x}_i + \bar{x}_i)x_i - \underline{x}_i\bar{x}_i)$ is an underestimator for the quadratic function. Now we can construct a convex relaxation

$$\begin{aligned}
\min \quad & x^T(Q^0 + \text{Diag}(\alpha^0))x + c_0^T x - \sum_{i=1}^{n} \alpha_i^0((\underline{x}_i + \bar{x}_i)x_i - \underline{x}_i\bar{x}_i) \\
\text{s.t.} \quad & x^T(Q^k + \text{Diag}(\alpha^k))x + c_k^T x - \sum_{i=1}^{n} \alpha_i^k((\underline{x}_i + \bar{x}_i)x_i - \underline{x}_i\bar{x}_i) \leq b_k \quad \forall \ k = 1, \ldots, m, \quad (1.12) \\
& \underline{x} \leq x \leq \bar{x}, \\
& x \in \mathbb{R}^n \quad (1.13)
\end{aligned}$$

1.5.3 Reformulation Linearization Technique

Reformulation Linearization Technique (RLT) is a well studied method of generating a linear relaxation for (Q). This method was first described for bilinear problem by Sherali and Alameddine [134] and then extended to polynomial problems by Sherali and Tuncbilek [136]. We first divide the constraints into two sets namely, linear constraints where $Q^k = 0$, and nonlinear constraints where $Q^k \neq 0$. Although in [136] bound constraints are considered separately we include bounds on the variables in the linear constraint set for the sake of simplicity.

Now take the product of any two pair of linear constraints and substitute the quadratic terms with auxiliary variables to obtain a linear constraint as shown below.

$$\begin{aligned}
& (b_{k_1} - c_{k_1}^T x)(b_{k_2} - c_{k_2}^T x) \geq 0 \\
\implies & b_{k_1}b_{k_2} - b_{k_2}c_{k_1}^T x - b_{k_1}c_{k_2}^T x + c_{k_1}^T x x^T c_{k_2} \geq 0 \\
\implies & b_{k_1}b_{k_2} - b_{k_2}c_{k_1}^T x - b_{k_1}c_{k_2}^T x + c_{k_1}^T X c_{k_2} \geq 0
\end{aligned}$$

where $X_{ij} = x_i x_j$ are auxiliary variables used for linearization. These linearized constraints are called RLT constraints. Adding all possible RLT constraints along with original linear

constraints and bound constraints generates the first order RLT relaxation for (Q). These relaxations can be strengthened using difference of convex underestimators as described in [149].

1.5.4 Semidefinite Programming Relaxation

Let $x \in \mathbb{R}^n$ be any vector, and X be a symmetric matrix satisfying $X = xx^T$. The set $S = \{(x,X) \mid x \in \mathbb{R}^n, X = xx^T\}$ can be written using several quadratic equations of the form $X_{ij} - x_i x_j = 0$. This set is nonconvex. If we relax $X - xx^T = 0$ to $X - xx^T \geq 0$, we get a convex relaxation [137]. The constraint $X - xx^T \geq 0$ is equivalent to

$$\begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succcurlyeq 0.$$

This observation can be applied to any QCO to obtain a convex, positive semidefinite relaxation. For example, the following Semidefinite Optimization (SDP) problem is a relaxation for (Q)

$$\begin{aligned} \min \quad & \langle Q^0, X \rangle + c_0^T x & & \\ \text{s.t.} \quad & \langle Q^k, X \rangle + c_k^T x \leq b_k & \forall \quad & k = 1, \ldots, m, \\ & \underline{x} \leq x \leq \bar{x}, & & \\ & \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succcurlyeq 0. & & \end{aligned}$$

This relaxation can be strengthened in the presence of binary variables by adding constraints $X_{ii} = x_i \ \ \forall i \in I, 0 \leq x_i \leq 1$.

1.6 Spatial Branch and Bound

The relaxations obtained by the above methods can be solved relatively easily to obtain a lower bound on the optimal value of the (Q). If the relaxation is infeasible, then so is the Problem (Q).

If the optimal solution of the relaxation satisfies all the constraints of the Problem (Q), then it is also optimal to (Q). Otherwise we need to search more. One method for this search is the spatial branch and bound (sB&B) [91, 105]. The main components of the algorithm are given in Algorithm 1.1. We create a tree of nodes each involving a subproblem of the original problem (Q). A relaxation is then created for the subproblem using one of the techniques described in the previous section. sB&B is similar to B&B described for MILO in Section 1.3.1 with an added feature that branching on continuous variables is also possible. This type of branching is known as spatial branching and we describe it next.

Spatial Branching

In a nonconvex optimization problem, branching is done on both integer as well as continuous variables. Integer variable branching is similar to MILO (see Section 1.3.1). But that is not sufficient when some of the variables in quadratic constraints are continuous. Consider an MIQCQO problem involving the constraint $y = x^2, x \in [0,1]$, its standard McCormick relaxation is shown in Figure 1.5. Now let us assume that the optimal solution of the relaxed problem $(x^*, y^*) = (0.5, 0.5)$. If we branch on $x \leq 0.5, x \geq 0.5$, the relaxation after branching in either direction is shown in Figure 1.6. The infeasible point is eliminated from the relaxation in either direction of branching but all feasible solution are present in at least one of the relaxation. Although, it is clear from Figure 1.6 that spatial branching does not create disjoint relaxations after branching as is the case with integer branching, it ensures that the intersection of the feasible regions of the relaxations will always be feasible to the original problem.

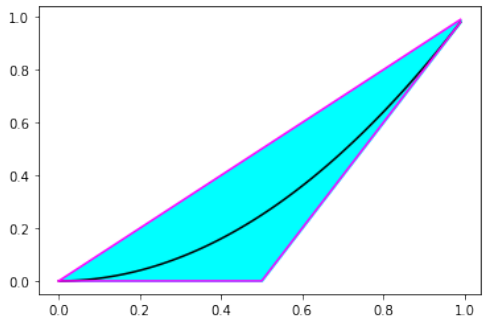


Figure 1.5: Relaxation of $y = x^2$ before branching

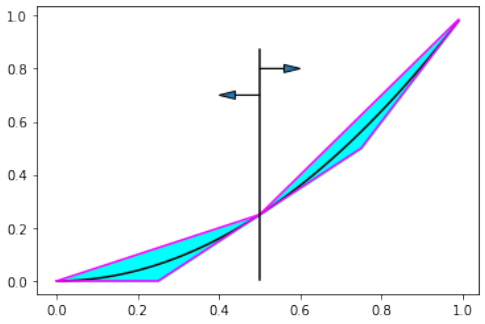


Figure 1.6: Relaxation of $y = x^2$ after branching at $x = 0.5$

For integer branch and bound algorithm, theoretically finite termination of standard branch

Algorithm 1.1 Spatial branch and bound

Input: Problem $Q, S = \{x \in \mathbb{R}^n \mid x^T Q^k x + c_k^T x \leq b_k, k = \{1,\ldots,m\}, \underline{x} \leq x \leq \bar{x}, x_i \in \mathbb{Z} \ \ \forall i \in I\}$ is the feasible region of (Q), $f(x) = x^T Q^0 x + c_0^T x$ is the objective function of (Q), $\varepsilon > 0$

Output: x^* such that $f(x^*) \leq f(x) - \varepsilon \ \ \forall x \in S$

1: **procedure** SPATIALBRANCHANDBOUND
2: Create a set of nodes $\mathscr{N} \leftarrow \{(Q, S, -\infty)\}$ where each node is a tuple of the subproblem to be solved, its feasible region, and node lower bound.
3: $z_u \leftarrow +\infty$
4: **while** $\mathscr{N} \neq \phi$ **do**
5: choose $\mathscr{N}_k := (\mathscr{P}_k, S_k, z_k) \in \mathscr{N}$ and remove $\mathscr{N}_k$ from $\mathscr{N}$
6: Create a Linear Relaxation $\mathscr{L}_k$ of $\mathscr{P}_k$
7: Let $\underline{f_k}$ be the objective function of L_k
8: Solve $\mathscr{L}_k$
9: Let $\tilde{x}$ be the optimum of $\mathscr{L}_k$
10: $z_k \leftarrow \underline{f_k}(\bar{x})$
11: **if** $\tilde{x}$ is feasible to $\mathscr{P}_k$ **then**
12: **if** $f(\tilde{x}) < z_u$ **then**
13: $z_u \leftarrow f(\tilde{x})$
14: $x^* = \tilde{x}$
15: Remove all $\mathscr{N}_j$ from $\mathscr{N}$ with node lower bound $z_j \geq z_u - \varepsilon$
16: **else if** $\mathscr{P}_k$ is infeasible **then**
17: **continue**
18: **else**
19: Choose a variable x_i for branching
20: **if** $i \in I$ **then**
21: $S_{\text{left}} \leftarrow S \cap \{x \in \mathbb{R}^n \mid x_i \leq \lfloor \tilde{x}_i \rfloor\}, S_{\text{right}} \leftarrow S \cap \{x \in \mathbb{R}^n \mid x_i \geq \lceil \tilde{x}_i \rceil\}$
22: **else**
23: $S_{\text{left}} \leftarrow S \cap \{x \in \mathbb{R}^n \mid x_i \leq \tilde{x}_i\}, S_{\text{right}} \leftarrow S \cap \{x \in \mathbb{R}^n \mid x_i \geq \tilde{x}_i\}$
24: $\mathscr{P}_{\text{left}} \leftarrow \min\{f(x) \mid x \in S_{\text{left}}\}, \mathscr{P}_{\text{right}} \leftarrow \min\{f(x) \mid x \in S_{\text{right}}\}$
25: $\mathscr{N}_{\text{left}} \leftarrow (\mathscr{P}_{\text{left}}, S_{\text{left}}, z_k), \mathscr{N}_{\text{right}} \leftarrow (\mathscr{P}_{\text{right}}, S_{\text{right}}, z_k)$
26: $\mathscr{N} \leftarrow \mathscr{N} \cup \{\mathscr{N}_{\text{left}}, \mathscr{N}_{\text{right}}\}$
27: **return** x^*

and bound algorithm is ensured. For spatial branch and bound, since continuous variables are branched as shown above, finiteness of the algorithm cannot be guaranteed. We now describe three properties of the sB&B algorithm that are useful to prove convergence of sB&B algorithm. By convergence, here we mean that

$$\min z_k \to f(x^*) \text{ as } k \to \infty$$

where z_k are the node lower bounds of the open nodes $\mathscr{N}$. That is lower bound of the problem approaches optimal value of the problem in the limit.

- Bound improving node selection [105] : Line 5 of Algorithm 1.1 is to choose a node from the set of open nodes. We do not exactly describe how this can be done. One way to select the nodes is choose that node which has the least node lower bound. This is called the best-first strategy. Any node selection strategy is said to be bound improving if the number of number of successive iterations in which node selected is different from the best first strategy is finite.

- Exactness [105] : When the box of bounds of the variables $[\underline{x}, \overline{x}]$ approaches a singleton the corresponding gap between the feasible region of the relaxation and the feasible region of the problem approaches 0. That is the relaxation becomes exact in the limit.

- Exhaustiveness [105] : sB&B algorithm equipped with the box branching as described in Algorithm 1.1 is said to be exhaustive if each infinite nested sequence generated by the algorithm converges to a singleton.

Under the assumptions of bound improving node selection, exactness, and exhaustiveness sB&B algorithm will either prove terminate in finite number of steps if the problem is infeasible, or terminate in finite number of steps and returns an optimal solution or converge to the optimal solution in the limit. For ε tolerance optimal solution, finite termination of the sB&B algorithm can be guaranteed without the requirement of the bound improvement condition.

1.7 Software for Optimization

There are many software packages available for a variety of classes of optimization problems. These software, or 'solvers', are implementations of algorithms to solve a particular class of optimization problems. We describe some of these software packages in this section.

1.7.1 LO, MILO and convex MINLO

Solvers for LO problems typically implement routines for simplex method and interior point methods. Some commonly used solvers for LO problems are CLP [67] under the COIN-OR project, HiGHS [93], Glop available in Google OR-Tools [126], SoPlex [78], etc.

MILO solvers implement branch-and-bound algorithm along with cutting planes and many other techniques to provide fast solutions on a large class of problems. Some commonly used solver for MILO include CPLEX [56], Xpress [23], Gurobi [85], CBC [66] under the COIN-OR project, HiGHS [93], SCIP [32], etc. Among these, the commercial solvers like CPLEX, Gurobi, and Xpress use their own inbuilt LO solvers. Open-source solvers, on the other hand, depend on other open-source or commercial LO solvers.

Several convex MINLO solvers use some form of branch-and-bound. The relaxations solved for the branch-and-bound may be linear or nonlinear depending on the solver. Some of these are Mosek [17], AlphaECP [146], BONMIN [39], DICOPT [83], Minotaur [111], SHOT [109], etc.

1.7.2 MIQCQO and nonconvex MINLO

Solvers for MIQCQO and nonconvex MINLO problems implement the spatial branch-and-bound algorithm along with many other techniques like cutting planes, bound tightening, etc. Some solvers for nonconvex MIQCQO problems are Alpine [124, 123], ANTIGONE [118], BARON [138], COUENNE [25], Minotaur [111], Octeract, RaPosa [81], SCIP [32] etc. Of these,

Alpine, ANTIGONE, BARON, COUENNE, Octeract, and SCIP are designed for global optimization of nonconvex MINLO. While RaPosa is designed for polynomial problems and Minotaur solves nonconvex MIQCQO only. For a detailed overview of solvers for MINLO see [44].

1.7.3 Modeling Tools

Often solvers require input problems in a specific format which may sometimes be quite difficult to create for large problems and can be difficult to read for a human. For example, solvers for nonlinear problems sometimes use `nl` file format [74] to input the problem, some solvers for polynomial problems allow `pip` format [3], solvers for MILO and LO problems require `lp` [1] and `mps` [2] format etc. To overcome this, there are several software packages which provide an interface to create a dialogue between solvers and human readable models. These packages allow the users to input the model in a much simpler way using a programming language and then they create an appropriate input for the solver. Some modeling tools commonly used to model optimization problems are AIMMS [36], AMPL [68, 69], GAMS [129], Pyomo [45], JuMP [107] etc. Pyomo and JuMP are open-source. Pyomo is based in Python programming language. A user can leverage many other features of Python (like reading from data files, plotting, sorting, etc.) and call the optimization solver. Similarly, JuMP is based in Julia programming language.

1.8 Minotaur framework for MIQCQO

Minotaur is a framework for development of solvers based on Relaxation based Branch-and-Bound algorithm. It is an open source software under the COIN-OR project. Source code for Minotaur can be accessed at `https://github.com/coin-or/minotaur` and its documentation can be found here[2]. It has several software components and data structures which makes it easier to develop algorithms that use tree search methods. For solving MIQCQO problems, we

[2]The code is written in C++ programming language. Most of the key algorithms and data structures use C++ classes. The modular code enables easy extensions and customization.

have developed mglob solver in Minotaur. Figure 1.7 shows an overview of the algorithmic framework of mglob for solving MIQCQO problems using sB&B. We now describe some of the Minotaur components.

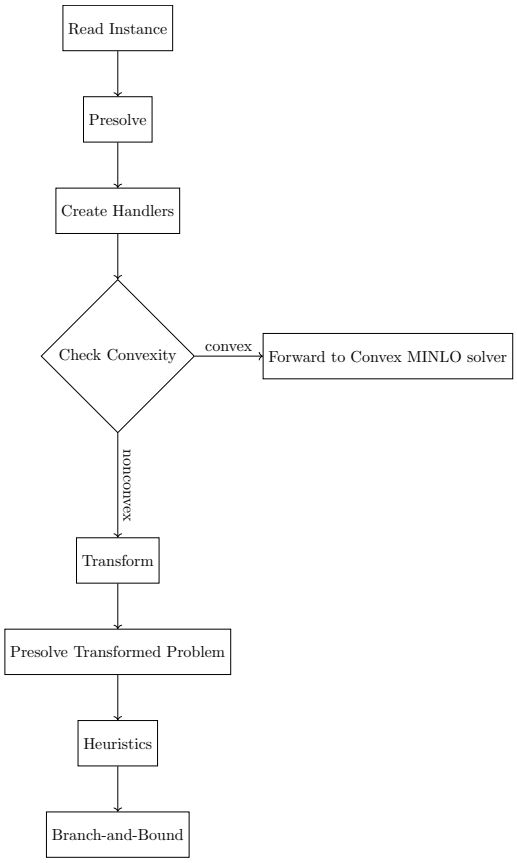


Figure 1.7: Algorithmic framework of mglob

Reader - Minotaur accepts problem instance in two formats, namely, nl, and mps. The nl files are read through a freely available external library (ASL). The reader for mps files is written within Minotaur. mps reader is limited to LO and MILO inputs only. Reader parses the input problem file and creates an instance of Problem class in Minotaur.

Presolver - Presolver identifies reductions and simplifications for the problem such that it reduces the overall computational effort to solve the problem. Many techniques are available in Minotaur for presolving like bound tightening, checking for redundant constraints, scaling and coefficient improvement, dual fixing etc. For sB&B algorithm in Minotaur, presolving is done twice before processing the root node. Once right after we create the Problem instance and once after we transform the problem by adding auxiliary variables. Presolving is also done at every node of the B&B tree.

Handler - Nonlinear problems can have several constraint types having specific structure which can be exploited individually. Minotaur offers a way to handle different constraints separately by creating a `Handler` for the constraint type. Currently, there are several `Handlers` in Minotaur like `LinearHandler` for handling linear constraints, `IntVarHandler` for handling integer variables, `QuadHandler` for handling quadratic constraints etc. Each `Handler` has a set of constraints and is responsible for creating a relaxation for its constraints, presolving the problem based on reductions implied by its constraints, check feasibility of its constraints, providing branching candidates for its constraints, and generating cuts based on its constraints.

Transformer - A `Transformer` in Minotaur is used to reformulate the problem by adding auxiliary variables and substituting nonlinear functions with those auxiliary variables. For every instance, Minotaur first reformulates the problem such that every constraint in the transformed problem can be handled by one of its `Handlers`. For example, given an MIQCQO instance Minotaur will replace all bilinear or square terms in the problem with auxiliary variables and adds additional nonlinear constraints that will be handled by `QuadHandler`.

Heuristics - Minotaur has a library of primal heuristics like feasibility pump, diving, multi-start, etc. that are used to get good upper bounds at the root node. Currently, our sB&B implementation uses only multi-start heuristic for problems with no integer variables.

Brancher - A `Brancher` is an implementation of a branching strategy that selects a branching candidate given a set of branching candidates. Every `Handler` provides a set of branching candidates and the `Brancher` then scores each candidate using some algorithm and returns a candidate variable to branch on. There are several `Branchers` in Minotaur, like, `MaxVioBrancher`, `MaxFreqBrancher`, `StrongBrancher`, `ReliabilityBrancher`, etc.

There are many other components in Minotaur like, node processors which processes the nodes, a tree manager for managing the B&B tree, several interfaces to third-party solvers that solves the relaxations etc. A detailed overview of Minotaur and its components can be found in [111].

1.9 Contributions and Outline of the Thesis

The main contribution of this thesis is the development of a general purpose solver `mglob` for MIQCQO problems in Minotaur framework. To this end we have done many improvements in existing code as well as made new developments. Thus one important outcome of this thesis is that `mglob` is now able to solve MIQCQO problems more reliably and faster. To describe the developments we have done in Minotaur we divide this thesis into four chapters.

Chapter 2 describes a presolver for MIQCQO problems in `mglob`. There are many presolving techniques in the literature, several of which were already implemented in Minotaur. We focus mainly on three techniques that we have implemented for MIQCQO specifically. We make the following contributions in Chapter 2.

(i) We study the impact of using two different representations of quadratic functions in our solver, namely, representation using computational graphs and dictionary of key format of a sparse matrix representation. We test the effectiveness of these two representations on function evaluation and gradient evaluation for quadratic functions. Our study shows that using a sparse matrix representation as we have described leads to much faster evaluations in most instances that we have tested.

(ii) We then describe a simple algorithm to decompose a quadratic function into different parts such that each part has mutually exclusive set of variables. This allows us to check the convexity of quadratic function much faster. This decomposition also allows us to gather information regarding separability which can be further used to develop better relaxations for the problem.

(iii) We also implement three different bound tightening techniques in `mglob` and study their effectiveness on reducing the range of variables.

In Chapter 3, we develop a novel general purpose cut generating algorithm for quadratically constrained optimization (QCO) problems. Cuts or cutting planes are additional constraints that are derived to tighten the relaxation for a nonconvex problems. Tighter relaxations provide better lower bounds for the problem. We analyze the simplex tableau of the

linear relaxation of a QCO and use McCormick estimators to generate cuts. Following are the key contributions in Chapter 3

(i) We describe a novel cutting plane algorithm for quadratically constrained optimization problems that is guaranteed to cut an infeasible solution of a linear relaxation for the problem.

(ii) Our procedure is computationally cheap and does not require any matrix factorization or decomposition, or solving cut generating LP etc. Thus we give a fast algorithm to separate an infeasible LP point from the problem.

(iii) Our algorithm has several choices and we devise six different variants of our algorithm on which we test the effectiveness of the cuts on benchmark instances.

(iv) We also describe a method to add additional variables to generate an RLT type relaxation based on our algorithm. The relaxation we describe is equivalent to adding all possible cuts from our algorithm.

In Chapter 4, we describe five branching strategies that we have implemented for spatial branching of MIQCQO problems. We make the following contributions in Chapter 4.

(i) We describe a branching scheme analogous to the maximum infeasible branching for MILO problems. We describe a distance measure that can be used to score the violation of a point that is infeasible to nonconvex nonlinear constraints.

(ii) We describe an extension to strong branching called bt-strong branching which does bound tightening before strong branching calls for every candidate.

(iii) We describe another branching scheme called bt-estimate branching where we do bound tightening for every candidate and then use reduced costs to estimate the lower bound improvement for each candidate.

(iv) We then combine these branching strategies to develop a reliability branching setup that can be used for spatial branching for MIQCQO problems. We call this branching scheme as bt-reliability branching.

In Chapter 5, we finally conclude this thesis. We benchmark the performance of `mglob` against SCIP, Gurobi and also an older version of `mglob` to see the effects of the techniques described. We also discuss some future research directions for development of `mglob`.

This page was intentionally left blank.

Chapter 2

Presolving Techniques

A general purpose solver for optimization problems usually begins by preprocessing the input problem. This preprocessing step is commonly known as presolving. It constitutes a wide range of techniques to transform the problem or collect important information about it. Transformation of the problem may include identifying and removing redundant constraints, substituting variables, tightening variable bounds, scaling, coefficient reduction, and more advanced techniques like reduced cost fixing. Presolving also allows for collecting important information regarding specific constraints or a group of constraints. It can help identify specific structures (like knapsack constraints, network flow structure, etc.) in the problem for which efficient solving techniques can be employed. For nonlinear problems, convexity detection and information regarding variables participating in nonlinear constraints also help devise special solution methods for specific problem types.

Achterberg and Wunderling [5] have done extensive computational analysis of various presolving components. Bixby and Rothberg [37] show that if root node presolve is disabled in CPLEX 8.0, then the performance of the solver degrades by a factor of 10.8, while if node presolve is disabled, performance degrades by a factor of 1.3. A detailed survey about presolving techniques for MILO problems can be found in [110]. Puranik and Sahinidis [128]

survey presolving techniques for NLO and MINLO problems.

This chapter describes three presolving techniques implemented in `mglob` for MIQCQO. In Section 2.1, we describe how representation of a quadratic function efficiently helps reduce function and gradient evaluation times while enabling easy access to the terms of the quadratic function. Section 2.2 shows the importance of convexity detection and describes an algorithm for fast convexity detection for a quadratic function. Section 2.3 describes several bound tightening techniques implemented in `mglob` for MIQCQO. In Section 2.4, we finally describe how all the presolving techniques described here are integrated with other techniques already present in Minotaur to get a functional presolver for MIQCQO problems, along with some concluding remarks. The presolving techniques implemented in `mglob` as part of this chapter makes the solver robust by significantly reducing the number of failing instances. It also helps in getting correct solutions for more instances than previously obtained. This creates a base solver that is used to implement and test ideas presented in the subsequent chapters.

2.1 Representation of a Quadratic function

To solve an optimization problem efficiently, appropriate data structures must be used to represent the problem such that fast and reliable computations are possible. Most algorithms for solving a nonlinear problem, including the branch-and-bound algorithm, require fast function, gradient, and Hessian evaluations of the nonlinear function. Typically, nonlinear functions are stored as a computational graph, and we use automatic differentiation to evaluate gradient and Hessian. Since quadratic functions are a special kind of nonlinear function, we can also use sparse matrix representation to store quadratic functions separately.

In this section, we describe both representations and how function and gradient evaluation are done for each. Since quadratic functions have constant Hessian and can be easily stored, we do not compare the efficiency of computing Hessian with either representation.

2.1.1 Computational Graphs

A Computational Graph (CG) is used to store an instance of a nonlinear problem in solvers for MINLO [144], nonlinear functions for automatic differentiation [82] in scientific applications, and for interval analysis of nonlinear functions [132], etc. A CG is a *directed acyclic graph* (DAG) where each node represents either an operator, a constant, or a variable of the problem. Edges entering an operator node are the inputs of the operator, and edges exiting a node carry the output value of the operation defined in the node. Consider the quadratic problem, (nvs03), from MINLPLib [43].

$$\begin{aligned} \min_{x\in\mathbb{Z}^2} \quad & (x_1-8)^2+(x_2-2)^2 \\ \text{s.t.} \quad & -0.1x_1^2+x_2 \geq 0, \\ & -0.33x_1-x_2 \geq -4.5, \\ & 100 \leq x_1,x_2 \leq 200. \end{aligned} \tag{nvs03}$$

The computational graph for (nvs03) is shown in Figure 2.1. The green source nodes x_1, x_2 are the variables of the problem. The blue sink node represents the objective function to be minimized. The yellow sink nodes are the constraint functions for which lower and upper bounds are provided according to the constraints. For a non-commutative operation like subtraction, the left and right nodes are defined appropriately to identify the order of operations correctly. A computational graph can thus store a nonlinear function composed of operators from a fixed set. They can not represent more general nonlinear functions like integro-differential equations. We next discuss how function evaluation and gradient evaluation are done using CG.

Function Evaluation

Given an $\hat{x}$ and a CG for a nonlinear function $f(.)$, function evaluation $f(\hat{x})$ is done using forward propagation of values from the variables in the graph. We propagate the values of the variables starting from the source node till the sink node corresponding to the nonlinear function. Each intermediate node computes its output from the values of the parent nodes. Figure 2.2 shows the computation at each node while evaluating the objective function value of

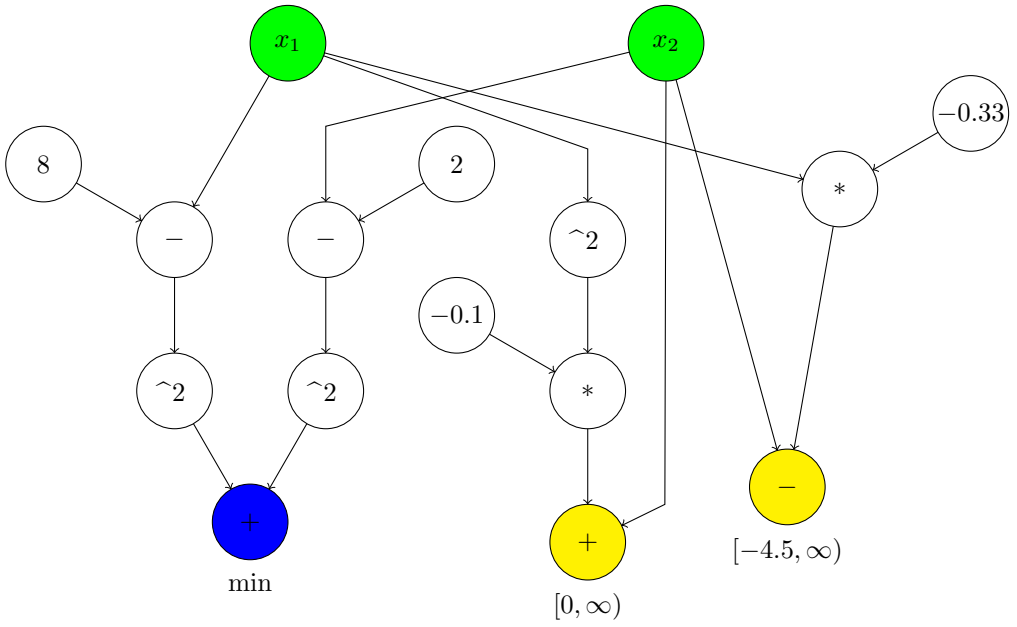


Figure 2.1: Computational graph of (nvs03)

(nvs03) at $\hat{x} = \begin{pmatrix} 6 & 3 \end{pmatrix}^T$. The values evaluated at each node is shown below the corresponding node and the function value, 5, is shown below the output node.

Gradient Evaluation:

Gradient evaluation is done using automatic differentiation techniques. We begin by evaluating the value of each node at a given $\hat{x}$. This step is the same as the function evaluation described above. For clarity, we have named each node N_i in Figure 2.2. Each pass of the CG evaluates the partial derivative with respect to a single variable using the chain rule of derivatives. Given a variable, say x_1, for which partial derivative needs to be computed, we begin by initializing the partial derivatives with respect to each variable at the variable nodes. The partial derivative with respect to the x_1 node is initialized with 1 and 0 for all other variable nodes. For all other nodes, the partial derivative with respect to x_1 is computed using the partial derivatives of the parent nodes and the chain rule for differentiation. Finally, we obtain the partial derivative of the function with respect to x_1 at the sink node. For illustration, the steps taken to compute the partial derivative with respect to x_1 for the objective function of (nvs03) is tabulated in Table 2.1. This method is called the forward mode in automatic differentiation. A somewhat less intuitive but more efficient way is to compute the partial derivatives in the reverse mode [82]. Minotaur uses the reverse mode.

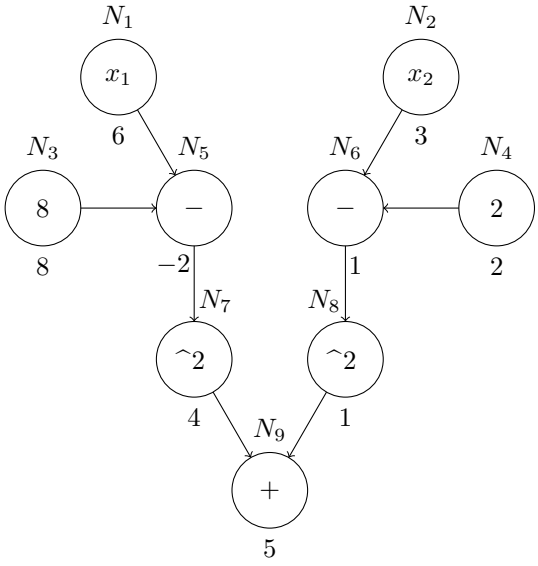


Figure 2.2: Objective function evaluation at $\hat{x} = (6, 3)$ for the CG of (nvs03)

Node	Function value	Partial derivative expression	Derivative evaluation
N_1	6	$\frac{\partial N_1}{\partial x_1}$	1
N_2	3	$\frac{\partial N_2}{\partial x_1}$	0
N_3	8	$\frac{\partial N_3}{\partial x_1}$	0
N_4	2	$\frac{\partial N_4}{\partial x_1}$	0
N_5	-2	$\frac{\partial N_5}{\partial x_1} = \frac{\partial N_5}{\partial N_1}\frac{\partial N_1}{\partial x_1} + \frac{\partial N_5}{\partial N_3}\frac{\partial N_3}{\partial x_1}$	$1 \times 1 + (-1) \times 0 = 1$
N_6	1	$\frac{\partial N_6}{\partial x_1} = \frac{\partial N_6}{\partial N_2}\frac{\partial N_2}{\partial x_1} + \frac{\partial N_6}{\partial N_4}\frac{\partial N_4}{\partial x_1}$	$1 \times 0 + (-1) \times 0 = 0$
N_7	4	$\frac{\partial N_7}{\partial x_1} = \frac{\partial N_7}{\partial N_5}\frac{\partial N_5}{\partial x_1}$	$2N_5 \times 1 = -4$
N_8	1	$\frac{\partial N_8}{\partial x_1} = \frac{\partial N_8}{\partial N_6}\frac{\partial N_6}{\partial x_1}$	$2N_6 \times 0 = 0$
N_9	5	$\frac{\partial N_9}{\partial x_1} = \frac{\partial N_9}{\partial N_7}\frac{\partial N_7}{\partial x_1} + \frac{\partial N_9}{\partial N_8}\frac{\partial N_8}{\partial x_1}$	$1 \times (-4) + (-1) \times 0 = -4$

Table 2.1: Partial derivative evaluation of the objective function of (nvs03) with respect to x_1

2.1.2 Dictionary of keys

Although CGs are powerful objects for representing a nonlinear function in general, quadratic functions have a particular structure in the sense that every term has a maximum degree of two and hence can be stored more efficiently using other methods. A quadratic function can be represented mathematically as $x^T Qx + a^T x$, given Q and a. Thus, one may use a sparse matrix for Q and a sparse vector for a to store a quadratic function.

There are various ways of storing sparse matrices. For example, the Compressed Sparse Row format stores three arrays, one each for nonzero entries, column indices, and row starts. It is used when row operations are done on the matrix frequently. We use the dictionary of keys format for storing the matrix Q and the vector a. We store the Q matrix as a dictionary where the key is the pair of variables, and the value is the coefficient of the term. The vector a is stored as a dictionary whose key is the variable, and the value is the coefficient of that variable. Since our primary use for storing the quadratic function is for reformulation and its relaxation rather than doing advanced matrix operations, a dictionary of keys format works very well for our purpose. It also allows us to easily access the pairs of variables for which a product exists and easily store all the nonlinearly participating variables. These additional features become helpful while doing presolving operations like Optimality Based Bound Tightening, described in Section 2.3.5. For illustration, we again consider the quadratic problem (nvs03) and represent it in a dictionary of keys format.

$$\begin{aligned} 68 + \min_{x \in \mathbb{Z}^2} \quad & \{< x_1, x_1 >: 1, < x_2, x_2 >: 1\}, \{x_1 : -16, x_2 : -4\} \\ \text{s.t.} \quad & \{< x_1, x_1 >: -0.1\}, \{x_2 : 1\} \geq 0, \\ & \{x_1 : -0.33, x_2 : -1\} \geq -4.5, \\ & 100 \leq x_1, x_2 \leq 200. \end{aligned}$$

Here $< \cdot, \cdot >$ represents a pair of variables and $\{\cdot : \cdot, \cdot : \cdot, \ldots\}$ represents a dictionary of (`key` : `value`) pair. For notational convenience, we will denote the representation of the Q matrix as the `qf` part of the quadratic function and the representation of the a vector as the `lf` part of the quadratic function.

Function Evaluation

Function evaluation for this format of representation is pretty straightforward. We start by initializing $s = 0$. Given an $\hat{x}$, for each term, $q_{ij}x_ix_j$, in the `qf` we increment s by $q_{ij}\hat{x_i}\hat{x_j}$. Then, for each term, l_ix_i, in `lf`, we increment s by $l_i\hat{x_i}$. s is then returned as the value of the quadratic function at $\hat{x}$.

Gradient Evaluation

For gradient evaluation, we start by initializing $g = 0 \in \mathbb{R}^n$. Given an $\hat{x}$, for each term, $q_{ij}x_ix_j$, in the `qf` we increment g_i by $q_{ij}\hat{x_j}$ and g_j by $q_{ij}\hat{x_i}$. Then, for each term, l_ix_i, in `lf`, we increment g_i by l_i. g is then returned as the gradient of the quadratic function.

2.1.3 Computational Results

In this section, we compare our implementation of CG and `qf` in Minotaur with respect to function and gradient evaluation. We used the MINLPLib [43] dataset and selected all instances with quadratic objective or constraints. There are 830 such instances. For each instance, we sample 1000 points uniformly from the box $[\underline{x}, \overline{x}]$. If $\underline{x}_i$ is not given for a variable, then we choose $\underline{x}_i = -1000$ for that variable. Similarly, if $\overline{x}_i$ is not given, then we choose $\overline{x}_i = 1000$. For all 1000 points sampled, we evaluate the function and gradient value using CG and `qf` for all quadratic functions in the problem. Finally, we report the total time taken for function and gradient evaluation for CG and `qf`. We keep a time limit of 120 seconds for this evaluation to complete. We removed those instances for which the total time for function and gradient evaluation for the 1000 sampled points by CG and `qf` is more than 120 seconds. This leaves us with 614 instances for which we present the results.

Table 2.2 shows the average time taken for function and gradient evaluation by CG and `qf` for the 614 instances. We see that `qf` is more than six times better in function evaluation than CG and more than eight times better in gradient evaluation. There are only 14 instances for which the function evaluation time for CG is better than that of `qf`, and we observe that in all those instances, either the quadratic functions present are very dense or there is a square of a linear function or a product of two linear functions. In the first case, representing a quadratic function using a sparse matrix has no additional benefits. Additionally, for such cases, CG

	CG	qf
Average time in function evaluation (ms)	1019.15	161.50
Average time in gradient evaluation (ms)	1644.83	192.89

Table 2.2: Average time for function and gradient evaluation by CG and qf

performs marginally better because, for a square term, CG computes the square of the given input, but qf will multiply the input with itself, which is slower. In the second case, the number of operations required for CG are just $k-1$ additions and 1 square operation, while for qf, the number of operations are $\mathcal{O}(k^2)$, where k is the number of linear terms. Thus, CG can perform better in this case. A similar analysis for the third case shows that CG will do $\mathcal{O}(k_1+k_2)$ operations while qf will do $\mathcal{O}(k_1k_2)$ operations, where k_1, k_2 are the number of linear terms in the product. For gradient evaluation, there are only four instances where CG performs better than qf, three of which are small instances with only 4 or 5 variables and only square terms present in the problem, and one instance has a product of linear functions as input. Again, in these situations, CG performs better than qf.

2.2 Convexity Detection

Convexity plays an important role in solving optimization problems, as discussed in Section 1.1. For nonconvex problems, identifying convex constraints can lead to tighter relaxations, and we can also generate stronger separation algorithms. In this section, we describe a decomposition based algorithm that exploits the graph structure of a quadratic function to detect convexity.

2.2.1 qf as a graph

A qf can be represented as a graph where nodes represent the variables in the qf and edges are connected if the product of the two variables in the qf exists. A square term in the qf is

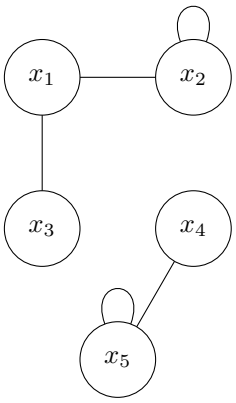


Figure 2.3: Representing `qf` (2.1) as a graph

represented as a self loop on the node of the variable. For example, consider the `qf`

$$x_1x_2 - 3x_1x_3 + 2x_2^2 - x_4x_5 + x_5^2 \tag{2.1}$$

This can be represented as a graph as shown in Figure 2.3. The coefficients of the term can also be represented by adding edge weights, but we have avoided it in this example for simplicity.

2.2.2 Convexity detection using subgraphs of the `qf`

A function is convex if its Hessian is positive semidefinite over all points in its domain, as seen in Property 1.6. Since quadratic functions have constant Hessian, we can check the convexity of a quadratic function by checking the positive definiteness of its Hessian matrix. Computational complexity of checking positive semidefiniteness of a matrix is $\mathcal{O}(n^3)$ either using Cholesky decomposition or eigenvalue decomposition. Thus, checking positive semidefiniteness of a set of smaller matrices is desirable rather than checking for a large matrix. This section describes a simple algorithm to decompose a `qf` into smaller parts to detect convexity quickly. This algorithm also allows us to separate variables of a `qf` to exploit separability to obtain stronger relaxation or better reformulations.

Our objective is to decompose the `qf` so that each separate part has a mutually exclusive set of variables from other parts. This decomposition is the same as finding disconnected subgraphs of the `qf`. We say that a subgraph S of graph G is a disconnected subgraph if, for a

Algorithm 2.1 An algorithm to find subgraphs of a $\mathtt{qf}$

Input: A quadratic function $\mathtt{qf}$

Output: A set S of quadratic functions which are subgraphs of $\mathtt{qf}$

procedure FINDSUBGRAPHS
    $S \leftarrow \phi$
    $I \leftarrow 0 \in \mathbb{R}^n$
    $k \leftarrow 0$
    for each term $q_{ij}x_ix_j$ in $\mathtt{qf}$ **do**
        if $I_i = 0$ & $I_j = 0$ **then**
            $k \leftarrow k+1$
            $\mathtt{qf}_k \leftarrow q_{ij}x_ix_j$
            $I_i \leftarrow k$
            $I_j \leftarrow k$
            $S \leftarrow S \cup \{\mathtt{qf}_k\}$
        else if $I_i = 0$ **then**
            $\mathtt{qf}_{I_j} \leftarrow \mathtt{qf}_{I_j} + q_{ij}x_ix_j$
        else if $I_j = 0$ **then**
            $\mathtt{qf}_{I_i} \leftarrow \mathtt{qf}_{I_i} + q_{ij}x_ix_j$
        else
            if $I_i = I_j$ **then**
                $\mathtt{qf}_{I_i} \leftarrow \mathtt{qf}_{I_i} + q_{ij}x_ix_j$
            else
                $k \leftarrow k+1$
                $\mathtt{qf}_k \leftarrow \mathtt{qf}_{I_i} + \mathtt{qf}_{I_j} + q_{ij}x_ix_j$
                for $t \in \{1,\ldots,n\}$ **do**
                    if $I_t = i$ or $I_i = j$ **then**
                        $I_t \leftarrow k$
                $S \leftarrow S \cup \{\mathtt{qf}_k\}$
                $S \leftarrow S \setminus \{\mathtt{qf}_{I_i}, \mathtt{qf}_{I_j}\}$
    return S

node v_1 in S and a node v_2 not in S there does not exist an edge connecting v_1, v_2 and additionally any two nodes in S are connected via a path. We use Algorithm 2.1 to find subgraphs of a `qf`. Given a `qf`, Algorithm 2.1 returns a set S of $\texttt{qf}_k$, for $k = 1, \ldots, |S|$ such that $\texttt{qf} = \sum_{k=1}^{|S|} \texttt{qf}_k$ and every variable in `qf` appears in exactly one $\texttt{qf}_k$.

Algorithm 2.1 describes an approach to find block diagonal structure of a matrix when it is stored as a dictionary of keys format (as a `qf`). Most implementations of finding block diagonal decomposition use the sparse matrix format. We use the `qf` representation for many purposes other than convexity detection. For example, it allows us to easily identify pairs of variables that has a product among them, create relaxation for such products easily, identify and store branching variables faster etc. Converting `qf` to sparse matrix and back to `qf` has extra computational overheads which is avoided by implementing Algorithm 2.1. Thus, we believe that our implementation will perform well within the context of the solver.

2.3 Bound Tightening

Stronger relaxations can be obtained if the variable bounds are tightened. For nonconvex problems where relaxations are created based on the bounds of the variables (for example, McCormick relaxation described in Section 1.5.1), tightening variable bounds allows us to create much tighter relaxation, and there is a nontrivial reduction in the size of the relaxation. For instance, consider the constraint $y = x^2, x \in [\underline{x}, \bar{x}]$, region showed in gray in Figure 2.4 shows the McCormick relaxation of the constraint. On the other hand, if we have tighter bounds for $x \in [\underline{x}', \bar{x}']$ then the feasible region of the relaxation gets tightened as shaded in blue in Figure 2.4. This shows that bound tightening reduces the search space beyond the trivial reduction. Tighter relaxations provide better lower bounds, reducing the number of nodes visited in the B&B tree.

Many bound tightening techniques have been studied in the literature and practically implemented in state-of-the-art solvers for global optimization. Two important bound tightening techniques studied extensively in the literature are *Feasibility Based Bound Tightening (FBBT)* and *Optimality Based Bound Tightening (OBBT)*. FBBT uses simple interval arithmetic to forward propagate variable bounds to the constraints and then backward

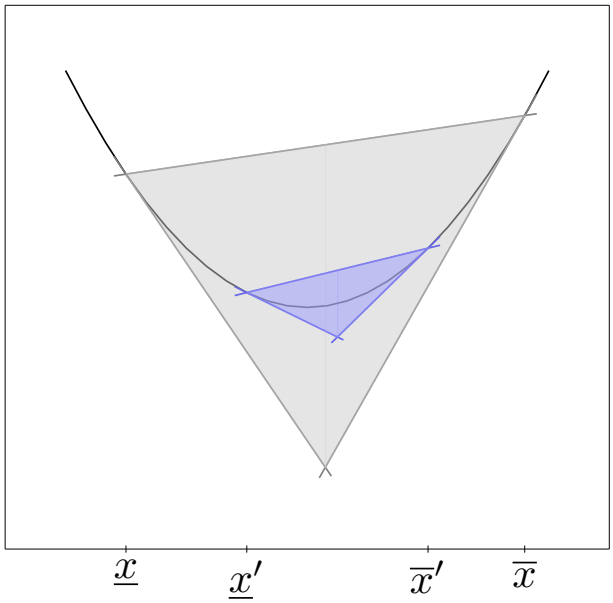


Figure 2.4: Relaxation of $y = x^2, x \in [\underline{x}, \overline{x}]$ shaded in gray and relaxation of $y = x^2, x \in [\underline{x}', \overline{x}']$ shaded in blue

propagate the constraint bounds to the variables using inverse interval arithmetic operations. On the other hand, OBBT uses a tractable relaxation of the original problem and then solves several optimization problems by maximizing and minimizing a variable over the feasible region of the relaxation.

In this section, we present three techniques for bound tightening implemented in `mglob`. All the techniques described in this section are taken from [60, 120, 128, 132]. We then present computational results to show the effectiveness of these techniques on benchmark instances.

2.3.1 Literature Review

FBBT is performed on a DAG of the constraints and uses interval arithmetic techniques to obtain bounds through the graph [120, 132]. Puranik and Sahinidis [128] survey several domain reduction techniques and describe several extensions to FBBT. Belotti et al. [24] describe a large Linear Program (LP) that converges to the fixed point of FBBT operations. They show that instead of solving multiple passes of FBBT, one can solve the LP and get tighter bounds for all the variables. Carrizosa et al. [47] describe how translating variables can improve bounds in univariate and multivariate polynomials. Domes and Neumaier [61] give rigorous methods for

bound tightening of Constraint Satisfaction Problems (CSP) using linear relaxations. They also give an algorithm for constraint propagation for quadratic constraint using univariate quadratic expression and removing bilinear terms from the constraints [60].

OBBT, although computationally expensive, is highly effective on some problems like optimal power flow problem [51]. Bynum et al. [46] describe a method to solve a series of small OBBT problems. They partition a bipartite graph of variables and constraints to obtain smaller OBBT problems. Gleixner et al. [77] describe three techniques to make OBBT more efficient, namely, aggressively filtering those OBBT problems which necessarily does not lead to tightening, ordering variables for OBBT problems so that less number of simplex iterations are done in each solve, and obtaining some additional redundant cuts which can be used to obtain better bounds.

2.3.2 Feasibility Based Bound Tightening

Feasibility Based Bound Tightening (FBBT) is a simple method to infer bounds for constraints and variables. Each iteration of FBBT has two steps. Forward propagation uses variable bounds and employs interval arithmetic operations to infer bounds on the constraints. Backward propagation uses bounds on the constraints and employs reverse interval arithmetic operations to infer tighter bounds on the variables. For an MIQCQO, the following are some interval arithmetic operations frequently used in FBBT. For any $x \in [\underline{x}, \bar{x}]$ where $\underline{x} \in \mathbb{R} \cup \{-\infty\}, \bar{x} \in \mathbb{R} \cup \{\infty\}$, we denote its corresponding interval variable as $\mathbf{x} = [\underline{x}, \bar{x}]$.

- **Sum of two intervals** - $\mathbf{x} + \mathbf{y} = [\underline{x} + \underline{y}, \bar{x} + \bar{y}]$

- **Product of a scalar and an interval** - $\alpha\mathbf{x} = \begin{cases} [\alpha\underline{x}, \alpha\bar{x}], & \alpha > 0 \\ [\alpha\bar{x}, \alpha\underline{x}], & \alpha < 0 \end{cases}$

- **Product of two intervals** - $\mathbf{x} \times \mathbf{y} = [\min(\underline{xy}, \bar{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\bar{y}), \max(\underline{xy}, \bar{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\bar{y})]$

- **Reciprocal of an interval** - $1/\mathbf{x} = \begin{cases} [1/\overline{x}, 1/\underline{x}], & 0 \notin [\underline{x}, \overline{x}] \\ [-\infty, 1/\underline{x}], & 0 = \overline{x} \\ [1/\overline{x}, \infty], & 0 = \underline{x} \\ [-\infty, \infty], & 0 \in (\underline{x}, \overline{x}) \end{cases}$
- **Quotient of two intervals** - $\frac{\mathbf{x}}{\mathbf{y}} = \mathbf{x} \times 1/\mathbf{y}$
- **Square of an interval** - $\mathbf{x}^2 = \begin{cases} [0, \max(\underline{x}^2, \overline{x}^2)], & 0 \in [\underline{x}, \overline{x}] \\ [\overline{x}^2, \underline{x}^2], & \overline{x} < 0 \\ [\underline{x}^2, \overline{x}^2], & \underline{x} > 0 \end{cases}$
- **Square root of an interval** - $\sqrt{\mathbf{x}} = \begin{cases} [-\sqrt{\overline{x}}, \sqrt{\overline{x}}] & \overline{x} \geq 0 \\ \phi & \overline{x} < 0 \end{cases}$

Forward Propagation

We propagate variable bounds to obtain lower and upper bounds for each constraint of the problem during forward propagation. Given a quadratic constraint of the form

$$l \leq \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ij} x_i x_j + \sum_{i=1}^{n} c_i x_i \leq u$$

We define $Q(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ij} x_i x_j + \sum_{i=1}^{n} c_i x_i$ and the interval $\mathbf{Q}(\mathbf{x}) := [Q_l^*, Q_u^*]$ where Q_l^*, Q_u^* is the optimal solution to the problem $\min\{Q_u - Q_l | Q(x) \in [Q_l, Q_u], x \in \mathbf{x}\}$ and $\mathbf{x}$ is the box defined by the corners $(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_n)^T, (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)^T$. $\mathbf{Q}(\mathbf{x})$ is the best possible bound for the constraint with the given bounds on the variables. In general, we get an interval which is larger than $\mathbf{Q}(\mathbf{x})$.

We begin by substituting each variable in $Q(x)$ by its interval counterpart. Using interval arithmetic we then obtain an interval $\mathbf{I}(\mathbf{x}) \supseteq \mathbf{Q}(\mathbf{x})$. Then, updated bounds on the constraint are $\mathbf{I}(\mathbf{x}) \cap [l, u]$. Note, if $[l, u] \supseteq \mathbf{I}(\mathbf{x})$, then the constraint is redundant and can be removed from the problem.

Example 2.1. Consider the quadratic constraint in $\mathbb{R}^2$

$$2x_1^2 - x_2^2 + 5x_1 - 4x_2 \leq 1, x_1 \in [0, 4], x_2 \in [-2, 2].$$

Here $l = -\infty, u = 1, Q(x) = 2x_1^2 - x_2^2 + 5x_1 - 4x_2$ and if we minimize and maximize Q over the given bounds of the variables then we get $\mathbf{Q}(\mathbf{x}) = [-12, 56]$. Now if we replace every variable in $Q(x)$ with its interval variable and apply interval arithmetic operations we get

$$\begin{aligned}
\mathbf{I}(\mathbf{x}) &= 2\mathbf{x_1^2} - \mathbf{x_2^2} + 5\mathbf{x_1} - 4\mathbf{x_2} \\
&= 2 \times [0,4]^2 + (-1) \times [-2,2]^2 + 5 \times [0,4] + (-4) \times [-2,2] \\
&= 2 \times [0,16] + (-1) \times [0,4] + [0,20] + [-8,8] \\
&= [0,32] + [-4,0] + [0,20] + [-8,8] \\
&= [-12,60]
\end{aligned}$$

Thus the new bounds on the constraint are $[-12, 1]$.

For a quadratic constraint, it is clear that if all the bounds on variables are finite, then finite bounds on the constraint can be computed.

Backward Propagation

Once we have forward propagated variable bounds on all constraints, we use constraint bounds to infer new bounds on the variables. For a given interval $\mathbf{I}(\mathbf{x})$ on the quadratic function $Q(x)$ we define $\mathbf{x}^*$ as the smallest box such that if $x \in \mathbf{x}, Q(x) \in \mathbf{I}(\mathbf{x})$ then $x \in \mathbf{x}^*$. The following inverse interval arithmetic operations give us new bounds on the variables.

- **Sum of p terms** - if $z = \sum_{k=1}^{p} t_k$ then $\mathbf{t_i} = \mathbf{z} - \sum_{\substack{k=1 \\ k \neq i}}^{p} \mathbf{t_k}$
- **Linear term** - if $z = at, a \in \mathbb{R}$ then $\mathbf{t} = (1/a) \times \mathbf{z}$
- **Quadratic term** - if $z = t^2$ then $\mathbf{t} = \sqrt{\mathbf{z}}$
- **Bilinear term** - if $z = t_1 t_2$ then $\mathbf{t_1} = \frac{\mathbf{z}}{\mathbf{t_2}}$

Example 2.2. Let us again consider the constraint illustrated in Example 2.1.

$$2x_1^2 - x_2^2 + 5x_1 - 4x_2 \leq 1, x_1 \in [0,4], x_2 \in [-2,2]$$

Using forward propogation we have derived $\mathbf{I}(\mathbf{x}) = [-12, 1]$ and we have calculated the bounds for each term. We note that $\mathbf{x}^* = [0, 1.5894] \times [-0.2679, 2]$

For each term we find

$$
\begin{aligned}
2 \times \mathbf{x_1^2} &= \mathbf{I(x)} - (\mathbf{x_2^2} - 5\mathbf{x_1} + 4\mathbf{x_2}) \\
&= [-40, 13] \\
\mathbf{x_1} &= [-\sqrt{6}, \sqrt{6}] \\
-\mathbf{x_2^2} &= \mathbf{I(x)} - (-2\mathbf{x_1^2} - 5\mathbf{x_1} + 4\mathbf{x_2}) \\
&= [-72, 9] \\
\mathbf{x_2} &= [-\sqrt{72}, \sqrt{72}] \\
5 \times \mathbf{x_1} &= \mathbf{I(x)} - (-2\mathbf{x_1^2} + \mathbf{x_1^2} + 4\mathbf{x_2}) \\
&= [-52, 13] \\
\mathbf{x_1} &= [-10.4, 2.6] \\
-4 \times \mathbf{x_2} &= \mathbf{I(x)} - (-2\mathbf{x_1^2} + \mathbf{x_2^2} - 5\mathbf{x_1}) \\
&= [-64, 5] \\
\mathbf{x_2} &= [-1.25, 16]
\end{aligned}
$$

Thus $\mathbf{x_1} = [-\sqrt{6}, \sqrt{6}] \cap [-10.4, 2.6] \cap [0, 4] = [0, 2.4495]$ and $\mathbf{x_2} = [-\sqrt{72}, \sqrt{72}] \cap [-1.25, 16] \cap [-2, 2] = [-1.25, 2]$.

2.3.3 FBBT for a univariate quadratic expression

In the previous section, we used the most straightforward bound propagation technique to identify better bounds on variables. Although inexpensive, it computes very weak bounds on several instances. In this section, we describe a computationally inexpensive method to get better bounds for some cases of quadratic constraints. We have implemented these techniques in `mglob`, but the ideas presented in this section are directly taken from [60].

Let us consider a univariate quadratic expression $q(x) = ax^2 + bx$. We know that $q(x)$ is a parabola with a minimum or maximum depending on the sign of a. We can use this to get better bounds on the expression. Once we know $q(x) \in [l, u]$, we can use quadratic formula to get bounds on the variable during backward propagation. Next, we describe the forward and

backward propagation for univariate quadratic expressions.

Forward Propagation

Consider $q(x) = ax^2 + bx, x \in [\underline{x}, \bar{x}]$ then we want to l, u such that $q(x) \in [l, u]$. We assume both $a, b \neq 0$ because we $a = 0$ then we have a linear term and if $b = 0$ then we can do forward propagation as described in Section 2.3.2. Now, if $a > 0$ we know that $q(x)$ is a convex parabola and has a minimum at $x^* = -\frac{b}{2a}$. Thus if $x^* \in [\underline{x}, \bar{x}]$, we set $l = q(x^*) = -\frac{b^2}{2a}, u = \max\{a\underline{x}^2 + b, a\bar{x}^2 + b\}$. If $x^* < \underline{x}$, we set $l = a\underline{x}^2 + b, u = a\bar{x}^2 + b$ and if $x^* > \bar{x}$, we set $l = a\bar{x}^2 + b, u = a\underline{x}^2 + b$. Similarly, if $a < 0$ then $q(x)$ is concave and we can find the values of l, u based on the region in which $q(x)$ attains its maximum.

Example 2.3. Let us again consider the constraint illustrated in example 2.1.

$$2x_1^2 - x_2^2 + 5x_1 - 4x_2 \leq 1, x_1 \in [0, 4], x_2 \in [-2, 2]$$

We define $q_1(x) = 2x_1^2 + 5x_1, q_2(x) = -x_2^2 - 4x_2, Q(x) = q_1(x) + q_2(x)$. $q_1(x)$ is convex with minimum at $x^* = -\frac{5}{4} < 0$. Therefore, we get $q_1(x) \in [0, 52]$. Also, $q_2(x)$ is concave with maximum at $x^* = -2 \in [-2, 2]$. Therefore, we get $q_2(x) \in [-12, 4]$. Thus, $\mathbf{I}(\mathbf{x}) = [-12, 56]$. And the bounds on the constraints are $[-12, 1]$.

Backward Propagation

For backward propagation of a univariate quadratic expression, we use the quadratic formula to get the bounds on the variables. Consider $q(x) = ax^2 + bx, x \in \mathbf{x}, q(x) \in [l, u]$ then we want to find $\mathbf{x}'$ such that $x \in \mathbf{x}'$. Note, $q(x) = ax^2 + bx \in [l, u] \implies ax^2 + bx - l \geq 0, ax^2 + bx - u \leq 0$. Now, if $a > 0$ then using $ax^2 + bx - l \geq 0$ we get $x \in \left[-\infty, \frac{-b-\sqrt{b^2+4al}}{2a}\right] \cup \left[\frac{-b+\sqrt{b^2+4al}}{2a}\right]$ and using $ax^2 + bx - u \leq 0$ we get $x \in \left[\frac{-b-\sqrt{b^2+4au}}{2a}, \frac{-b+\sqrt{b^2+4au}}{2a}\right]$. Finally,

$$\mathbf{x}' = \mathbf{x} \cap \left(\left[-\infty, \frac{-b-\sqrt{b^2+4al}}{2a}\right] \cup \left[\frac{-b+\sqrt{b^2+4al}}{2a}\right] \right) \cap \left[\frac{-b-\sqrt{b^2+4au}}{2a}, \frac{-b+\sqrt{b^2+4au}}{2a}\right].$$

Similarly we can find updated bounds on x when $a < 0$.

Example 2.4. In Example 2.3 we derived bounds for $Q(x)$ using forward propagation of univariate quadratic expressions $q_1(x), q_2(x)$ as $\mathbf{I}(\mathbf{x}) = [-12, 1]$. We also derived bounds

on $q_1(x) \in [0,52], q_2(x) \in [-12,4]$. Now backward propagation of $\mathbf{I}(\mathbf{x})$ to $q_1(x), q_2(x)$ we see that $q_1(x) \in ([-12,1] - [-12,4]) \cap [0,52] = [0,13]$. Now using $2x_1^2 + 5x_1 \geq 0$ we get $x_1 \in [-\infty, -2.5] \cup [0,\infty]$ and using $2x_1^2 + 5x_1 - 13 \leq 0$ we get $x_1 \in [-4.0894, 1.5894]$. Thus, we get $x \in [0, 1.5894]$. Similarly computing the bounds using $q_2(x)$ we get $x_2 \in [-0.2679, 2]$.

2.3.4 Adding Default Bounds

After employing the FBBT operations described in the previous sections it may happen that we still have variables with infinite bounds. Jeroslow [94] showed that there cannot be any algorithm for solving quadratic problems with unbounded integer variables. Many relaxation techniques for MIQCQO problems depend on having finite bounds for the variables, see Section 1.5.

Thus after employing the FBBT if we do not have finite bounds on some variables then we assume default bounds for those variables. For both lower and upper bounds, we first compute the largest magnitude of that bound among all the variables that has a finite value for that bound. Let that magnitude be m then we set the default bound for all other variables to be $\pm 100m$ where the sign is $+$ if the we need default upper bound and it is $-$ for default lower bound. It may happen that m is 0 or all variables do not have finite bounds still. In that case we set the default bound value to be ± 1000 depending on the bound type.

2.3.5 Optimality Based Bound Tightening

In the previous sections we have looked at methods which depend on a single constraint of the original problem and derives bound based on the feasibility of the constraint. In this section we look at a method where we first obtain a valid tractable relaxation and compute bounds based on the feasibility of the entire relaxation. We solve a series of linear optimization problems whose objective function is minimizing or maximizing certain variable and the feasible region is same as that of the linear relaxation of (Q) along with additional constraint that bounds the objective function based on the relaxation solution obtained. This method is known as *Optimality Based Bound Tightening (OBBT)*.

OBBT requires a relaxation for (Q). We use McCormick relaxation (1.11) described in Section 1.5.1. Let R be the feasible region of the McCormick relaxation and let $\hat{z}$ be the optimal objective value of the relaxation, then OBBT solves a sequence of LO problems as follows.

$$\begin{aligned} \min / \max \quad & x_i \\ \text{s.t.} \quad & (x,y) \in R \\ & \sum_{(i,j)\in E_0} q^0_{ij} y_{ij} + c_0^T x \geq \hat{z} \end{aligned}$$

This gives us lower and upper bounds for each variable by solving $2n$ LPs. OBBT is computationally very expensive and we wish to eliminate solving those LPs that are necessarily not going to improve the bounds of a variable. To do so we use a filtering approach for solving OBBT problems.

We begin by creating a set of objective functions

$$O = \{\min x_i \mid x_i \in NL\} \cup \{\max x_i \mid x_i \in NL\}$$

where NL is the set of variables that appear in any product term or square term in (Q). Let $\tilde{x}$ be the optimal solution of the linear relaxation of the problem. For all $x_i \in NL$, if $\tilde{x}_i \leq \underline{x}_i + \varepsilon$ then remove $\min x_i$ from O and if $\tilde{x}_i \geq \bar{x}_i - \varepsilon$ then remove $\max x_i$ from O. We now choose an objective function from O and remove it from O. We solve the OBBT problem with the chosen objective function. We again filter O as described above using the optimal solution of the OBBT problem. We repeat this process until O is empty. Since optimal solution for an LP contains many variables at one of their bounds, this strategy reduces the number of LP solves dramatically.

2.3.6 Computational Results

We have implemented the three bound tightening techniques described in the previous sections in `mglob`. Given an instance we do FBBT as described in Section 2.3.2 on all nodes of the sB&B tree. We call this technique as simpleBT in this section. We then do FBBT for univariate

quadratic expression as described in Section 2.3.3 on all nodes of the sB&B tree and call this technique as univarBT in this section. If any variable appearing in a quadratic term still does not have finite bounds then we add default bounds as described in Section 2.3.4. The problem is then transformed with the addition of auxiliary variables and we create the McCormick relaxation. We then solve this relaxation to get a lower bound on the problem. We then solve OBBT problems as described in Section 2.3.5 at the root node only. We will call this technique as OBBT in this section.

We selected all instances with either nonconvex quadratic objective or nonconvex quadratic constraints from the MINLPLib dataset [43]. There are 685 such instances. We removed 6 instances which got solved even before OBBT was done. We also removed instances which takes more than 900 seconds for root node processing. Instances that are removed from the test set are those which have number of variables, or number of constraints, or the number of quadratic terms are very high. This increases the time taken in processing the root node. Root node processing includes presolving, creating a root relaxation and OBBT iterations. We finally have 499 instances for which computational results have been presented here. We run `mglob` on three different settings to compare the effectiveness of the techniques presented previously. In the first setting we do only simpleBT and add default bounds if necessary. This is our baseline configuration with which we compare other two settings. In the second setting we do simpleBT, univarBT, and add default bounds if necessary. And the third setting we do all three techniques and add default bounds whenever necessary.

Table 2.3 summarizes the results obtained. We measure the range of variables before processing the root node. Range of a variable is the difference between its lower and upper bounds. We then take the sum of the range of all variables for every instance. We have reported the total range of variables for all instances here. We see that doing univarBT after simpleBT there is a marginal 0.09% improvement in the range of variables. We also note that doing univarBT affects only 4 instances in our test set all other instances the range of variables remain unchanged. While doing OBBT we get more than 8% improvement in the range of variables. 236 instances had bounds improvements reported after OBBT was done. We see that similar results are obtained if we compare the number of instances solved within time limit or the shifted geometric mean of time for solved instances.

The bound tightening techniques described above improve the solver's performance,

	simpleBT	simpleBT + univarBT	simpleBT + univarBT + OBBT
Total range of variables at root node	2.7319E+14	2.7295E+14	2.4901E+14
Percent change in range of variables	0	0.09	8.77
Number of instances solved	242	243	246
Shifted geometric mean of time for instances solved by all techniques	2.9937	2.9911	2.9062

Table 2.3: Summary of results comparing bound tightening techniques

particularly OBBT, which can significantly reduce solve time. We have integrated these bound tightening techniques in `mglob` solver along with other presolving techniques.

2.4 Conclusion

In this chapter, we have described three presolving techniques that we implemented in `mglob` for MIQCQO problems. We conclude this chapter by presenting how our techniques have been integrated into the solver. Given an MIQCQO instance `mglob` first presolves the problem to extract meaningful information and deduce reduction such that it becomes easier to solve the problem. The first step is to check if the instance is a maximization problem and convert it to a minimization problem by multiplying the objective function with -1. We iteratively do the following presolving operations until there is no significant change.

1. Tighten variable bounds using linear constraints.

2. Check for duplicate or redundant constraints.

3. For constraints of the type $ax_1 + bx_2 = 0$ we substitute $x_2 = -\frac{a}{b}x_1$ in the problem.

4. Dual fixing of variables to one of its bounds.

5. simpleBT for quadratic constraints.

6. univarBT for quadratic constraints.

We then check for the convexity of each constraint, and if all constraints are convex and the objective function is also convex, we forward the problem to `qg`, a convex MINLO solver in Minotaur. We then transform the problem by adding auxiliary variables of the type $y_{ij} = x_i x_j$. For problems with no integer variables, we use a multi-start heuristic to get an upper bound for the problem. Our multi-start heuristic randomly selects initial points, and then we use a local NLP solver to get a local optimal solution. We then presolve the problem again to propagate variable bounds of original x variables to auxiliary y variables. We then create a McCormick relaxation and solve it. Finally, we do OBBT as described in Section 2.3.5 and update the bounds on the variables accordingly.

Several presolving techniques can be implemented. We have seen in Algorithm 2.1 that we derive subgraphs for a quadratic constraint, but we use this only to check convexity. One can use these separable quadratic functions and develop better relaxations and reformulations. For example, if some parts of the constraints are convex, then we can use tangent hyperplanes for those parts to derive better cutting planes or develop specialized relaxations. We can also study the impact of several other bound tightening techniques described in the literature.

Chapter 3

Cutting Planes for Quadratically Constrained Optimization Problems

We consider a Quadratically Constrained Optimization (QCO) problem with a single quadratic constraint of the following form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & x^T Q x + a^T x \leq d, \\ & Gx = h, \\ & \underline{x} \leq x \leq \overline{x}, \end{aligned} \tag{QCP1}$$

where $c, a \in \mathbb{R}^n, d \in \mathbb{R}, G \in \mathbb{R}^{k \times n}, h \in \mathbb{R}^k$, and the symmetric matrix $Q \in \mathbb{R}^{n \times n}$ are given as inputs. The cutting plane procedure proposed here considers one quadratic constraint at a time for notational convenience. It works for any number of quadratic constraints as described in the computational experiments in Section 3.6.

Given a QCO of the form (QCP1) above, Branch-and-cut algorithms require a suitable relaxation. A relaxation should be easy to solve and at the same time be a close approximation

to the original problem. A linear relaxation is often used as it is easy to solve repeatedly in a branch-and-cut framework. McCormick [112] inequalities are commonly used to obtain a linear relaxation of (QCP1). Simplex method is then used to solve this linear relaxation because of two practical reasons. First, simplex method has superior warm starting ability, that is, if a basic solution is known then it is relatively simple to restart the algorithm after the problem is modified, and second, cutting planes can sometimes be derived from the simplex tableau, for example, Gomory Mixed Integer cuts [80], Gomory fractional cuts [79]. The procedure proposed here is similar in vein to these two methods. A gist of the method is first provided along with an example, and a detailed description is provided subsequently.

Suppose we have solved a linear relaxation (LP) of (QCP1) using the simplex method and obtained a solution, say x^*, not feasible to the quadratic constraint. The main idea proposed here is to first substitute some or all basic variables in the quadratic constraint using the corresponding row of the simplex tableau. A new quadratic inequality valid for (QCP1) is thus obtained. The substitution ensures that each term in the new quadratic function has at least one nonbasic variable. Each term is then relaxed using McCormick estimators. Since one of the variables in each term is at its bounds, the McCormick estimators are '*tight*' at x^* for the term. The linear inequality obtained as the sum of McCormick estimators will cut off x^*. Here is a toy example to illustrate the procedure.

Example 3.1. Suppose we get the following two rows in the optimal simplex tableau while solving a linear relaxation of a given QCO.

$$\begin{aligned} x_1 + 2x_3 - 3x_4 + 2x_5 &= 0.3, \\ x_2 + x_6 &= 0.5, \\ x_i \in [0,1]\ i &= 1,\ldots,6. \end{aligned} \tag{3.1}$$

Here x_3, x_4, x_5, x_6 are nonbasic variables currently at their lower bounds. A basic feasible solution for the relaxation is $x^* = (0.3, 0.5, 0, 0, 0, 0)$. Further suppose that the QCO has a quadratic constraint $x_1 x_2 \leq x_3$ that is not satisfied by x^*. Substitute x_1 in the quadratic constraint using (3.1) to obtain a new quadratic constraint

$$0.3x_2 - 2x_2x_3 + 3x_2x_4 - 2x_2x_5 \leq x_3 \tag{3.2}$$

that is valid for the given QCO. We can use term-by-term McCormick underestimators to obtain a relaxation of this new quadratic constraint. That is, we use the inequalities $-2x_3 \leq -2x_2x_3, 0 \leq 3x_2x_4,$ and $-2x_5 \leq -2x_2x_5$ to obtain

$$0.3x_2 - 3x_3 - 2x_5 \leq 0.$$

This inequality is valid for the given QCO, and it cuts off x^*.

The rest of the chapter is outlined as follows. In Section 3.1 we describe the McCormick estimators and their key properties used in the procedure. In Section 3.2 we review existing literature. We then describe our procedure in detail in Section 3.3. We show how our procedure is analogous to Gomory's fractional cuts in Section 3.4. We next show some connections of our procedure with Reformulation Linearization Technique (RLT) in Section 3.5. Finally, in Section 3.6 we discuss some computational results to show the efficiency of the cuts we generate, and we conclude in Section 3.7.

3.1 Properties of McCormick Estimators

Property 3.2. *Under- and over-estimators of a bilinear function:* For a bilinear function $f : \mathbb{R}^n \to \mathbb{R}$ given by $f(x) = x_i x_j$ for some $i, j \in \{1, \ldots, n\}$, over a given box $B = \{x \in \mathbb{R}^n \mid \underline{x} \leq x \leq \bar{x}\}$ the following inequalities give a pair of underestimators and a pair of overestimators for f over B

$$\left.\begin{array}{l} \underline{x_j}x_i + \underline{x_i}x_j - \underline{x_i}\,\underline{x_j} \\ \overline{x_j}x_i + \overline{x_i}x_j - \overline{x_i}\,\overline{x_j} \end{array}\right\} \leq x_i x_j \leq \left\{\begin{array}{l} \underline{x_j}x_i + \overline{x_i}x_j - \overline{x_i}\,\underline{x_j} \\ \overline{x_j}x_i + \underline{x_i}x_j - \underline{x_i}\,\overline{x_j} \end{array}\right. \tag{3.3}$$

These inequalities are the well known McCormick [112] inequalities for f.

Property 3.3. *McCormick inequalities are tight at bounds:* It is well known that when either x_i or x_j is at its bounds (lower or upper), the under- and over-estimators of f are both *tight* i.e. at least one under- and one over-estimator evaluate to function value at that point. The *tight* under- and over-estimators for four different cases (arising from the condition that one of the

Edge	Underestimator	Overestimator
$x_1 = \underline{x_1}$	$\underline{x_2}x_1 + \underline{x_1}x_2 - \underline{x_1}\underline{x_2}$	$\overline{x_2}x_1 + \underline{x_1}x_2 - \underline{x_1}\overline{x_2}$
$x_2 = \underline{x_2}$	$\underline{x_2}x_1 + \underline{x_1}x_2 - \underline{x_1}\underline{x_2}$	$\underline{x_2}x_1 + \overline{x_1}x_2 - \overline{x_1}\underline{x_2}$
$x_1 = \overline{x_1}$	$\overline{x_2}x_1 + \overline{x_1}x_2 - \overline{x_1}\overline{x_2}$	$\underline{x_2}x_1 + \overline{x_1}x_2 - \overline{x_1}\underline{x_2}$
$x_2 = \overline{x_2}$	$\overline{x_2}x_1 + \overline{x_1}x_2 - \overline{x_1}\overline{x_2}$	$\overline{x_2}x_1 + \underline{x_1}x_2 - \underline{x_1}\overline{x_2}$

Table 3.1: Under- and over-estimators that are tight at the edges of the box $B = [\underline{x_1}, \overline{x_1}] \times [\underline{x_2}, \overline{x_2}]$ for the function $f(x) = x_1 x_2$

two variables is at one of its bounds) are given in Table 3.1. At points when neither variable is at its bounds, there is a gap between the estimators and the function value.

Property 3.4. *Under- and over-estimators of a quadratic function:* Given a general quadratic function $f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_i x_j$, where $q_{ij} \in \mathbb{R}$ for all $i, j \in \{1, \ldots, n\}$, a linear underestimator or overestimator of f over a given box $B := [\underline{x}, \overline{x}]$ can be obtained using the above McCormick estimators for each term, depending on the sign of q_{ij}. For example, one underestimator of f is

$$\sum_{\substack{i=1}}^{n}\sum_{\substack{j=1 \\ q_{ij}>0}}^{n} q_{ij}(\underline{x_j}x_i + \underline{x_i}x_j - \underline{x_i}\underline{x_j}) + \sum_{\substack{i=1}}^{n}\sum_{\substack{j=1 \\ q_{ij}<0}}^{n} q_{ij}(\underline{x_j}x_i + \overline{x_i}x_j - \overline{x_i}\underline{x_j}). \tag{3.4}$$

Note that many underestimators can be obtained by choosing one of the two estimators possible for each term.

Property 3.5. *Tight under- and over-estimators of a quadratic function:* Consider a quadratic function f over a box B as described in Property P3 and an $x^* \in \mathbb{R}^n$ such that for every pair (i, j) with $q_{ij} \neq 0$ at least one of x_i, x_j is at one of its bounds. We can find an under- and over-estimator for f that is *tight* at x^* by selecting an appropriate estimator depending on the sign of q_{ij} (using Table 3.1) for each term in f.

For example, consider the quadratic function $f(x) = x_1^2 - 2x_1x_2 + x_2x_3$ over the box $B = [0,1]^3$ and let $x^* = (0, 0.5, 1)$. Clearly, for every term in f at least one of the variables is at its bounds at x^*. From Table 3.1, we can underestimate x_1^2 with $0, -2x_1x_2$ with $-2x_1$, and x_2x_3 with $x_2 + x_3 - 1$ to obtain a *tight* underestimator $-2x_1 + x_2 + x_3 - 1$ of f. Similarly, a *tight* overestimator for f at x^* is $x_1 + x_2$.

3.2 Literature review

Most state-of-the-art global optimisation solvers for nonconvex problems use Branch-and-Bound algorithms augmented by cutting planes, primal heuristics, presolving, infeasibility analysis etc. [28, 25, 117, 138]. Cutting planes for QCO have also been developed using several approaches both for general purpose QCO and for certain special structures that are commonly seen in applications. Some of these are described below.

Sherali and Alameddine [134] described the Reformulation Linearization Technique (RLT) for bilinear problems. The constraint $x^T Qx + a^T x \leq d$ in (QCP1) is first replaced by the linear constraint $\langle Q, X \rangle + a^T x \leq d$. Then, multiplying any two linear constraints, one gets a new quadratic constraint valid for (QCP1). Substituting $X_{ij} = x_i x_j$ in the new quadratic constraint gives us the RLT constraint. When only bound constraints are multiplied, this approach reduces to McCormick [112] relaxation.

In the presence of linear equality constraints $Gx = h$ in the problem, there are many RLT constraints possible, some of which turn out to be unnecessary. For example, RLTs obtained by multiplying an equality constraint with all the variables individually implies all other RLTs generated using that constraint (see [133]). Liberti [102] show that RLT constraints $X_{ij} = x_i x_i$ corresponding to the nonbasic variables of a certain companion system is sufficient to generate all RLT constraints $X_{ij} = x_i x_j$. Liberti and Pantelides [103] then extend the results obtained above to general sparse MINLPs. They describe a graph theoretical approach to filter out some RLT constraints. Sherali et al. [135] show that given a basis of G along with the corresponding index set B of basic variables and N of nonbasic variables, we need to only relax the RLT constraints $X_{ij} = x_i x_j \forall\ i, j \in N$.

Our procedure is related to the analysis of Sherali et al. [135] as we also rely on the basis matrix to find the cut. However, instead of trying to get an equivalent representation for the first-order RLT relaxation, we propose a fast cutting plane procedure to separate a given basic point. In Section 3.5 we describe a method to add RLT variables. While our procedure only adds the RLT variables corresponding to the bilinear terms present in the new quadratic constraints, Sherali et al. [135] add all the RLT variables without considering the structure of the quadratic constraint present. Also, our computational results show that keeping some basic variables in

the product terms is beneficial practically. More details of key differences are given in Section 3.3.

Audet et al. [19] describe a branch-and-cut approach using the RLT method and give four classes of cutting planes derived from RLT. Adams and Johnson [6] give a first order RLT formulation of Quadratic Assignment Problem. Recently, Bestuzheva et al. [33] give a separation algorithm based on RLT cuts. They identify products of a bound factor and a linear constraint which will not produce a violated inequality. Such products are then discarded, and other products are considered. They also project some linear constraints on a subspace of variables to obtain RLT cuts for a smaller system of inequalities. Luedtke et al. [108] provide several results on the strength of McCormick relaxations for multilinear problems and show that the McCormick relaxation of a bilinear function is within a constant factor of the convex hull at every point within the bounds of the variables. All these methods try to search for an RLT inequality by trying different combinations of linear and bound constraints. As far as we understand, information from the simplex tableau has not been used earlier to generate inequalities that are guaranteed to cut off a basic feasible solution of the linear relaxation.

Intersection cuts can also be derived for (QCP1) as described in [63]. They use the corner polyhedron associated with the optimal basis of the linear relaxation of (QCP1) and derive a polyhedron which does not contain any feasible point to the (QCP1) in its interior. Bienstock et al. [34] develop a cut generation procedure using intersection cuts for polynomial optimisation problems.

Semidefinite programming (SDP) relaxations for QCO are also well studied in the literature. Shor [137] proposed an SDP relaxation of the QCO by relaxing the constraint $X = xx^T$ to $X - xx^T \succcurlyeq 0$. Saxena, Bonami, and Lee [131] provide a disjunctive approach to generate valid inequalities based on their SDP relaxation. Burer and Saxena [42] review methods to obtain linear inequalities from SDP.

Given a QCO of the form (QCP1), it can be relaxed by rewriting the matrix Q as a difference of two positive semidefinite matrices [38, 127, 149]. Another related approach is the αBB underestimators developed by [15] and [8].

3.3 A Procedure for generating cuts

We now describe the algorithm for generating cuts. We first describe it for the canonical form of the linear relaxation because it requires less notation. Then we describe the algorithm for the standard form of the linear relaxation.

3.3.1 Canonical form of the relaxation

Let $R = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ be a linear relaxation of the (QCP1) with $\text{rank}(A) = n$. Note that R may have additional auxiliary variables. We also assume that the constraints $Gx = h, \underline{x} \leq x \overline{x}$ are either implied by or included in $Ax \leq b$. Let $S = R \cap \{x \in \mathbb{R}^n \mid x^T Qx + a^T x \leq d\}$ be the equivalent feasible region of (QCP1). The inequalities $Ax \leq b$ include lower and upper bound constraints on each variable along with any other additional constraints.

At an optimal extreme point of R, n linearly independent constraints from $Ax \leq b$ will be active. Let such a set of active constraints be $Bx \leq b^B$, where B is a nonsingular square matrix. We can add additional slack variables $s^B \geq 0$ such that $Bx + s^B = b^B$. Since all x feasible to R satisfy $Bx + s^B = b^B$ we get, $x = B^{-1}b^B - B^{-1}s^B$. The optimal solution to R has $x^* = B^{-1}b^B, s^{B*} = 0$. Thus, any feasible solution to R (and also S) must satisfy $x = x^* - B^{-1}s^B$. If x^* is feasible to S then we have obtained an optimal solution to S. Otherwise, the quadratic constraint in S must be violated at x^*, i. e. $x^{*T}Qx^* + a^T x^* > d$.

We substitute $x = x^* - B^{-1}s^B$ on one side of $x^T Qx$ to obtain

$$\begin{aligned} & x^T Q(x^* - B^{-1}s^B) + a^T x \leq d \\ \Longrightarrow\ & x^T Qx^* + a^T x - x^T QB^{-1}s^B \leq d. \end{aligned} \tag{3.5}$$

Let $\widetilde{Q} = \left(\widetilde{q_{ij}}\right) = -QB^{-1}$. Then the quadratic inequality

$$x^T Qx^* + a^T x + \sum_{i=1}^{n}\sum_{j=1}^{n} \widetilde{q_{ij}} x_i s_j^B \leq d, \tag{3.6}$$

is valid for S. Now we relax this quadratic inequality using McCormick inequalities (3.3) and Table 3.1 to get

$$x^T Q x^* + a^T x + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ \widetilde{q_{ij}}>0}}^{n} \widetilde{q_{ij}} \underline{x_i} s_j^B + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ \widetilde{q_{ij}}<0}}^{n} \widetilde{q_{ij}} \overline{x_i} s_j^B \leq d. \tag{3.7}$$

At the point $x = x^*, s^{B*} = 0$ the left hand side of the inequality (3.7) evaluates to $x^{*T} Q x^* + a^T x^*$. Since we assumed $x^{*T} Q x^* + a^T x^* > d$ the linear inequality (3.7) cuts off x^*.

Example 3.6. Let $S = \{x \in \mathbb{R}^2 \mid x_1 x_2 \leq 4, 4x_1 - 3x_2 \leq 8, 0 \leq x_1, x_2 \leq 4\}$ and $z = \min\{-x_1 \mid x \in S\}$. The optimal $z, z^* = -3$ obtained at $(3, \frac{4}{3})^T$. Consider the linear relaxation $R = \{x \in \mathbb{R}^2 \mid 0 \leq x_1, x_2 \leq 4, 4x_1 - 3x_2 \leq 8\}$. An optimal solution of R is $x^* = (4, \frac{8}{3})^T$, where constraints $x_1 \leq 4,$ and $4x_1 - 3x_2 \leq 8$ are active. We have $Q = \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 \\ 4 & -3 \end{bmatrix}, b_B = \begin{pmatrix} 4 \\ 8 \end{pmatrix}$. Substituting in (3.6), we obtain the valid quadratic inequality

$$\frac{4}{3} x_1 + 2x_2 - \frac{2}{3} x_1 s_1 + \frac{1}{6} x_1 s_2^B - \frac{1}{2} x_2 s_1^B \leq 4,$$

McCormick underestimators as shown in (3.7) provide the cut

$$\frac{4}{3} x_1 + 2x_2 - \frac{14}{3} s_1^B \leq 4.$$

Substituting the slack variable using the active constraint we get

$$6x_1 + 2x_2 \leq \frac{68}{3}.$$

Solving the problem after adding the cuts improves the lower bound to $z_l = -3.231$. □

In the above procedure when we substitute $x = x^* - B^{-1} s^B$ to obtain (3.5), we substitute only one of x's in $x^T Q x$, one can substitute both the x's to obtain an inequality in only slack s^B variables, i. e.

$$(x^* - B^{-1} s^B)^T Q (x^* - B^{-1} s^B) + a^T (x^* - B^{-1} s^B) \leq d$$

$$\implies x^{*T}Qx^* + a^T x^* - 2x^{*T}QB^{-1}s^B - a^T B^{-1}s^B + s^{BT}B^{-T}QB^{-1}s^B \leq d. \tag{3.8}$$

Let $\widetilde{Q} = \left(\widetilde{q_{ij}}\right) = B^{-T}QB^{-1}$ then the following quadratic inequality is valid for S

$$x^{*T}Qx^* + a^T x^* - 2x^{*T}QB^{-1}s^B - a^T B^{-1}s^B + \sum_{i=1}^{n}\sum_{j=1}^{n}\widetilde{q_{ij}}s_i^B s_j^B \leq d. \tag{3.9}$$

This quadratic inequality can be underestimated using McCormick underestimators for $\sum_{i=1}^{n}\sum_{j=1}^{n}\widetilde{q_{ij}}s_i^B s_j^B$ to obtain a different cut. In this case we will require to compute the bounds on the s^B variables, which can be computed from the equation $Bx + s^B = b^B$ and bounds on x i. e., $\overline{s^B} = b^B - B_+\underline{x} - B_-\bar{x}$, where B_+ is obtained by replacing all the negative entries in B with 0 and B_- is obtained by replacing all the positive entries in B with 0. For every term $\widetilde{q_{ij}}s_i^B s_j^B$ in (3.6), if $\widetilde{q_{ij}} \geq 0$, then 0 an underestimator for the term and if $\widetilde{q_{ij}} < 0$, then either overestimators $\widetilde{q_{ij}}\overline{s_j^B}s_i^B$ and $\widetilde{q_{ij}}\overline{s_i^B}s_j^B$ can be used (see Table 3.1 again).

Example 3.7. Consider again the problem from Example 3.6. We substitute Q, B, b^B in (3.8) to obtain the following quadratic inequality

$$\frac{4}{3}s_1^B s_1^B - \frac{1}{3}s_1^B s_2^B - 8s_1^B + \frac{4}{3}s_2^B \leq -\frac{20}{3}.$$

Note that $s_1^B \in [0,4], s_2^B \in [0,20]$. Also, $s_1^B s_1^B$ is underestimated using 0 and $s_1^B s_2^B$ is overestimated using either $20s_1^B$ and $4s_2^B$ to obtain the cuts

$$\begin{aligned} -\frac{44}{3}s_1^B + \frac{4}{3}s_2^B &\leq -\frac{20}{3}, \\ -8s_1^B &\leq -\frac{20}{3}. \end{aligned}$$

Substituting the slack variables and simplifying gives the cuts

$$\begin{aligned} 7x_1 + 3x_2 &\leq 31, \text{ and} \\ x_1 &\leq \frac{19}{6}. \end{aligned}$$

And the lower bound increases to -3.167. □

3.3.2 Standard form of linear relaxation

In this section we describe our procedure for the standard form of a linear relaxation of (QCP1). Suppose we are given a QCO of the form (QCP1) and its linear relaxation $R = \min\{c^T x \mid Ax = b, \underline{x} \leq x \leq \bar{x}\}$. We assume that all the additional variables, either substituted for quadratic terms or added as slack/surplus variables to obtain the standard form of the relaxation, are included in x, and finite bounds are available for all variables. If R is infeasible, then so is (QCP1), and no cuts are required. Let x^* be the optimal solution of R. If $x^{*T}Qx^* + a^T x^* \leq d$, then x^* is optimal to (QCP1). Otherwise, let B denote the optimal basis matrix identified by the simplex method and N denote the submatrix of A associated with nonbasic variables. The simplex method provides linear equalities of the form $x_B = B^{-1}b - B^{-1}Nx_N$. For every term $x_i x_j$ in $x^T Qx$ with nonzero q_{ij}, if both $x_i,\ x_j$ are basic variables then substitute at least one of the variables with its corresponding simplex row. If one of the two variables is a nonbasic variable, then either substitute the basic variable or leave the term as is. This step ensures that the quadratic function obtained after substitution has at least one nonbasic variable in each term. This substituted quadratic function can then be relaxed using McCormick estimators to obtain a cutting plane. This gives us Algorithm 3.1 to separate x^* from the feasible region of (QCP1). In the algorithm 3.1, while defining the relaxation R we have assumed y to be bounded by $[\underline{y}, \bar{y}]$ which may not be readily available for all auxiliary variables or the slack variables. In that case, we have to infer the bounds on these variables. This is a valid assumption to make since we assume x variables to be bounded it is possible to infer bounds on the auxiliary variables or for the slack variables.

Theorem 3.8. In Algorithm 3.1, $f(x^*) = \pi^T x^* + c$. Further, the inequality $\pi^T x \leq \pi_0$ is valid for (QCP1) and cuts off x^*.

Proof. In Algorithm 3.1, $g(x)$ is a quadratic function obtained by substituting some variables in $f(x)$ by their corresponding rows of simplex tableau, therefore, it is clear that $f(x) = g(x)$ for every point $x \in R$. In particular, $f(x^*) = g(x^*)$. Each quadratic term in $g(x)$ has at least one nonbasic variable. Property (P4) in Section 3.1 ensures $g(x^*) = \pi^T x^* + c$. Since $f(x^*) > d, \pi^T x^* > d - c = \pi_0$.

Since $f(x) = g(x)$ for x feasible to (QCP1), an underestimator of g is also an underestimator of f for all feasible points of (QCP1). Hence, the inequality $\pi^T x \leq \pi_0$ is

Algorithm 3.1 Cut generating algorithm

Input: A linear relaxation $R := \min\{c^T x \mid Ax = b, \underline{x} \leq x \leq \bar{x}\} \in \mathbb{R}^p$ of a QCO of the form (QCP1), a basic solution x^* with $x^{*T} Q x^* + a^T x^* > d$, set of indices for basic and nonbasic variables B, N respectively, and a row of the optimal simplex tableau for each basic x_i i. e. $x_i + \sum_{j \in N} \alpha_{ij} x_j = \beta_i \; \forall \; i \in B$.

Output: $(\pi, \pi_0) \in \mathbb{R}^{p+1}$ such that $\pi^T x^* > \pi_0$

procedure GENERATECUTS
  $f(x) \leftarrow \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j$
  $g(x) \leftarrow 0$
  for every quadratic term $x_i x_j$ of f, where $q_{ij} \neq 0$ **do**
    if $i, j \in B$ **then**
      $h(x) \leftarrow q_{ij}(\beta_i - \sum_{k \in N} \alpha_{ik} x_k) x_j$
      (Optional) substitute x_j by $(\beta_j - \sum_{k \in N} \alpha_{jk} x_k)$ in $h(x)$
    else
      $h(x) \leftarrow q_{ij} x_i x_j$
      if $i \in B$ **then**
        (Optional) substitute x_i by $(\beta_i - \sum_{k \in N} \alpha_{ik} x_k)$ in $h(x)$
      if $j \in B$ **then**
        (Optional) substitute x_j by $(\beta_j - \sum_{k \in N} \alpha_{jk} x_k)$ in $h(x)$
    $g(x) \leftarrow g(x) + h(x)$
  for every quadratic term $x_i x_j$ of g **do**
    if coefficient of $x_i x_j$ is nonnegative **then**
      underestimate the term using the appropriate underestimator from Table 3.1
    else
      underestimate the term using the appropriate overestimator from Table 3.1
  Let $\pi^T x + k$ be the linear underestimator obtained. $\pi_0 \leftarrow d - k$
return (π, π_0)

valid for (QCP1). □

The cutting planes derived are computationally cheap since no additional linear programs are solved and no matrix factorisation or eigenvalue are required. Note that there are several cuts possible for a quadratic constraint. If both variables of a quadratic term are basic, then one can substitute either one of them or both (steps 6, 7 of Algorithm 3.1). After $g(x)$ is obtained from step 14 of Algorithm 3.1, it is possible that some quadratic terms in $g(x)$ have both the variables at their bounds (for example, during substitution if one substitutes both the basic variables of a quadratic term) and it may happen that both the underestimators for that term can be used for underestimating the term. In that case, we can select either of the estimators or can take a convex combination of the two. Regardless of how one selects the variables or the estimators, the cut violation at x^* is the same. Hence other criteria like sparsity of the cut, range of coefficients etc. maybe needed to pick an appropriate cut. We now give a small example to show that the cutting plane method converges to the optimal solution in the limit.

Example 3.9. Consider the problem $\min\{x_1 \mid x_1 + 2x_2 = 1, x_2 = x_1^2, 0 \leq x_1, x_2 \leq 1\}$. Let $R = \{x_1 + 2x_2 = 1, 0 \leq x_1, x_2 \leq 1\}$. Optimal solution $(0, \frac{1}{2})$ can be cut off using the McCormick overestimator for the constraint $x_1^2 - x_2 \geq 0$, $x_1 - x_2 \geq 0$. Let us call this iteration - 0. Applying the above procedure after adding a surplus variable, gives us the cut $x_1 - x_2 \geq \frac{2}{11}$ in the original space of variables. Define the sequence $\{b_k\}$ of right hand sides of the cuts added in each iteration e.g. $b_0 = 0, b_1 = \frac{2}{11}$, etc. Now we show that $b_{k+1} > b_k \forall\ k$ and the cuts generated in the k^{th} iteration is of the form $x_1 - x_2 \geq b_k$. Assume this is true for some k, then in $(k+1)^{\text{th}}$ iteration the active constraints will be $x_1 - x_2 \geq b_k, x_1 + 2x_2 = 1$, and therefore the optimal solution $x_{k+1} = (\frac{1+2b_k}{3}, \frac{1-b_k}{3})$. When we apply Algorithm 3.1 using these as active constraints we get the following cut

$$x_1 - x_2 \geq \frac{11b_k + 2}{4b_k + 11}.$$

Setting $b_{k+1} = \frac{11b_k+2}{4b_k+11}$ and observing that $b_{k+1} > b_k$ whenever $b_k \geq 0$ completes the proof using induction. In the limit $b_k \to \frac{1}{4}$ and $x_k \to (\frac{1}{2}, \frac{1}{4})$, which is optimal solution to the problem. □

We do not know whether the method always converges. A pure cutting plane algorithm for general QCO is still an open question. However, the above example shows that it can be

slow. However, it can still be used in a branch-and-cut framework to tighten the relaxations.

It is not necessary to use the optimal basis of the relaxation, sometimes using a non-optimal basis or an infeasible basis may result in a better cut as shown below.

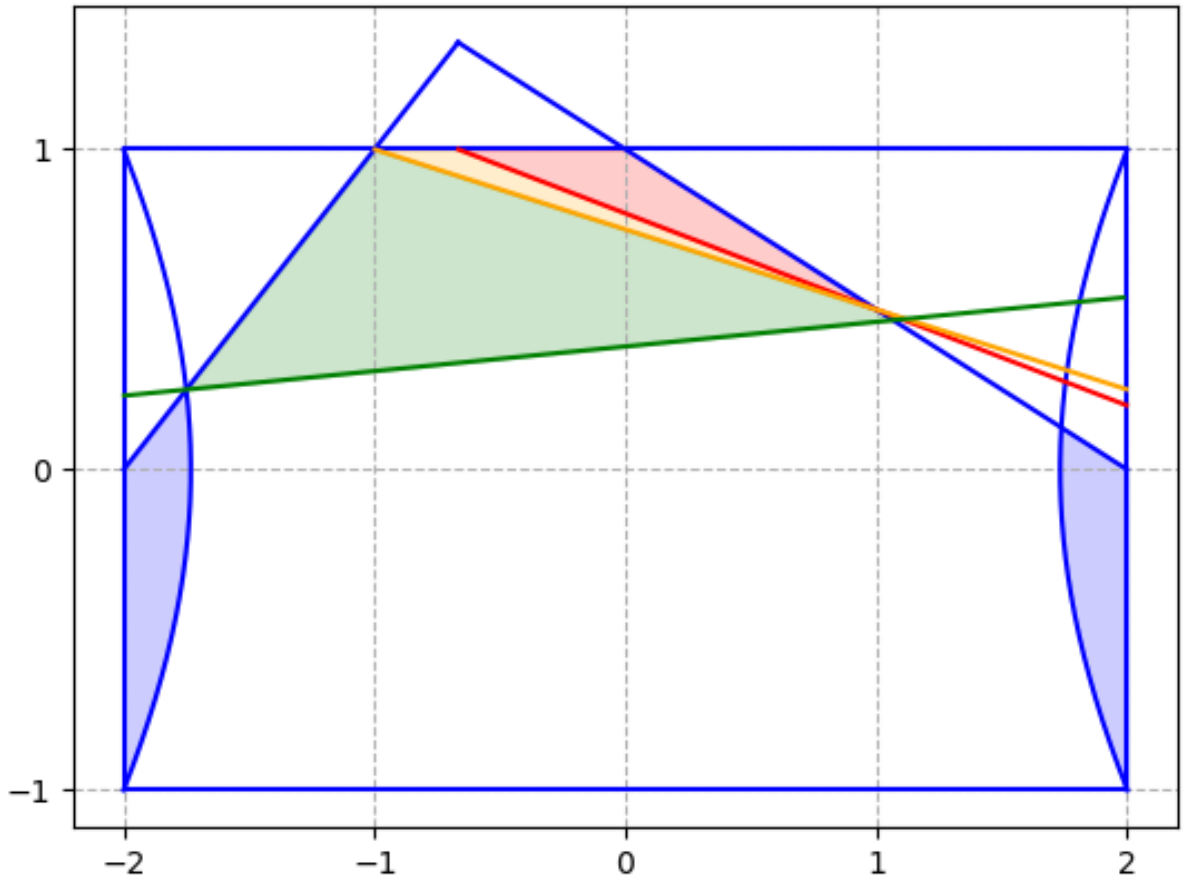
Example 3.10. Let P be the problem $\min\{-x_1 - 4x_2 \mid x_1^2 - x_2^2 \geq 3, x_1 + 2x_2 \leq 2, -x_1 + x_2 \leq 2, -2 \leq x_1 \leq 2, -1 \leq x_2 \leq 1\}$. The feasible region of P is shown in Figure 3.1. The optimal solution is $x^* = (1.74, 0.13)^T$ with the optimal objective value -2.26 Let $R = \min\{-x_1 - 4x_2 \mid x_1 + 2x_2 + s_1 = 2, -x_1 + x_2 + s_2 = 2, -2 \leq x_1 \leq 2, -1 \leq x_2 \leq 1, 0 \leq s_1 \leq 6, 0 \leq s_2 \leq 5\}$ be a relaxation of P. The optimal solution to R is $x^* = (0,1)^T, s^* = (0,1)^T$ with the optimal objective value $z^* = -4$. Algorithm 3.1 gives the cut $6x_1 + 20x_2 \leq 16$ and the lower bound increases to -3.33.

Instead, consider a sub optimal corner point $\hat{x} = (2,0)^T, \hat{s} = (0,4)^T$. Note that the quadratic constraint is already satisfied at this point. Algorithm 3.1 gives the cut $x_1 + 4x_2 \leq 3$, and the lower bound increases to -3. This cut dominates the cut obtained from the optimal basis (see Figure 3.1). Now choose yet another "corner" point, say, $\hat{x} = \left(-\frac{2}{3}, \frac{4}{3}\right)^T, \hat{s} = (0,0)^T$ which is infeasible to the relaxation. Algorithm 3.1 provides the cut $x_1 - 13x_2 \geq -5$ and the lower bound increases to -2.93. This cut dominates the other two cuts obtained.

Thus, carefully choosing a basis to obtain the cuts can impact the performance of Algorithm 3.1. The use of non-optimal bases to generate cuts has been used in MILP literature as well. For example, see Section 5.1.1 in [55] and the discussion about Figure 5.2. Although this seems counterintuitive that non-optimal bases may generate cuts, theoretically that is possible for certain polyhedron. The example described above also points to the same ideas in the context of cuts generated by Algorithm 3.1. While it is guaranteed that if the corner point is infeasible to the quadratic constraint Algorithm 3.1 will generate a cut that will separate the point from the feasible region of (QCP1), using the optimal basis to generate the cut will depend on the objective function and in general will be better to improve the bound since it will guarantee cutting the LP optimal solution. Using non-optimal bases may even generate redundant constraints or cut region which is not in the direction of objective function. One should use non-optimal bases in the cuts only if the given instance has such structures amenable to it.

We now compare our cut generating algorithm with the full level-1 RLT relaxation of the

Figure 3.1: Cuts generated for Example 3.10.



Note : Blue shaded region shows the feasible region of P. Red line is the cut generated from the optimal solution $x^* = (0,1)^T, s^* = (0,1)^T$ and red shaded region is the region cut off by this cut. Orange line is the cut generated from the sub optimal corner point $\hat{x} = (2,0)^T, \hat{s} = (0,4)^T$ and orange shaded region is the region cut off by this cut. Green line is the cut generated from the infeasible corner point $\hat{x} = \left(-\frac{2}{3}, \frac{4}{3}\right)^T, \hat{s} = (0,0)^T$ and green shaded region is the region cut off by this cut.

problem in Example 3.10. RLT relaxation of the problem can be obtained by taking the product of two linear constraints or squaring linear constraint and substituting the product of variables or a square of a variable with auxiliary variables as described in Section 1.5.3. For example, we can take the product of $x_1 + 2x_2 \leq 2$ and $-x_1 + x_2 \leq 2$ to obtain

$$\begin{aligned}(2 - x_1 - 2x_2)(2 + x_1 - x_2) &\geq 0\\ 4 - 6x_2 - x_1^2 + 2x_2^2 - x_1x_2 &\geq 0\\ 4 - 6x_2 - X_{11} + 2X_{22} - X_{12} &\geq 0.\end{aligned}$$

Similarly other products can be taken to obtain more constraints. Finally we obtain 21 additional constraints and we add 3 auxiliary variables to the relaxation. Solving the full level-1 RLT relaxation gives the optimal solution $x^* = (1.6, 0.2), X_{11} = 3.2, X_{22} = 0.2, X_{12} = 0$ with the optimal objective value -2.4. Clearly, this is a much better solution compared to what is obtained by our cut. Also this is very close to the optimal solution as well. But as pointed out, full level-1 RLT requires $\mathcal{O}(m^2)$ constraints and $\mathcal{O}(n^2)$ auxiliary variables. This increases the size of the problem considerably and further processing becomes more computationally expensive. While our cuts are computationally very cheap and do not increase the size of the problem to prohibitively.

As mentioned in Section 3.2, the idea of substituting basic variables by linear functions of nonbasic variables and only considering the products of nonbasic variables was also analyzed in [135]. We now highlight key differences between the ideas proposed here and [135].

- Algorithm 3.1 is a fast procedure to separate a basic solution from the linear relaxation of the problem. On the other hand, Sherali et al. [135] provide a “complete" first-order RLT relaxation of the problem that consists of many new variables and constraints, some of which may potentially be of little use in practically solving the instance. In fact, their procedure is independent of Q, and generates all RLT constraints using the current basis. On the other hand, our procedure derives cuts directly from the quadratic constraint.

- Algorithm 3.1 can be used repeatedly to obtain cuts that are guaranteed to cut off a given basic solution. Some of these cuts can be higher ordered RLT constraints, which are practically difficult to obtain using past approaches.

- If one substitutes only a subset of basic variables in the quadratic function while ensuring each bilinear term in the substituted quadratic has at least one nonbasic variable, then the cut obtained is different from the approach of Sherali et al. [135]. While this cut is in theory dominated by other RLTs, it is usually sparser and more effective, as our computational results in Section 3.6 indicate.

- The approach presented here is easier to integrate with branch-and-cut algorithms as it does not require creating any new variables, and can be used repeatedly. This aspect is demonstrated in Section 3.6 where it is applied to several benchmark instances from a standard library. In contrast, generating all first-order RLT constraints is severely limited

by computational resources and is useful for only small instances.

3.4 Analogy with Gomory's fractional cuts

Gomory's fractional cuts ([79]) were the first general purpose cutting planes developed for pure integer linear programs. We describe how our procedure discussed in Section 3.3 is similar to Gomory's fractional cuts. Let us consider the integer program

$$\min\{c^T x \mid Ax = b, x \geq 0, x \in \mathbb{Z}^n\}. \tag{IP}$$

We can rewrite the integer constraints equivalently as $x_i \leq \lfloor x_i \rfloor \ \forall i \in \{1,\ldots,n\}$. When we solve the natural linear programming relaxation of (IP) using simplex method ignoring the integer constraints then each row of the optimal simplex tableau will be of the form

$$x_i^B + \sum B_{ij}^{-1} x_j^N = x_i^*$$

where x^B, x^N are the set of basic and non-basic variables respectively, B is the optimal basis matrix as described in the previous section, and $x^B = x^*, x^N = 0$ is the optimal solution to the linear relaxation of (IP). As described in Section 3.3 we will substitute the equations obtained in the optimal simplex tableau into the nonconvex constraint if that is not satisfied. Here the only nonconvex constraints are $x_i \leq \lfloor x_i \rfloor$ representing the integrality of the variables. Thus, if for some $i, x^* \notin \mathbb{Z}$, then we can substitute the corresponding simplex row to obtain the following inequality

$$x_i^* - \sum B_{ij}^{-1} x_j^N \leq \lfloor x_i^* - \sum B_{ij}^{-1} x_j^N \rfloor \tag{3.10}$$

Similar to the McCormick relaxation of each bilinear or square term in the quadratic case discussed above we will relax each term in the floor function in (3.10). Since floor function is monotonically nondecreasing, overestimating each term in the floor function will overestimate

the function value as well. Therefore,

$$\begin{aligned} & B_{ij}^{-1} \geq \lfloor B_{ij}^{-1} \rfloor \\ \implies & x_i^* - \sum B_{ij}^{-1} x_j^N \leq x_i^* - \sum \lfloor B_{ij}^{-1} \rfloor x_j^N \\ \implies & \lfloor x_i^* - \sum B_{ij}^{-1} x_j^N \rfloor \leq \lfloor x_i^* - \sum \lfloor B_{ij}^{-1} \rfloor x_j^N \rfloor \\ \implies & \lfloor x_i^* - \sum B_{ij}^{-1} x_j^N \rfloor \leq \lfloor x_i^* \rfloor - \sum \lfloor B_{ij}^{-1} \rfloor x_j^N \end{aligned} \tag{3.11}$$

where the first inequality follows by the definition of the floor function, second inequality follows because $x_j^N \geq 0$, third inequality follows because floor function is monotonically nondecreasing and the last inequality follows because $x_j^N \in \mathbb{Z}$.

Combining (3.10) and (3.11) and rearranging the terms we get

$$\sum (B_{ij}^{-1} - \lfloor B_{ij}^{-1} \rfloor) x_j^N \geq x_i^* - \lfloor x_i^* \rfloor$$

which is Gomory's fractional cut. This shows that if one applies the above procedure discussed in Section 3.3 to the inequality $x \leq \lfloor x \rfloor$ for any integer variable x then we get the Gomory's fractional cut.

3.5 Adding new variables and connections with RLT

In the previous section, linear estimators were obtained for each term of the substituted quadratic. These estimators were then summed together to obtain a valid inequality. Another way to linearize the substituted quadratic is by adding auxiliary variables for each quadratic term and then using McCormick relaxation as described in Section 3.1. This procedure obviously creates several new variables that must be added to the LP relaxation. On the other hand, this form is equivalent to adding all the cuts possible by after $g(x)$ is obtained by selecting different combinations of under- or over-estimators in Algorithm 3.1, and hence may tighten the relaxation more. We illustrate this using an example.

Example 3.11. Consider the substituted quadratic inequality (3.2) obtained in Example 3.1. We add auxiliary variables $w_{23} = x_2x_3, w_{24} = x_2x_4, w_{25} = x_2x_5$ and add McCormick relaxation for

these added variables to obtain the following relaxation

$$
\begin{aligned}
x_1 + 2x_3 - 3x_4 + 2x_5 &= 0.3 \\
x_2 + x_6 &= 0.5 \\
0.3x_2 - x_3 - 2w_{23} + 3w_{24} - 2w_{25} &\leq 0 \qquad (3.12) \\
x_2 + x_3 - w_{23} &\leq 1 \\
w_{23} - x_2 &\leq 0 \\
w_{23} - x_3 &\leq 0 \qquad (3.13) \\
x_2 + x_4 - w_{24} &\leq 1 \\
w_{24} - x_2 &\leq 0 \\
w_{24} - x_4 &\leq 0 \\
x_2 + x_5 - w_{25} &\leq 1 \\
w_{25} - x_2 &\leq 0 \\
w_{25} - x_5 &\leq 0 \qquad (3.14) \\
x_i &\in [0,1],\ i = 1,\ldots,6 \\
w_{23}, w_{24}, w_{25} &\geq 0
\end{aligned}
$$

Taking the linear combination $(3.12) + 2 \times ((3.13) + (3.14))$ and using the fact that $w_{24} \geq 0$ we get the cut $0.3x_2 - 3x_3 - 2x_5 \leq 0$ obtained in Example 3.1. Consider the point $\hat{x} = (1, 0.5, 0.15, 1, 1, 0)$ which satisfies both the equality constraints and the cut but there does not exist any $\hat{w}$ such that $(\hat{x}, \hat{w})$ is feasible to the above relaxation. This shows that the above relaxation is tighter than simply adding the cut.

It is clear from the above example that our procedure adds cuts which are equivalent to some of the cuts obtained by Reformulation Linearization Technique (RLT) [133]. However, it should be noted that the substituted quadratic inequality obtained is dense in the nonbasic variables and thus several auxiliary w_{ij} variables will be required. This increases the size of the LP significantly and is practically not suitable for a solver. Thus one must judiciously choose quadratic terms that should be replaced by an auxiliary variable and other quadratic terms can be relaxed using McCormick inequalities described in the previous sections.

We now describe this procedure of adding new variables in the general form for completeness. Inequality (3.9) in a slightly different form is

$$f(x^*) - 2x^{*T}QB^{-1}s^B - a^TB^{-1}s^B + \langle \widetilde{Q}, s^B s^{BT} \rangle \leq d.$$

We add a matrix variable $W = (s^B)(s^B)^T$ to obtain the linear inequality

$$f(x^*) - 2x^{*T}QB^{-1}s^B - a^TB^{-1}s^B + \langle \widetilde{Q}, W \rangle \leq d. \tag{3.15}$$

The constraint $W = (s^B)(s^B)^T$ can then be relaxed using the standard McCormick relaxation

$$W \geq 0, \tag{3.16a}$$
$$W \geq \overline{s^B} s^{BT} + s^B \overline{s^B}^T - \overline{s^B s^B}^T, \tag{3.16b}$$
$$W \leq \overline{s^B} s^{BT}, \tag{3.16c}$$

where all the matrix inequalities above are element wise inequalities. It is evident that any point that is infeasible to any of the cuts that can be derived from (3.9) will also be infeasible to the inequalities (3.15) and (3.16) since every cut in (3.9) is a McCormick underestimator of $(s^B)(s^B)^T$ and here we add the McCormick relaxation of $(s^B)(s^B)^T$. The inequalities in (3.16), after doing correct substitutions for the auxiliary variables W, will give us some nonnegative linear combinations of the RLT inequalities obtained by taking product of the active constraints and bound constraints. Inequality (3.15) can also be derived from RLT when the auxiliary RLT variables are substituted in the original quadratic constraint in the problem. It becomes apparent that the cuts derived from (3.9) can also be obtained from the RLT inequalities by projecting them to the x space. Thus, our procedure in essence gives a method to identify which RLT cuts can be added into the problem without generating all of them (many of which may be redundant). It should also be noted that repeated addition of our cuts iteratively essentially adds higher order RLT cuts without adding auxiliary variables.

Proposition 3.12. The inequalities (3.16) are nonnegative linear combination of RLT inequalities for the problem (QCP1).

Proof. Recall that $s^B = b^B - Bx$ and $\overline{s^B} = b^B - B_+ l - B_- u$. For (3.16a), we can easily verify

that $W \geq 0$ is equivalent to the RLT applied to the product of $(b^B - Bx)(b^B - Bx)^T \geq 0$.

For (3.16b), it is a linear relaxation of $(\overline{s^B} - s^B)(\overline{s^B} - s^B)^T \geq 0$. Now,

$$\overline{s^B} - s^B = Bx - B_+\underline{x} - B_-\overline{x} = B_+x + B_-x - B_+\underline{x} - B_-\overline{x} = B_+(x - \underline{x}) - B_-(\overline{x} - x),$$

and thus

$$\begin{aligned}
(\overline{s^B} - s^B)(\overline{s^B} - s^B)^T &= (B_+(x - \underline{x}) - B_-(\overline{x} - x))(B_+(x - \underline{x}) - B_-(\overline{x} - x))^T \\
&= B_+(x - \underline{x})(x - \underline{x})^T B_+^T - B_-(\overline{x} - x)(x - \underline{x})^T B_+^T \\
&\quad - B_+(x - \underline{x})(\overline{x} - x)^T B_-^T + B_-(\overline{x} - x)(\overline{x} - x)^T B_-^T \\
&= \langle (x - \underline{x})(x - \underline{x})^T, B_+B_+^T \rangle + \langle (\overline{x} - x)(x - \underline{x})^T, -B_-B_+^T \rangle \\
&\quad + \langle (x - \underline{x})(\overline{x} - x)^T, -B_+B_-^T \rangle + \langle (\overline{x} - x)(\overline{x} - x)^T, B_-B_-^T \rangle \geq 0,
\end{aligned}$$

each term in the above inequality is a nonnegative linear combination of an RLT inequality, and so is their overall sum.

For (3.16c), it is a linearization of $s^B(\overline{s^B} - s^B)^T = (b^B - Bx)(B_+(x - \underline{x}) - B_-(\overline{x} - x))^T = B_+(x - \underline{x})(b^B - Bx)^T - B_-(\overline{x} - x)(b^B - Bx)^T \geq 0$ which again is a nonnegative linear combination of RLT inequalities. □

3.6 Computational results

We describe two sets of experiments to assess the computational impact of adding the cuts described in the previous sections. In the first set of experiments (Section 3.6.1) cuts are added as described in Algorithm 3.1, i.e., without introducing new variables in the cutting stage. Six variants of this procedure are tested. In the second set of experiments (Section 3.6.2), new variables are introduced in each round of cutting, as described in Section 3.5. The second approach results in a much tighter relaxation after adding cuts, but comes with the additional cost of adding more variables each time a cut is added. This experiment is proposed to quantify the effect of deriving one or two inequalities from a quadratic constraint (Section 3.6.1) relative

to adding all possible ones from Algorithm 3.1.

We implemented the procedures in `mglob` solver. All computational experiments have been performed on a computer with a 64-bit Intel(R) Xeon(R) E5-2670 v2, 2.50GHz CPU, and 128 GB RAM. The programs were run on a single core of the CPU. The code was compiled using GCC-4.9.2 compiler. CLP-1.17.6 [67] was used as an LP solver.

We selected 216 QP, QCP, and QCQP instances from MINLPLib [43] for the experiments that have an optimal solution available, i.e. there is a gap of less than 10^{-6} between the primal and dual bound in MINLPLib dataset. Limiting our experiments to these "easy" instances enables us to check whether the cuts erroneously cut the optimal point and also to precisely compute the gap closed. We did not consider instances with integer variables as we wanted to focus on our procedure in isolation from other tightening and cut generation procedures. We further excluded 51 instances for which either `mglob` solved in the root node without any cuts or the gap between root node relaxation and the optimal objective value was less than 10^{-6}. One instance for which root node relaxation processing by `mglob` took more than 30 minutes was also excluded. After this filtering 164 instances remained for the computational experiments described here. We have chosen both convex and nonconvex problems which contain either quadratic objective or one or more quadratic constraints. Routines to automatically identify and exploit convex nonlinear constraints in `mglob` were disabled for these experiments. Thus all these instances were treated as nonconvex. We call this test set $\mathscr{T}_1$.

We consider another set, $\mathscr{T}_2$ of pooling problems [115] from the MINLPLib. Pooling problem is a problem in petrochemical industries. All quadratic terms in these instances are bilinear, and may be suited for the RLT cuts like we propose. From a total of 88 pooling instances in MINLPLib dataset, three are not quadratic problems and were removed. One more instance was removed because default `mglob` solves the problem in the root node (without any cuts). Eight instances were removed because default `mglob` takes more than 30 minutes to process the root node. We select the remaining 76 instance. Unlike the set $\mathscr{T}_1$, we do not know the optimal solution value of some of the instances in $\mathscr{T}_2$. Some of these instances have integer or binary variables.

3.6.1 Cuts in original space of variables

We now describe the computational impact of adding cuts as described in Section 3.3. The input QCO problem is first transformed by substituting each quadratic term, $x_i x_j$, that appears in the problem (including the objective function), with an auxiliary variable y_{ij} and then adding the constraint $y_{ij} = x_i x_j$. Bound propagation techniques [24, 128, 60] are then applied to obtain bounds on each variable. An initial LP relaxation of the transformed problem is then obtained using McCormick inequalities for each bilinear term $y_{ij} = x_i x_j$. We then solve the relaxation to obtain a lower bound that we call z_{before}. Cuts are then added using the proposed variants of Algorithm 3.1 as described below. The lower bound obtained after adding cutting planes and solving the tightened relaxation is called z_{after}. We compute the gap closed by the cuts using the formula

$$\text{Gap closed} = \frac{(z_{\text{after}} - z_{\text{before}}) \times 100}{z^* - z_{\text{before}}}, \tag{3.17}$$

where z^* is the best known optimal objective value available from MINLPLib. Note $z^* \geq z_{\text{after}} \geq z_{\text{before}}$.

We have conducted two types of experiments here each having three sub-variants. For each quadratic constraint $y_{ij} = x_i x_j$ when both x_i, x_j are basic variables and $y^*_{ij} \neq x^*_i x^*_j$ (in the initial LP solution), then two possible ways of substituting this quadratic constraint are possible.

1. Substitute both x_i and x_j with their corresponding simplex rows to obtain a quadratic function in only nonbasic variables and then under- or overestimate the new terms to obtain the cuts. We propose three different variants of obtaining the linear estimator of the quadratic function for this case.

   (a) Minimum coefficient sum - Suppose there is a term $x_k x_l$ (after the above substitution) that needs to be overestimated and both x_k, x_l are at their lower bounds $\underline{x_k}, \underline{x_l}$. Then two overestimators $\overline{x_l} x_k + \underline{x_k} x_l - \underline{x_k}\,\overline{x_l}$, and $\underline{x_l} x_k + \overline{x_k} x_l - \overline{x_k}\,\underline{x_l}$ are available. If $|\underline{x_k}| + |\overline{x_l}| < |\overline{x_k}| + |\underline{x_l}|$ then we choose the first overestimator, and the second one otherwise. Similar rules are used for other cases. The motivation behind using the minimum coefficient sum rule is that we prefer smaller coefficients in the cut.

(b) Equal weight - In this variant, if we have two under- or overestimators for a quadratic term then we take a convex combination of the two estimators with $\lambda = 0.5$.

(c) Reduced cost weight - Instead of giving equal weights to the two estimators as in (b), reduced costs are used to decide a different convex combination. Let us consider the term with x_k, x_l and the corresponding reduced costs μ_k, μ_l. If $|\mu_k| + |\mu_l| < \varepsilon$ then we select the weights for each estimator as 0.5, otherwise we normalise the reduced costs so that $d_k = \frac{|\mu_k|}{|\mu_k|+|\mu_l|}, d_l = \frac{|\mu_l|}{|\mu_k|+|\mu_l|}$. The underestimators and overestimators then are given in Table 3.2. We set $\varepsilon = 10^{-6}$ in our experiments.

State of variable x_k	State of variable x_l	Underestimators to choose	Overestimators to choose
At lower bound ($\underline{x_k}$)	At lower bound ($\underline{x_l}$)	$\underline{x_l}x_k + \underline{x_k}x_l - \underline{x_k}\underline{x_l}$	$d_k(\overline{x_l}x_k + \underline{x_k}x_l - \underline{x_k}\overline{x_l}) +$ $d_l(\underline{x_l}x_k + \overline{x_k}x_l - \overline{x_k}\underline{x_l})$
At lower bound ($\underline{x_k}$)	At upper bound ($\overline{x_l}$)	$d_k(\underline{x_l}x_k + \underline{x_k}x_l - \underline{x_k}\underline{x_l}) +$ $d_l(\overline{x_l}x_k + \overline{x_k}x_l - \overline{x_k}\overline{x_l})$	$\overline{x_l}x_k + \underline{x_k}x_l - \underline{x_k}\overline{x_l}$
At upper bound ($\overline{x_k}$)	At lower bound ($\underline{x_l}$)	$d_l(\underline{x_l}x_k + \underline{x_k}x_l - \underline{x_k}\underline{x_l}) +$ $d_k(\overline{x_l}x_k + \overline{x_k}x_l - \overline{x_k}\overline{x_l})$	$\underline{x_l}x_k + \overline{x_k}x_l - \underline{x_l}\overline{x_k}$
At upper bound ($\overline{x_k}$)	At upper bound ($\overline{x_l}$)	$\overline{x_l}x_k + \overline{x_k}x_l - \overline{x_k}\overline{x_l}$	$d_l(\overline{x_l}x_k + \underline{x_k}x_l - \underline{x_k}\overline{x_l}) +$ $d_k(\underline{x_l}x_k + \overline{x_k}x_l - \overline{x_k}\underline{x_l})$

Table 3.2: Underestimators/overestimators based on the weights from the reduce cost of the variables

2. In the second set of variants, only one variable is substituted. For a constraint $y_{ij} = x_i x_j$ with $y^*_{ij} \neq x^*_i x^*_j$, we substitute only one out of x_i or x_j with its corresponding simplex row to obtain a new quadratic function. Again, we propose three different ways of choosing which variable to substitute.

   (a) Least infeasible - Among the two variables x_i, x_j we substitute the variable that appears in the fewer number of quadratic constraints that are violated by the current basic solution. If there is a tie we use the one which has fewer nonzero terms in its simplex row.

Substitute both variables	**Minimum Coefficient**	**Equal Weight**	**Reduced cost weight**
Average gap closed	12.75	10.45	13.31
Substitute one variable	**Least infeasible**	**Most sparse**	**One-by-one**
Average gap closed	31.06	30.86	35.53

Table 3.3: Average gap closed after adding the cuts on set $\mathscr{T}_1$.

(b) Most sparse - Among the two variables x_i, x_j we substitute the variable which has fewer nonzero terms in its simplex row. If there is a tie we use the one that appears in the fewer number of quadratic constraints violated by the current basic solution.

(c) One-by-one - We substitute both variables one-by-one to obtain two quadratic functions. For example, if the term x_1x_2 needs to substituted and if $x_1 + \sum_{j\in N} \alpha_{1j}x_j = \beta_1$ and $x_2 + \sum_{j\in N} \alpha_{2j}x_j = \beta_2$ are the corresponding simplex rows, then we first substitute x_1 to obtain the quadratic $\beta_1 x_2 - \sum_{j\in N} \alpha_{1j}x_j x_2$ and then we substitute x_2 to obtain another quadratic $\beta_2 x_1 - \sum_{j\in N} \alpha_{2j}x_j x_1$. Thus we get two quadratic functions and two cuts for each quadratic constraint.

We do only a single round of cut generation in these experiments. For all the six variants discussed, we only add a cut to the relaxation if the current LP solution x^* violates it by at least 10^{-3}. In a more practical setting, one would apply these cuts repeatedly and manage them in a more sophisticated manner (see [14, 145, 141] for example). We leave this aspect of tighter integration with other components of the solver to a future study. We also limit ourselves to only measuring the gap closed by these cuts, and not focus on their overall effectiveness in solving the problems as this would also require a lot of fine tuning and integration with the solver. While performing the experiments some instances on some of the variants faced numerical issues, and for such cases we report zero gap closed. The average gap closed per instance in $\mathscr{T}_1$ is tabulated in Table 3.3. On an average we close about 13% of the gap on the instances tested when both the basic variables are substituted while more than 30% of gap was closed when only one basic variable is substituted.

We plot a profile in Figure 3.2, to visualise the distribution of the performance of each of the six variants over 164 instances. The horizontal axis in the plot shows the gap closed (3.17) and the vertical axis counts the number of instances. A point (x, y) on the plot shows that at least $x\%$ gap was closed on y instances. It is clear from the profiles that substituting

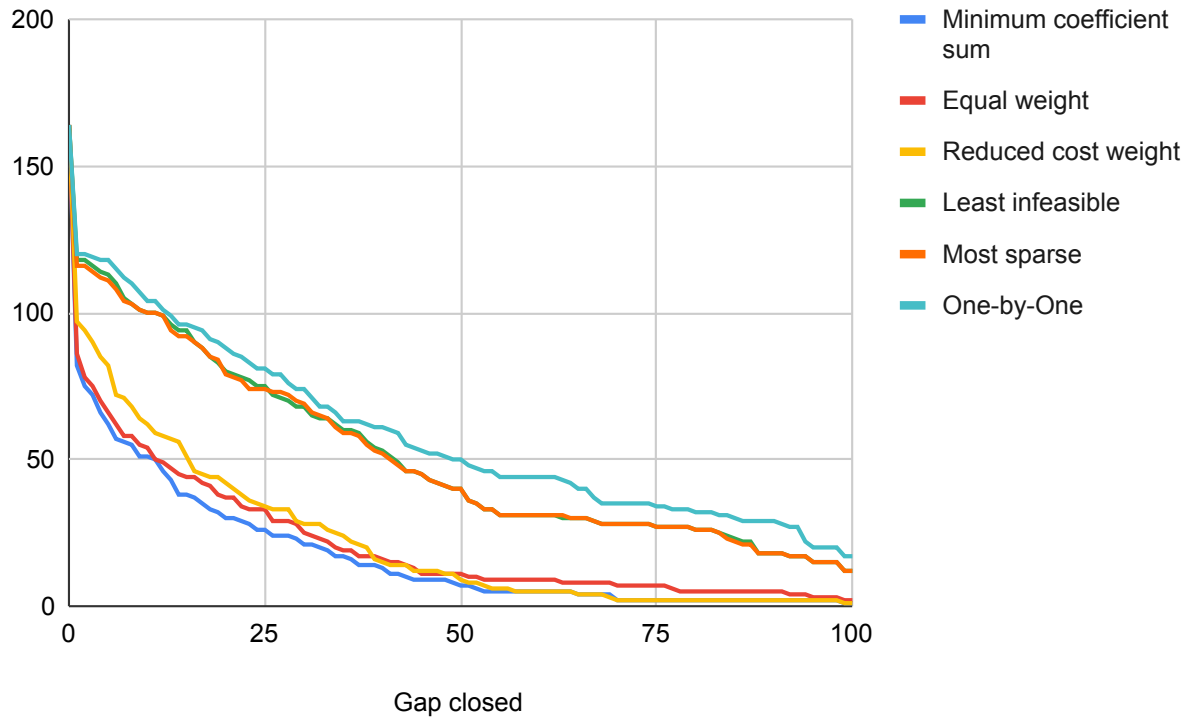


Figure 3.2: Profile of gap closed by one round of cuts on $\mathscr{T}_1$.

only one variable is superior to substituting both variables. The choice of sub-variants did not seem to have much influence on the gap closed. The detailed summary of the results for the instances in $\mathscr{T}_1$ for the six variants described here is reported in the file `DatasetT1.csv` in the supplementary material attached with this paper. We observe that the time taken in cutting is reasonably low for all instances, and that our procedure is computationally cheap. Also, time taken when substituting one variable is lower than that when we substitute both variables. It is unsurprising because the number of terms in the new quadratic increases significantly if both variables are substituted.

We also compare the objective lower bound obtained from our cutting plane procedure to that obtained by two different settings of the SCIP 8.0.2. [32]. SCIP is one of the leading open-source solvers for integer linear and nonlinear optimisation. The first setting keeps the default values of all parameters of SCIP. In the second setting, primal heuristics are turned off in order to isolate the effects of lower bound improvements from cuts and bound tightening routines alone. To switch off all primal heuristics in SCIP we use the option `set/heuristics/emphasis/off`. We call the lower bound provided by SCIP after

processing the root node as z_{SCIP}. Our algorithm is then compared to z_{SCIP} using the formula

$$\text{Percent change} = \frac{(z_{\text{SCIP}} - z_{\text{after}}) \times 100}{|z_{\text{before}}|}, \tag{3.18}$$

where z_{before} and z_{after} are obtained from `mglob` as described above. When $|\text{Percent change}| \leq 1$ we say our procedure and SCIP perform similarly. On the other hand if $(\text{Percent change}) < -1$ then we say our procedure performs better than SCIP and if $(\text{Percent change}) > 1$ then we say SCIP performs better than our procedure. SCIP root node processing of some instances in $\mathscr{T}_1$, and $\mathscr{T}_2$ were not completed even after 9600 seconds, and for these instances we use the lower bound provided within this time limit. We summarise the results in Table 3.4 for test set $\mathscr{T}_1$ and Table 3.5 for $\mathscr{T}_2$. In both these experiments the variant One-by-One was used as it was the most promising in our previous analysis. We observe that, on an average, the lower bounds from one round of our procedure are inferior to those from default SCIP on $\mathscr{T}_1$ and comparable to those from default SCIP on $\mathscr{T}_2$. The lower bounds from our procedure are seen to be superior to those from SCIP when primal heuristics of SCIP are turned off. It indicates that the proposed procedure may be quite helpful in improving the bounds, and needs good integration with other components of the solver. Detailed summary of the results have been reported in the supplementary material attached.

SCIP setting	**Number of instances**	**Number of instances where both SCIP and Algorithm 3.1 perform similarly**	**Number of instances where Algorithm 3.1 performs better**	**Number of instances where SCIP performs better**
Default	164	62	27	75
No heuristics	164	50	62	52

Table 3.4: Comparison of SCIP and Algorithm 3.1 for $\mathscr{T}_1$ instances

SCIP setting	Number of instances	Number of instances where both SCIP and Algorithm 3.1 perform similarly	Number of instances where Algorithm 3.1 performs better	Number of instances where SCIP performs better
Default	76	46	13	17
No heuristics	76	43	22	11

Table 3.5: Comparison of SCIP and Algorithm 3.1 for $\mathscr{T}_2$ instances

3.6.2 Adding variables

Now we describe the computational impact when we add auxiliary variables as described in Section 3.5 to obtain tighter relaxation. We first obtain an initial LP relaxation as explained in Section 3.6.1. For each quadratic constraint $y_{ij} = x_i x_j$ when both x_i, x_j are basic variables and $y^*_{ij} \neq x^*_i x^*_j$ (in the initial LP solution), we substitute variables with their corresponding simplex row using the following two strategies.

1. Substitute both variables - We substitute both the basic variables x_i and x_j to obtain a new quadratic function. For each term $x_k x_l$ in this new quadratic function, if an auxiliary variable for $x_k x_l$ is already present in the relaxation we substitute the term $x_k x_l$ with that variable. Otherwise we introduce a new variable $w_{kl} = x_k x_l$ and relax it using McCormick relaxations.

2. Substitute one variable - We substitute one variable at a time to obtain two new quadratic functions as described in variant One-by-one in Section 3.6.1. A new variable w_{kl} is then introduced for each term in the two quadratic constraints as described above.

In both these variants it is sometimes observed that the bounds on w_{kl} introduced can be quite large. If that is the case, the McCormick relaxation can have large coefficients and can cause numerical issues with the LP solvers. If $\max\{|\underline{w_{kl}}|, |\overline{w_{kl}}|\} \geq 10^6$ we do not add a new variable, but rather just add a linear term as described in Section 3.6.1. There are 53 such instances when both variables were substituted and 42 such instances when one variable was

	Substitute both variables	**Substitute one variable**
Average Gap Closed	25.50	39.89
Added variables w.r.t. original number of variables	3.87	2.84

Table 3.6: Average gap closed and relative size of the problem after adding auxiliary variables on set $\mathscr{T}_1$

substituted where such large coefficients were observed and thus not all variables have been added for such instances.

We test the above two variants on the test set $\mathscr{T}_1$ but four instances we removed because these instance hit time limit of 9600 seconds (while the new relaxation was still not generated). Also similar to the case in Section 3.6.1 there were instances facing numerical issues whose gap closed has been reported as 0%. Average gap closed for the two variants is reported in Table 3.6. We also measure the relative size of the new relaxation in terms of the number of variables in the initial relaxation i.e. the ratio of the number of variables in the new relaxation to the number of variables in the initial relaxation. The second row in Table 3.6 shows the average relative size of the new relaxation. We observe that substituting both variables and adding auxiliary variables closes 25% of gap on an average in the instances tested while the size of the relaxation increases to more than three times on average. On the other hand substituting one variable at a time and adding auxiliary variables for both quadratic functions closed about 39% of the gap while adding slightly fewer variables. Figure 3.3 shows the distribution of the performance of both the variants based on the gap closed. The experiment again demonstrates that substituting only one basic variable at a time is more beneficial. Substituting both variables increases the number of terms in the new quadratic whose termwise relaxation can be relatively weak.

We do not compare our results when auxiliary variables are added against SCIP because of the following reasons:

1. Adding auxiliary variables was computationally tested to create a benchmark for the cuts

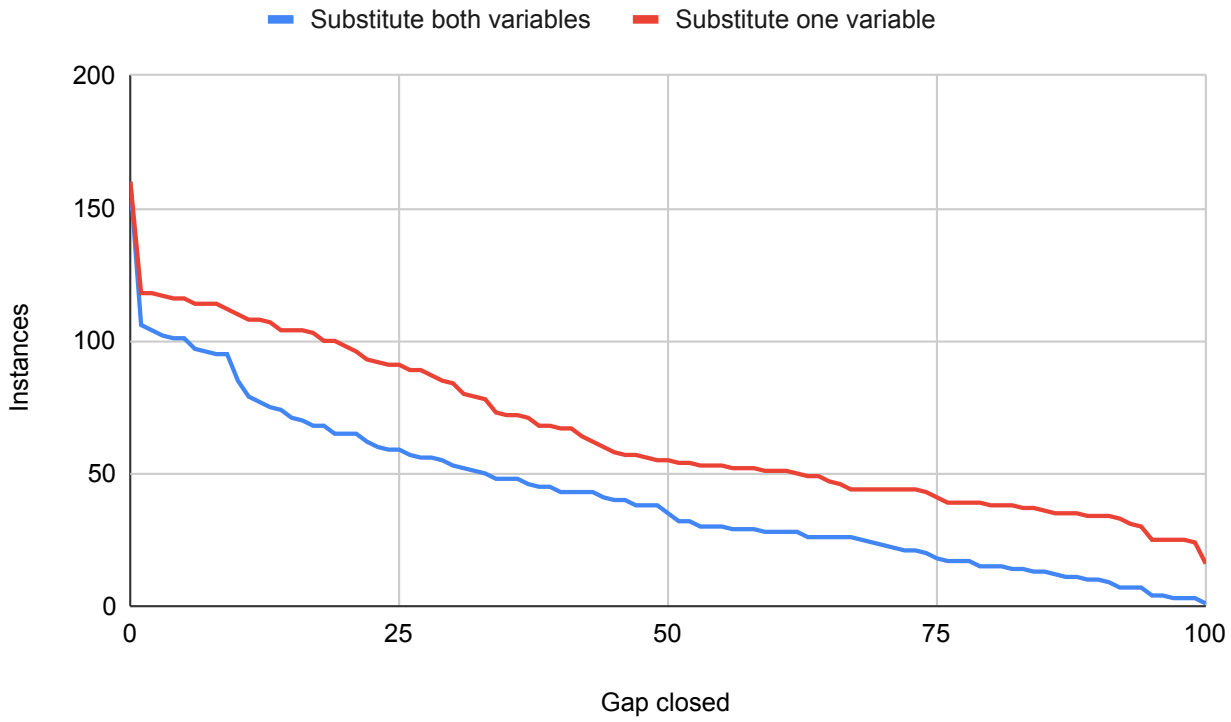


Figure 3.3: Profile of gap closed by adding auxiliary variables.

generated by Algorithm 3.1.

2. For a general purpose solver it is very difficult to add variables after root node processing has started. We only add additional variables while presolving or creating the relaxations if needed. Once the problem is loaded in the LP solver if additional variables are added (along with additional constraints) the LP problem doesn't remain either primal feasible or dual feasible and thus simplex method has to restart effectively. This makes the solve time quite higher.

3. It is difficult to manage additional variables added after the problem is loaded in the LP solver. Thus we avoid doing this within the solver.

The general scheme of introducing new variables while generating cuts is not recommended in a practical setting. Most branch-and-cut implementations do not allow adding new variables after the presolving stage. These experiments however are useful for understanding the effectiveness of the cuts described in Section 3.6.1. By adding variables, we are introducing all possible cuts that can be generated by Algorithm 3.1. These experimental

results suggest that the heuristic strategy of One-by-one generates sufficiently good cuts and closes a sizeable gap as compared to what is possible by adding all cuts.

3.7 Conclusion and Future Work

We have presented a procedure for deriving cutting planes for a linear relaxation of QCP. Our procedure is guaranteed to cut off LP basic feasible solution that is not feasible to the QCP. Our tests of applying one round of cuts yield promising results. Even though these cuts are a particular type of RLT inequalities, they are available readily and do not require any search or guess-work. Successful integration with a general purpose solver would require multiple rounds of cut generation, careful selection, and management of these cuts along with careful tuning of parameters.

There are several open questions with regards to this procedure. First, the convergence of this procedure on general and specific classes of QCP can be analysed. Second, several cuts are possible with different choices available in the algorithm and from different basic solutions. Practical strategies for finding computational effective cuts would be an interesting topic, as would integrating them fully in an MIQCP solver.

Chapter 4

Branching Strategies

Presolving, primal heuristics, and cutting planes are typically used extensively in the root node of a branch-and-cut algorithm to close the gap between the upper and lower bounds of the optimal value. These techniques significantly reduce the size of the branch-and-bound tree and the overall effort required to solve the problem. Once these approaches are exhausted, we 'branch' the search space to create subproblems that are then individually relaxed and solved further. This recursive procedure is described already in Sections 1.3.1 and 1.6.

The subproblems created are called the child nodes of the problem (parent node). In this sense, branching creates a tree of problems where the root is the original relaxation, and each node subsequently is a subproblem of its parent node. The subproblems created must have the following two properties:

- The relaxation solution of the parent node must be infeasible to all the children nodes.
- Every feasible solution to the parent node must be feasible to at least one child node.

In nonconvex MIQCQP, two types of nonconvexities are present: integrality and nonconvex nonlinear functions. For integrality, we use integer branching as described in Section 1.3.1, and for nonconvex nonlinear functions, we use spatial branching as described in

1.6. Typically, at every node of the sB&B tree, there are several variables for which branching can be done, and one must choose which branching variable to select for the current node. This choice of branching 'candidate' is essential since it significantly affects the number of nodes to be processed. Thus, given a set of branching candidates, we need a selection criterion to choose a candidate to branch. A *branching strategy* is the selection criterion used to choose a branching candidate.

Branching is an essential component of the branch-and-cut and branch-and-bound algorithms. Most of the practically arising MIQCQO problems require branching to solve them. Since branching increases the number of subproblems to be solved, a good branching strategy used recursively can dramatically improve the solution time. Despite its central role in solving these problems, our understanding and analysis of branching strategies is quite limited, especially compared to reformulation and cutting plane techniques.

Minotaur has a few inbuilt functions for branching. We review these techniques along with others described in the literature and propose new strategies specially designed for spatial branching for MIQCQO. Empirical results are compiled to assess the impact of these strategies on benchmark instances.

4.1 What is a branching strategy?

Let us use an example to show the effect on the size of sB&B tree when selecting a branching candidate.

Example 4.1. Consider the MIQCQP

$$\begin{aligned} z = \min\ & x_1^2 - 2x_1x_2 \\ \text{s. t. } & x_1x_2 + x_1 \leq 2, \\ & x_1, x_2 \in [0,2], \\ & x_1 \in \mathbb{Z}, \\ & x_2 \in \mathbb{R}. \end{aligned}$$

Branching variable	Left child			Right child		
	$\hat{x},\hat{y}$	$\hat{z}$	Status	$\hat{x},\hat{y}$	$\hat{z}$	Status
x_1	$(0,0)$, $(0,0)$	0	Pruned by feasibility	$(1,1)$, $(1,1)$	-1	Pruned by feasibility
x_2	$(1,0.333)$, $(0,0.667)$	-1.333	Requires further branching	$(0.667,1.111)$, $(0,1.333)$	-2.667	Requires further branching

Table 4.1: Effect of selecting different branching variables

The optimal solution to the problem is $x^* = (1,1)$ with $z^* = -1$. We add auxiliary variables $y_1 = x_1^2, y_2 = x_1x_2$ and relax the problem using standard McCormick relaxation as described in Section 1.5 and relax the integrality of x_1 to obtain a linear relaxation of the problem. Solving the relaxation we get the LP solution $\hat{x} = (0.667, 0.667), \hat{y} = (0, 1.333)$ with $\hat{z} = -2.667$. Note that this solution is not feasible to both the nonlinear constraints $y_1 = x_1^2, y_2 = x_1x_2$ and x_1 is fractional.

Now, we can either branch on $(x_1 \leq 0, x_1 \geq 1)$ or we can branch on $(x_2 \leq 0.667, x_2 \geq 0.667)$. We summarise the effect of branching on either of these variables in Table 4.1. In Table 4.1, the Left and Right child correspond to the branch when the upper and the lower bound of the variable are changed, respectively. Clearly, branching on x_1 solves the problem since both child nodes are pruned by feasibility while branching on x_2, we still need to branch both the child nodes further.

Example 4.1 shows that when several variables are possible to branch, deciding which variable to branch on significantly affects the size of the sB&B tree. Therefore, we need to identify which variable to branch to obtain the smallest sB&B tree. It is not possible to predict the size of the sB&B tree for a given candidate without solving the problem. Thus, in principle, a branching strategy provides an approximate score that represents the 'benefit' of selecting a branching candidate. A branching strategy uses some criteria to provide an estimate of the score. Following are some of the criteria used to estimate the score for a branching candidate:

- The approximate increase in the lower bound after branching.
- Amount of feasible region eliminated by branching.

- Violation from the constraints.

There can be other criteria for selecting a branching candidate, like the number of infeasible constraints a variable appears in, ensuring that the two subproblems have similar sizes, etc. One can also use a combination of these to obtain a hybrid score. Since all these scoring methods are estimates, choosing a branching strategy is difficult, and thorough experimental testing is needed to select a good performing strategy.

4.2 Literature Review

Extensive research has been conducted on branching variable selection. Several papers describe new rules to estimate a score for candidates. A common scoring technique is to use the violation of the current relaxation solution from the new relaxation in the child nodes. A common method used for integer branching is Maximum Infeasible branching as described in [4]. In this branching scheme, we simply calculate the score $s_i = 0.5 - |\hat{x}_i - \lfloor \hat{x}_i \rfloor - 0.5|$ as the closeness of fractional part of x_i from 0.5. An analogous infeasibility measure for nonconvex constraints is the *rb-inf* as described in [25]. Belotti et. al. [25] first describe an infeasibility measure as the scaled distance between the current variable value and the constraint activity at the point. That is consider a constraint of the form $x_i = v_i(x)$, then the infeasibility measure $U_i = \frac{|\hat{x}_i - v_i(\hat{x})|}{1+\|v_i(\hat{x})\|}$. The *rb-inf* measure for x_i is the sum of all such infeasibilities where x_i appears. Another related approach is the idea of 'violation transfer' described in [138]. In this scheme, violations of nonconvex constraints are first assigned to the problem variables, and then the violation is transferred between problem variables and auxiliary variables. The updated violations are then used to obtain a score for branching candidates.

Another approach to estimate a score for branching candidates is the idea of *pseudocosts* first described in [27]. This rule keeps a history of lower bound (on z^*) changes when a particular variable is selected for branching. We then take the average lower bound update as the score for branching. More details about pseudocost branching are available in [104].

One of the most computationally expensive branching strategies is *strong* branching [16]. In this branching scheme, we solve two LPs, one for the left child and one for the right child, for each branching candidate. We then compute the lower bound change and obtain a candidate

score. Recently, Dey et al. [58] showed that for small MILO instances, strong branching tree size is within a factor of two of the optimal tree size.

A common problem in pseudocost branching is the initialisation of pseudocost in the beginning of the B&B tree. Since in the root node (and initial stage of the tree search), selecting the correct branching candidate is much more crucial than later nodes, as pointed out by Forrest et al. [65]. Initialising pseudocost correctly is very important. To this end, pseudocost are generally initialised using strong branching in the initial nodes. This method of combining strong branching with pseudocost branching is called reliability branching which was introduced by Achterberg et al. [4]. Reliability branching is also used for nonconvex problems as described in [25, 144].

Recently, machine learning techniques have also been employed to select a good branching candidate, see [12, 21, 76]. These techniques try to approximate strong branching or estimate tree depth. See [106] for a survey of machine learning based techniques for branching variable selection.

4.3 Branching strategies for nonconvex problems

This section describes five branching strategies implemented in Minotaur [111]. We first present a simple rule similar to maximum infeasible branching in MILP, which computes violation of the nonconvex constraints to obtain a score for branching candidates. We then describe strong branching for nonconvex problems and describe some implementation details for the same. Next, we describe two new branching rules for nonconvex problems, namely, *bt-strong* branching and *bt-estimate* branching. Finally, we combine some of these strategies to obtain a reliability branching scheme for nonconvex problems, which we call bt-reliability branching. We then compare these strategies in terms of the number of nodes explored and the time taken to solve the problem.

For each $x_i x_j$ term in the problem, we add an auxiliary y_{ij} variable. We then transform the problem by adding the constraint $y_{ij} = x_i x_j$, which is then relaxed using McCormick [112] relaxation. Thus, the only nonconvexities in the problem, apart from the integrality restrictions, are the constraints of the form $y_{ij} = x_i x_j$. Now, if $\hat{y}_{ij} \neq \hat{x}_i \hat{x}_j$ then we add both variables x_i, and x_j

as branching candidates. Also, an integer variable x_i having fractional value at the current point is added as a branching candidate. Likewise, we obtain a set $\mathscr{B}$ of branching candidates. All branching strategies described next compute a score for each branching candidate.

4.3.1 Maximum Violation Branching

A simple measure to identify a good branching candidate is to use the violation of the current solution at the node from the relaxation of the child nodes. For integer variables, this is simply the distance of the fractional solution from the nearest integer. For nonconvex constraints, it is not as straightforward. Consider the constraint $y_{ij} = x_i x_j$ which is infeasible for the relaxation solution $(\hat{x}, \hat{y})$, i.e. $\hat{y}_{ij} \neq \hat{x}_i \hat{x}_j$. We add both variables x_i, x_j as branching candidates. The violation score for the constraint is computed as the orthogonal distance from the point $(\hat{x}_i, \hat{x}_j, \hat{y}_{ij})$ to the updated McCormick constraint that separates the relaxation solution in the branch. For instance, if $\hat{y}_{ij} < \hat{x}_i \hat{x}_j$ and we wish to down branch on $x_i \leq \hat{x}_i$ then the McCormick constraint that separates the point $(\hat{x}_i, \hat{x}_j, \hat{y}_{ij})$ from the relaxation is $y_{ij} \geq \hat{x}_i x_j + \overline{x}_j x_i - \hat{x}_i \overline{x}_j$. The orthogonal distance from $(\hat{x}_i, \hat{x}_j, \hat{y}_{ij})$ to the plane $y_{ij} = \hat{x}_i x_j + \overline{x}_j x_i - \hat{x}_i \overline{x}_j$ is $\frac{\hat{x}_i \hat{x}_j - \hat{y}_{ij}}{\sqrt{1+\hat{x}_i^2+\overline{x}_j^2}}$ which is taken as the violation score for the down branch. Similarly, the branching score for the up branch is computed, and an aggregated score for the candidate is then computed based on these violation scores.

Based on the above discussion, we use Algorithm 4.1 to compute a score for all the branching candidates. In Algorithm 4.1, we compute the down violation and up violation separately and then compute the score of the candidate as a convex combination of the two violations. The score factor $\mu = 0.8$ is used in taking the convex combination.

Notice that violations computed in Algorithm 4.1 are added for each nonconvex constraint to obtain a cumulative score for the candidate. This is desirable because it gives a higher score to a variable appearing in several infeasible nonconvex constraints so that branching on that variable affects a large portion of the feasible region.

Algorithm 4.1 Scoring for Maximum Violation Branching

Input: A set $\mathscr{B}$ of branching candidates

Output: Score s_i for each branching candidate $i \in \mathscr{B}$

Parameters: Score factor μ

procedure SCOREFORMAXVIOBRANCHING

for $i \in \mathscr{B}$ **do**

$d_i \leftarrow 0$

$u_i \leftarrow 0$

if $x_i \in \mathbb{Z}$ **then**

$d_i \leftarrow d_i + (\hat{x}_i - \lfloor \hat{x}_i \rfloor)$

$u_i \leftarrow u_i + (\lceil \hat{x}_i \rceil - \hat{x}_i)$

for $j \in \mathscr{B}$ **do**

if y_{ij} exists **then**

if $\hat{y}_{ij} < \hat{x}_i \hat{x}_j$ **then**

$d_i \leftarrow d_i + \frac{\hat{x}_i \hat{x}_j - \hat{y}_{ij}}{\sqrt{1+\hat{x}_i^2+\underline{x_j}^2}}$

$u_i \leftarrow u_i + \frac{\hat{x}_i \hat{x}_j - \hat{y}_{ij}}{\sqrt{1+\hat{x}_i^2+\overline{x_j}^2}}$

else

$d_i \leftarrow d_i + \frac{\hat{y}_{ij} - \hat{x}_i \hat{x}_j}{\sqrt{1+\hat{x}_i^2+\underline{x_j}^2}}$

$u_i \leftarrow u_i + \frac{\hat{y}_{ij} - \hat{x}_i \hat{x}_j}{\sqrt{1+\hat{x}_i^2+\overline{x_j}^2}}$

$s_i \leftarrow \mu \min\{d_i, u_i\} + (1-\mu)\max\{d_i, u_i\}$

4.3.2 Strong Branching

Strong branching is empirically one of the most effective branching strategies considering the number of nodes processed in the B&B tree [4]. The idea is to identify which branching candidate will improve the lower bound the most and then branch on that candidate. We first temporarily branch on each candidate one at a time and note the lower bound increase in either branching direction. We then score each candidate based on the weighted increase in the lower bound in both branching directions.

We describe the scoring method for strong branching for spatial branch-and-bound in

Algorithm 4.2. There are some features of strong branching worth highlighting here. Firstly, for spatial branching, bound changes to the candidate variable are not sufficient, as done in integer branching, and we need to update the McCormick relaxation for all the quadratic terms that have the candidate variable. Secondly, as opposed to maximum violation branching described in Section 4.3.1, strong branching can apply modifications to the current node without branching or can even decide to prune the current node.

4.3.3 Bt-strong Branching

We now describe a new branching technique unique to the spatial branch-and-bound algorithm. Empirically, strong branching explores fewer nodes than other branching strategies for branch-and-bound algorithm for MILP as discussed in Section 4.3.2. For spatial branch-and-bound, we observe that full strong branching, as described in Section 4.3.2, is also quite effective for most instances but may not be effective for some instances. Strong branching does not work well when the score for all branching candidates becomes 0. As described in Section 2.3, there is a nontrivial change in the McCormick relaxation when bounds are tightened. This motivates us to first update the bounds on all the variables based on the candidate bound change and then do strong branching on the candidate. Let us show this effect of doing bound tightening before strong branching using an example.

Example 4.2. Consider the instance (`st_pan1`) from MINLPLib [43]. We tighten the bounds using our bound tightening approaches to the upper bounds on the variables $x_1 \leq 1.11, x_2 \leq 0.93, x_3 \leq 1.10$. We relax this problem using McCormick relaxation described in Section 1.5.1 by adding auxiliary variables $y_1 = x_1^2, y_2 = x_2^2, y_3 = x_3^2$.

$$
\begin{aligned}
\min \quad & 1.25x_1 - 2.5x_1^2 - 5x_2^2 - 7.5x_3^2 + 5x_3 \\
\text{s.t.} \quad & 10x_1 + 0.2x_2 - 0.1x_3 \leq 11 \\
& -0.3x_1 + 9x_2 + 0.2x_3 \leq 18 \qquad \text{(st\_pan1)} \\
& -0.1x_1 + 0.4x_2 + 11x_3 \leq 12 \\
& 6x_1 + 8x_2 + 9x_3 \leq 18 \\
& x_1, x_2, x_3 \geq 0
\end{aligned}
$$

Algorithm 4.2 Scoring for Strong Branching

Input: A set $\mathscr{B}$ of branching candidates, Relaxation R at the current node, current objective value z^*

Output: Score s_i for each branching candidate $i \in \mathscr{B}$

Parameters: Score factor μ

procedure SCOREFORSTRONGBRANCHING

for $i \in \mathscr{B}$ **do**

if $x_i \in \mathbb{Z}$ **then**

$R_d \leftarrow R \cap \{(x,y) \mid x_i \leq \lfloor \hat{x}_i \rfloor\}$

$R_u \leftarrow R \cap \{(x,y) \mid x_i \geq \lceil \hat{x}_i \rceil\}$

else

$R_d \leftarrow R, R_u \leftarrow R$

for $j \in \mathscr{B}$ **do**

if y_{ij} exists **then**

$R_d \leftarrow R_d \cap \{(x,y) \mid x_i \leq \hat{x}_i, y_{ij} \geq \overline{x}_j + \hat{x}_i x_j - \overline{x}_j \hat{x}_i, y_{ij} \leq \underline{x}_j x_i + \hat{x}_i x_j - \underline{x}_j \hat{x}_i\}$

$R_u \leftarrow R_u \cap \{(x,y) \mid x_i \geq \hat{x}_i, y_{ij} \geq \underline{x}_j x_i + \hat{x}_i x_j - \underline{x}_j \hat{x}_i, y_{ij} \leq \overline{x}_j x_i + \hat{x}_i x_j - \overline{x}_j \hat{x}_i\}$

Solve R_d, R_u and let z_d, z_u be their objective values respectively.

if R_d is infeasible & R_u is infeasible **then**

Prune the current node

else if R_d is infeasible **then**

$R \leftarrow R \cap \{(x,y) \mid x_i \geq \hat{x}_i\}$

Reprocess the current node.

else if R_u is infeasible **then**

$R \leftarrow R \cap \{(x,y) \mid x_i \leq \hat{x}_i\}$

Reprocess the current node.

else

$d_i \leftarrow z_d - z^*$

$u_i \leftarrow z_u - z^*$

$s_i \leftarrow \mu \min\{d_i, u_i\} + (1-\mu)\max\{d_i, u_i\}$

The optimal solution to the relaxation is $\hat{x} = (1.1071, 0.1887, 1.0941), \hat{y} = (1.2301, 0.1748, 0.1989)$ with a lower bound on the objective value $\hat{z} = -5.61$ Clearly, $y_1 \neq x_1^2, y_2 \neq x_2^2$, and $y_3 \neq x_3^2$

and hence we decide to branch here. Now, let us branch on x_1 and focus on the left child i. e. $x_1 \leq 1.1071$. We update the McCormick relaxation the constraint $y_1 = x_1^2$ as is done in strong branching. The optimal solution of the relaxation remains -5.61. On the other hand, let us update the upper bounds on x_2, x_3 based on the new bound on x_1 and update the McCormick relaxation for all three variables. If we solve the new relaxation, the optimal solution increases to -5.5859.

Similarly, for all branching candidates in either direction, strong branching cannot estimate the lower bound change for the problem correctly, but tightening bounds before solving the strong branching problem gives us a better estimate of bound update for all candidates. If this problem is solved using strong branching, then seven nodes are required, but if bound tightening is done before strong branching, then only three nodes are required to solve the problem.

As motivated by Example 4.2, we thus do bound tightening for all variables before strong branching to obtain a better estimate of lower bound update. We call this type of branching as *bt-strong* branching. This branching strategy is even more expensive than strong branching because of the additional bound tightening step before strong branching. Nonetheless, we expect the size of branch-and-bound tree to be smaller than that of strong branching.

4.3.4 Bt-estimate Branching

As seen in the previous section, tightening variable bounds provides a better estimate of strong branching. Motivated by this idea, we define a new branching strategy for spatial branch-and-bound algorithm called *bt-estimate* branching. Since strong branching and bt-strong branching are computationally expensive branching strategies, we wish to estimate the increase in the lower bound without explicitly solving an LP. We do this by tightening bounds and estimating the change in objective using the reduced cost information. The algorithm for bt-estimate branching is given in Algorithm 4.3.

Algorithm 4.3 provides a fast heuristic to estimate the lower bound update for a given candidate without the need to solve the LP. Since we tighten the bounds, we can trivially check the feasibility of the node by checking if the new lower of a variable is greater than the new upper bound. Thus, similar to strong branching, we may either prune the node, provide

Algorithm 4.3 Scoring for bt-estimate Branching

Input: A set $\mathscr{B}$ of branching candidates, Relaxation R at the current node, reduced cost vector r.

Output: Score s_i for each branching candidate $i \in \mathscr{B}$

Parameters: Score factor μ

procedure SCOREFORBTESTIMATEBRANCHING

for $i \in \mathscr{B}$ **do**

$d_i \leftarrow 0$

$u_i \leftarrow 0$

$R_d \leftarrow R \cap \{(x,y) \mid x_i \leq \hat{x}_i\}$

$R_u \leftarrow R \cap \{(x,y) \mid x_i \geq \hat{x}_i\}$

Tighten bounds for R_d and R_u

Let $\underline{x}^d, \overline{x}^d$ be the new lower and upper bounds on the variables in R_d. Similarly we define $\underline{x}^u, \overline{x}^u$ for R_u

if R_d is infeasible & R_u is infeasible **then**

Prune the current node

else if R_d is infeasible **then**

$R \leftarrow R \cap \{(x,y) \mid x_i \geq \hat{x}_i\}$

Reprocess the current node.

else if R_u is infeasible **then**

$R \leftarrow R \cap \{(x,y) \mid x_i \leq \hat{x}_i\}$

Reprocess the current node.

else

for $j \in \{1, \ldots, n\}$ **do**

if $r_j \geq 0$ **then**

$d_i \leftarrow d_i + r_j(\underline{x}^d_j - \underline{x}_j)$

$u_i \leftarrow u_i + r_j(\underline{x}^u_j - \underline{x}_j)$

else

$d_i \leftarrow d_i + r_j(\overline{x}^d_j - \overline{x}_j)$

$u_i \leftarrow u_i + r_j(\overline{x}^u_j - \overline{x}_j)$

$s_i \leftarrow \mu \min\{d_i, u_i\} + (1-\mu)\max\{d_i, u_i\}$

modifications to reprocess the node or return branching scores. We estimate the lower bound update based only on the bound changes of the variables. Because of the bound changes, the McCormick relaxation changes as well, and this significantly changes the feasible region of the relaxation. One can consider the McCormick relaxation update by defining an appropriate measure for the change in the McCormick constraint and then using the dual values for the corresponding constraint. We have left this for future work since defining a measure for the update of McCormick constraint suitably may require significant testing before an appropriate measure can be derived.

4.3.5 Bt-reliability Branching

We now describe a branching strategy combining the strengths of bt-strong branching, pseudocost branching, and maximum violation branching that can be used for spatial branching. Our computational results showed that bt-strong branching performs the best in terms of the number of nodes processed, but the time taken for a single node processing is very high. This can become prohibitive for moderate to large instances; thus, bt-strong branching cannot be directly used. A simple approach like pseudocost branching does reasonably well in estimating the lower bound update, given a good initialization scheme. In MILO problems with only integer branching, a hybrid scheme involving strong branching initialization for pseudocost branching, known as reliability branching [4], has been shown to perform well in terms of both time taken and the number of nodes processed. For MIQCQO problems that require spatial branching, we propose a similar branching strategy that does pseudocost branching with bt-strong branching initialization. We use violation scores for unreliable candidates to prioritize which candidates to bt-strong branch. We use violation distances for reliable candidates as a distance measure for pseudocost branching. We describe our bt-reliability branching setup in Algorithm 4.4, scoring for reliable and unreliable candidates in Algorithms 4.5 and 4.6 respectively.

We first divide the set of branching candidates into two sets, namely, reliable candidates and unreliable candidates. Reliable candidates are those for which we have done bt-strong branching more than τ times. The remaining candidates are called the unreliable candidates. We have set $\tau = 5$ in Minotaur. For reliable candidates, we estimate the lower bound increase

Algorithm 4.4 Algorithm for bt-reliability Branching

Input: A set $\mathscr{B}$ of branching candidates, vectors indicating number of times up or down branching is done t^u, t^d

Output: Candidate $i \in \mathscr{B}$ that should be branched

Parameters: Threshold τ

procedure RELIABILITYBRANCHING

$\mathscr{B}_r \leftarrow \phi$

$\mathscr{B}_u \leftarrow \phi$

for $i \in \mathscr{B}$ **do**

if $t_i^d \geq \tau$ & $t_i^u \geq \tau$ **then**

$\mathscr{B}_r \leftarrow \mathscr{B}_r \cup \{i\}$

else

$\mathscr{B}_u \leftarrow \mathscr{B}_u \cup \{i\}$

$s_r, c_r \leftarrow$SCOREFORRELIABLECANDIDATE($\mathscr{B}_r$)

$s_u, c_u \leftarrow$SCOREFORUNRELIABLECANDIDATE($\mathscr{B}_u$)

if $s_r \geq s_u$ **then**

return c_r

else

return c_u

using pseudocosts. Initially, pseudocost ρ_i is set to zero for all candidates. When we bt-strong branch on variable i, we increment the number of times bt-strong branch is done (the variables t_i^d, t_i^u). Let the current objective function value be z^*, the objective function value after bt-strong branching for down direction is z_d and the violation distance for down direction computed using Algorithm 4.1 is d_i then pseudocost is updated for the candidates using the formula $\rho_i^d = \frac{\rho_i^d t_i^d + \frac{z^* - z_d}{d_i}}{t_i^d + 1}$. Similarly, pseudocost for up direction is also computed. Pseudocost measures the average lower bound increase per unit of the violation distance. Now, to obtain an estimate of lower bound increase for a candidate, we take the product of the pseudocost and the violation distance for the candidate (see Algorithm 4.5).

In the initial portion of the search tree, the number of unreliable candidates can be very high. Doing bt-strong branching on all such candidates can be computationally very expensive. Thus, we restrict the number of candidates to bt-strong branch by MAXCANDS (20

Algorithm 4.5 Scoring for Reliable Candidates

Input: A set $\mathscr{B}_r$ of reliable candidates, vectors ρ^d, ρ^u representing the pseudocost for down and up direction respectively

Output: Candidate $i \in \mathscr{B}_r$ having the best score and the corresponding score

Parameters: Score factor μ

procedure SCOREFORRELIABLECANDIDATES

$s_{\max} \leftarrow 0$

$c_{\max} \leftarrow 0$

for $i \in \mathscr{B}_r$ **do**

Get violation distances d_i, u_i for the candidate (see Algorithm 4.1).

$d_i \leftarrow \rho_i^d d_i$

$u_i \leftarrow \rho_i^u u_i$

$s_i \leftarrow \mu \min\{d_i, u_i\} + (1-\mu)\max\{d_i, u_i\}$

if $s_i \geq s_{\max}$ **then**

$c_{\max} \leftarrow i$

$s_{\max} \leftarrow s_i$

return $s_{\max}, c_{\max}$

for Minotaur). We need to decide which candidates to bt-strong branch on. It is desirable if we do not bt-strong on the same candidates repeatedly, and we want to bt-strong branch on candidates with a high violation score. Therefore, we sort the unreliable candidates based on $\frac{s_i}{\max\{t_i^d,\, t_i^u\}+1}$, which encourages the candidates with high violation scores and discourages the candidates which are bt-strong branched many times. We then bt-strong branch on MAXCANDS candidates and use pseudocost estimate for remaining candidates (see Algorithm 4.6).

4.4 Computational Results

We selected all instances with either nonconvex quadratic objective or nonconvex quadratic constraints from the MINLPLib dataset [43]. There are 686 such instances. Of these, we only keep those instances for which the root node was fully processed within a time limit of 900 seconds by all branching strategies. We consider a node fully processed when either we prune it,

Algorithm 4.6 Scoring for Unreliable Candidates

Input: A set $\mathscr{B}_u$ of unreliable candidates, vectors indicating number of times up or down branching is done t^u, t^d

Output: Candidate $i \in \mathscr{B}_u$ having the best score and the corresponding score

Parameters: Max candidates to bt-strong branch MAXCANDS

procedure SCOREFORUNRELIABLECANDIDATES

$s_{\max} \leftarrow 0$

$c_{\max} \leftarrow 0$

for $i \in \mathscr{B}_r$ **do**

Get violation score s_i using Algorithm 4.1

$s_i \leftarrow \frac{s_i}{\max\{t_i^d, t_i^u\}+1}$

Sort $\mathscr{B}_u$ based on s_i in descending order

for first MAXCANDS candidates in $\mathscr{B}_u$ **do**

Get bt-strong branching score s_i

if $s_i \geq s_{\max}$ **then**

$c_{\max} \leftarrow i$

$s_{\max} \leftarrow s_i$

for remaining candidates in $\mathscr{B}_u$ **do**

Get pseudocost score s_i (see Algorithm 4.5)

if $s_i \geq s_{\max}$ **then**

$c_{\max} \leftarrow i$

$s_{\max} \leftarrow s_i$

return $s_{\max}, c_{\max}$

or it returns branches. Note that although there is no branching at the root node, the processing time still varies significantly among different branching strategies because of the time taken to select the branching candidate and subsequently create the branches. Particularly, while doing strong branching and bt-strong branching there were several instances for which we could not process the root node within the time limit. After removing those instances where the root node is not processed by some of the branching strategies, we are left with 544 instances, which we call as set $\mathscr{T}$. We have implemented all the five strategies described in the previous section in Minotaur solver [111]. All computational experiments were performed on a computer with a

	Maximum Violation	**Strong**	**Bt-strong**	**Bt-estimate**	**Bt-reliability**
Instances solved within time limit	251	251	254	217	258
Shifted geometric mean of time for $\mathscr{T}_s$	2.06	2.68	2.47	3.38	2.01
Average nodes processed for $\mathscr{T}_s$	3161.80	9231.60	395.74	11244.79	1718.32
Instances where finite upper bound not found	79	133	141	124	90
Average nodes processed for $\mathscr{T}_u$	168719.15	5953.48	2241	15521.11	253522.23
Average relgap for $\mathscr{T}_u^{ub}$	18.24	18.75	19.34	20.12	14.01

Table 4.2: Summary of results comparing different branching strategies

64-bit Intel(R) Xeon(R)E5-2670 v2, 2.50GHz CPU, and 128 GB RAM. We used GCC-4.9.2 to compile our code and CLP-1.17.6 to solve the linear relaxations at every node. We ran our code on a single core of the CPU.

For every instance, we begin by presolving the problem. Then, we transform the problem by substituting quadratic terms with auxiliary variables and creating the McCormick relaxation as described in Section 1.5.1. We presolve the transformed problem and obtain initial root relaxation. We then solve the root relaxation. Let LB^{root} be the objective function value at the optimal solution of the root relaxation. Cutting planes are switched off for the experiments to assess the impact on the lower bound by a given branching strategy independently. Thus, after root relaxation is solved, we call the specific *brancher* [1] which either returns two new subproblems, called branches or sometimes provides modifications to the root relaxation which require further processing of the root node.

We create three subsets of instances in $\mathscr{T}$. Those instances solved by all branching strategies within the time limit of 900 seconds are called the set $\mathscr{T}_s$. There are 207 instances in $\mathscr{T}_s$. The instances not solved by any solver within the time limit of 900 seconds are called

[1] A *brancher* is an implementation of a particular branching strategy

the set $\mathscr{T}_u$. There are 250 instances in $\mathscr{T}_u$. And we define set $\mathscr{T}_u^{ub}$ which are instances in $\mathscr{T}_u$, and all branching strategies obtained a finite upper bound. There are 100 instances in $\mathscr{T}_u^{ub}$. Table 4.2 reports the summary of results obtained. We observe that bt-reliability branching solves the maximum number of instances within the time limit. Although the number of instances solved by bt-reliability branching is not significantly higher than other strategies, we see that bt-estimate branching solves significantly fewer instances. We compute the shifted geometric mean of time and average number of nodes processed for instances in $\mathscr{T}_s$. With respect to time, we observe that bt-reliability brancher is marginally better than maximum violation brancher, while others take even more time. With respect to the number of nodes processed, unsurprisingly, bt-strong branching uses the fewest nodes and beats other branching rules handsomely. However a surprising result is that strong branching performs worse than maximum violation branching. This happens because there are 8 instances where strong branching performs significantly worse than expected. For example, in the instance ST_RV7, the number of nodes processed by maximum violation branching is 397, and bt-strong brancher takes only 71 nodes, but strong brancher takes $1,412,767$ nodes. Such a marked difference arises because while processing several nodes (including the root node), strong branching has all candidates with 0 score i. e. both down and up direction objective value improvement is 0. Then, it selects the variable, which is the lexicographic first variable. In comparison, bt-strong branching provides an appropriate score for all the candidates and does very well in solving the problem. Similar issues happen for other 7 instances as well. If we remove these 8 instances from the average computation, we see that the average number of nodes for strong branching reduces to 2518.52.

For instances in $\mathscr{T}_u$, as one would expect, we see that the number of nodes processed by bt-reliability branching and maximum violation branching is more than other strategies since these are computationally less expensive. Moreover, since the number of nodes processed is higher for maximum violation branching and bt-reliability branching, they can explore more parts of the feasible region. Thus, finite upper bounds are found for more instances with these strategies than others. Also, bt-strong branching processes the least number of nodes; thus, the highest number of instances are there where it cannot find a finite upper bound. We compare the relative optimality gap to assess the performance of branching strategies on these $\mathscr{T}_u$ instances.

We define relative optimality gap as

$$\text{relgap} = \frac{UB - LB}{|UB| + \varepsilon}$$

where UB is the upper bound on the objective value, LB is the lower bound on the objective value, ε is a small positive number to avoid division by zero in case UB is zero. We use $\varepsilon = 10^{-6}$ in our computations. If we have not obtained a finite upper bound for an instance, then the relative gap is not well defined according to our definition. Thus, we compute the average relative optimality gap for only 100 instances in $\mathscr{T}_u^{ub}$. We see that bt-reliability branching performs better than other strategies.

We can see that comparison of branching strategies based on solving time can be done only for solved instances in $\mathscr{T}_s$. In contrast, branching strategies are compared based on relative optimality gap for $\mathscr{T}_u^{ub}$ instances. Also, the union of the sets $\mathscr{T}_s, \mathscr{T}_u^{ub}$ is not the entire set of instances $\mathscr{T}$. To get an overall comparison considering both time taken and gap closed, we compare the pace of each strategy as defined in [76]

$$\text{pace} = \frac{\text{time}}{LB^{\text{end}} - LB^{\text{root}} + \varepsilon}$$

where time is the time taken during solving, LB^{end} is the lower bound obtained after solving, and $\varepsilon = 10^{-6}$ is used to avoid division by zero. Since a good branching strategy often increases the lower bound fastest, pace measures the time taken to increase the lower bound by one unit. Therefore, a better branching strategy will have lower pace and vice versa. This allows us to fairly compare two branching strategies, even if one has solved the instance to optimality while the other has not solved the instance within the time limit. We plot the performance profile [59] of all the branching strategies based on pace in Figure 4.1. For each instance, we first find the best pace among all the strategies and then take the ratio of the pace of each strategy with respect to the best pace. The horizontal axis in Figure 4.1 is a logarithmic axis of pace ratio. The vertical axis is the number of instances where pace ratio for a given strategy is less than the corresponding pace ratio on the horizontal axis. We observe that there is no clear winner among maximum violation branching and bt-reliability branching. There are some instances where maximum violation branching performs better while other instances where bt-reliability

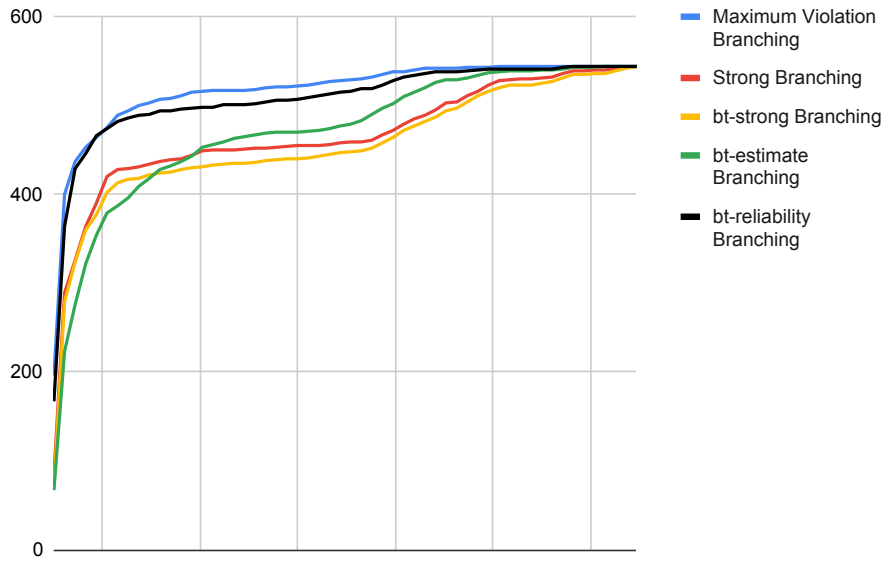


Figure 4.1: Performance profile of different branching strategies based on pace.

branching has a better pace. Other three branching strategies perform worse. Combining results from Table 4.2 and Figure 4.1, we conclude that bt-reliability branching is marginally better than maximum violation branching and dominates other branching strategies.

4.5 Conclusion and Future Work

We have described five different branching strategies and implemented these in Minotaur framework. The branching strategies are designed for spatial branching for MIQCQO problems. For example, we define a measure for violation of nonconvex constraints that takes into account the subsequent relaxation after branching, and our bt-strong branching strategy takes explicitly into account the fact that bound tightening reduces the feasible region nontrivially and allows much better scoring for branching candidates. We develop a reliability branching type setup designed for spatial branching that initializes the pseudocosts based on bt-strong branching instead of strong branching as done in integer branching for MILO problems. Bt-reliability branching also integrates violation scores to compute the distance for pseudocosts and generate a priority list for bt-strong branching. We thoroughly tested these strategies on benchmark instances and reported the results. Our results indicate that bt-reliability branching is better than other branching strategies described, with a close second

being maximum violation branching.

Bt-reliability brancher does bt-strong branching for a fixed MAXCANDS number of unreliable candidates at each node; a possible extension would be to do bt-strong branching for a variable number of unreliable candidates depending on the depth of the node. Another extension is for each variable doing bt-strong branching τ_1 times and then doing strong branching τ_2 times before considering the variable reliable.

Chapter 5

Conclusion and Future Work

MIQCQO problems are hard to solve to global optimality, both theoretically and practically. We develop a general purpose solver based on spatial branch-and-bound algorithm to solve MIQCQO problems. We have discussed the implementations of three components of the solver, namely, presolver, specialized cut generator, and brancher.

In Chapter 2, we described three presolving techniques. Representing a quadratic function efficiently helps in reducing the function and gradient evaluation times. We observe that using a dictionary of keys format of sparse matrix representation significantly helps reduce the function and gradient evaluation time as opposed to using a computational graph. We also described an algorithm for detecting convexity based on separability of the variables. This helps in getting information regarding which constraints or parts of constraints are convex. We then describe three techniques for bound tightening for MIQCQO problems. We observe that doing OBBT significantly reduces the bounds of the variables, and we can solve three more instances within time limit when OBBT is turned on in our solver.

In Chapter 3, we have developed a novel cut generating algorithm for quadratically constrained optimization problem. We prove that our algorithm will always separate an LP basic feasible solution that is infeasible to the original problem. We show that our cuts are

a subset of RLT cuts when the RLT variables are projected out. Our algorithm is fast and generates these cuts by manipulating the simplex tableau. We do thorough computational testing on several variants of our algorithms and show that One-by-One is superior in closing the optimality gap among the variants tested.

In Chapter 4, we describe five branching strategies developed for MIQCQO problems. Two strategies described are analogous to maximum infeasible branching and strong branching in MILO problems. We describe three other branching strategies developed specifically for spatial branching by doing bound tightening before computing the improvement in the lower bound for a given candidate. We developed bt-reliability branching strategy, which is a variant of reliability branching for MILO. Bt-reliability branching combines pseudocost branching, bt-strong branching, and maximum violation branching to obtain a good score for branching candidates. We observe that bt-reliability brancher performs better than other branching strategies in terms of both the time taken to solve and the rate of closing the optimality gap. A close second is maximum violation branching.

In the next section, we benchmark our solver against an open source solver SCIP and a commercial solver Gurobi. We also compare the current `mglob` against an older version of `mglob` to show combined improvements from all the developments discussed in this thesis.

5.1 Performance of `mglob`

We benchmark current `mglob` (git hash `5592878`, commit date October 6, 2023) against SCIP, Gurobi and `mglob` from Minotaur 0.2.2 (git hash `b944ef6`, release date October 15, 2020) when many of the developments described in this thesis were not present. The current version of `mglob` has the following defaults for the algorithms described in the thesis.

- Quadratic functions are represented using `qf` for every instance.
- Convexity detection is turned on. Problems which are detected as convex (i.e. all constraints are detected as convex, and the objective function is also detected as convex) are forwarded to `qg` solver of Minotaur.
- simpleBT and univarBT are done on all nodes. OBBT is done at the root node only.

- Minimum three rounds and maximum 25 rounds of cutting is done using Algorithm 3.1.
- We keep a list of lower bounds on the objective function lb_k where k is the cut round. After three rounds of cutting are done, we do cut generation only if there is a 10% increase in the lower bound. We compute the percent increase in the lower bound in 3 rounds using the below formula.

$$\text{percent increase in lower bound} = \frac{lb_k - lb_{k-3}}{b} \times 100,$$

  where b is taken as the upper bound on the objective function if a finite upper bound is available otherwise, $b = lb_{k-3}$.
- For every violated constraint $\hat{y}_{ij} \neq \hat{x}_i\hat{x}_j$, we generate a cut based on this constraint only if

$$|\hat{y}_{ij} - \hat{x}_i\hat{x}_j| \geq 0.1 \left(\frac{(\overline{x_i} - \underline{x_i})(\overline{x_j} - \underline{x_j})}{4} \right)$$

  where the left side of the expression is the constraint violation and the right side is 0.1 times the maximum possible violation of that constraint. That is, we generate a cut only if the constraint violation is more than 0.1 times the maximum possible violation of the constraint.
- We only add a cut to the relaxation if the current LP solution violates it by at least 10^{-3}.
- bt-reliability branching is used as the branching strategy.

We have taken all 830 convex and nonconvex MIQCQO problems from the MINLPLib dataset [43]. All computational experiments were performed on a computer with a 64-bit Intel(R) Xeon(R)E5-2670 v2, 2.50GHz CPU, and 128 GB RAM. For both versions of Minotaur, we used GCC-4.9.2 to compile our code. Minotaur 0.2.2 uses CLP-1.16.9 as a linear solver for solving linear relaxations and IPOPT-3.12.7 as an NLP solver. The current version of Minotaur uses CLP-1.17.6 as a linear solver and IPOPT-3.14.12 as an NLP solver. Binary files provided by SCIP Optimization Suite 8.0.2 were used to test the performance of SCIP. Gurobi 10.0.1 was used to test the performance of Gurobi. We have set a time limit of 600 seconds for each instance.

	mglob current	mglob 0.2.2	SCIP	Gurobi
Instances solved correctly within time limit	342	203	464	548
Instance with errors	3	90	2	0
Instance with incorrect bounds	7	133	5	9
Instances solved within time limit by all solvers	186	186	186	186
Shifted geometric mean time	1.86	1.35	0.60	0.03
Instances with finite lower and upper bounds	553	330	737	788
Instances with finite lower and upper bounds by all solvers	296	296	296	296
Average relative gap for instances having finite bounds by all solvers	0.28	0.29	0.08	0.09

Table 5.1: Comparison of `mglob` current, `mglob` 0.2.2, SCIP, and Gurobi

Table 5.1 summarises the results obtained for all three solvers. We see that Gurobi solves the maximum number of instances within time limit, followed by SCIP, then current `mglob` comes third, and `mglob` 0.2.2 solves the least. `mglob` 0.2.2 had 90 instances for which it failed. These were common errors reported like invalid memory access, use of uninitialized pointers, accessing freed memory, etc. Currently, `mglob` fails on only three instances, while SCIP fails on two instances, and Gurobi does not report any errors. There were 133 instances in `mglob` 0.2.2, which gave incorrect solutions. A solution is considered incorrect when the lower bound obtained is more than the upper bound available in MINLPLib or when the upper bound obtained is less than the lower bound available in MINLPLib. Currently, there are seven instances where such incorrect solutions are obtained, and we see that these are numerical issues within the solver. SCIP has five instances providing incorrect solutions, which are only numerical issues, and Gurobi has nine.

Looking at the shifted geometric mean of time taken to solve, we see that current `mglob` is worse than that of `mglob` 0.2.2. This is because we use only 188 instances solved by all solvers within time limit. These are easy instances, and some techniques discussed in this thesis, like OBBT and bt-reliability branching, which solve many LPs at the root node, are optional to solve them. This can be observed further by the fact that 253 more instances in current `mglob` have

both finite upper and lower bounds compared to `mglob` 0.2.2.

We can thus conclude that current `mglob` significantly performs better than `mglob` 0.2.2. It is more stable, gives correct solutions for most instances within tolerance, and is reasonably faster. While the current `mglob` does not perform as well as SCIP 8.0.2 or Gurobi 10.0.1, many promising future developments can be done so that `mglob` can become a more competitive solver. We describe some of the future work in the next section.

5.2 Future Work

There are several promising future research directions and further development of our solver for MIQCQO problems and general nonconvex problems.

Exploiting convexity - Our convexity detection algorithm described in Section 2.2 detects convexity for each constraint separately. For convex constraints in nonconvex problems, one can use stronger relaxation techniques rather than McCormick relaxation as currently done in `mglob`. Consider a problem with the following constraints

$$\begin{aligned}(2x_1 + 3x_2 - x_3 - x_4)^2 &\leq 1,\\ x_1 + 3x_1x_3 &\leq 5.\end{aligned}$$

Currently, `mglob` will create a McCormick relaxation for each quadratic term (x_1^2, x_1x_2, etc.) in the first constraint as well for x_1x_3 in the second constraint. We can easily relax the first constraint by adding tangent inequalities at some points, and the second constraint can be relaxed using McCormick relaxation. Such reformulations and relaxations can also be aimed at constraints, which are convex in some variables and nonconvex in other variables.

Bound Tightening - Many advanced bound tightening approaches in the literature can be explored. For instance, reduced cost bounding as described in [117], faster OBBT operations [77, 46] etc. Other presolving techniques like probing, as done for MILO problems, can also be implemented to improve the coefficients further or tighten the bounds of the variables.

Cuts - As seen in Chapter 3, our cutting plane algorithm can potentially generate several cuts

for a problem. A good cut selection strategy that identifies deeper cuts can be considered to integrate our cuts nicely with a general purpose solver. Our cuts are quite dense since they depend on the optimal simplex tableau. A good sparsification strategy that produces deep cuts can also be studied. One can add more cutting planes like the αBB cuts, semidefinite cuts, disjunctive cuts, etc., for MIQCQO problems.

Branching - We have presented bt-reliability branching setup in Section 4.3.5. It has a parameters like MAXCANDS, τ. A tuning of these parameters such that the algorithm works well for a diverse set of instances can be studied. One can also implement branching strategies based on machine learning as discussed in [76] and others.

Heuristics - Currently, our solver only does multi-start local search for problems with no integer variables. Other heuristics can also be implemented, like feasibility pump, diving, large neighborhood search etc. Metaheuristics like particle swarm optimization, genetic algorithm, and ant colony optimization can also be tried. These give good quality feasible solutions at the start of the root node so that faster tree pruning can be achieved.

Integration with MILO solver - Many convex MINLO solvers, including those in Minotaur, solve MILO relaxation of MINLO problems instead of LO relaxation. These algorithms then use a single tree exploration for MINLO and MILO subproblems. Similar techniques for nonconvex problems can be implemented. This also allows to add cuts based on specific MILO structures like knapsack cuts, flow cover cuts, Gomory's mixed integer cuts, etc.

Parallel computations - Many tree search based algorithms can be easily parallelized to improve the performance. Shared memory parallel extensions to our sB&B algorithm are an evident future work. Apart from parallelizing sB&B, many other components, like the presolver, heuristics, etc., can also be parallelized.

References

[1] LP file format: algebraic representation. https://www.ibm.com/docs/en/icos/22.1.1?topic=cplex-lp-file-format-algebraic-representation. Accessed: 2023-11-28.

[2] MPS file format: industry standard. https://www.ibm.com/docs/en/icos/22.1.1?topic=cplex-mps-file-format-industry-standard. Accessed: 2023-11-28.

[3] POLIP - library for polynomially constrained mixed-integer programming. https://polip.zib.de/pipformat.php. Accessed: 2023-11-28.

[4] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[5] Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In *Facets of combinatorial optimization: Festschrift for martin grötschel*, pages 449–481. Springer, 2013.

[6] Warren P Adams and Terri A Johnson. Improved linear programming-based lower bounds for the quadratic assignment problem. *DIMACS series in discrete mathematics and theoretical computer science*, 16:43–77, 1994.

[7] Nilanjan Adhya, Mohit Tawarmalani, and Nikolaos V Sahinidis. A Lagrangian approach to the pooling problem. *Industrial & Engineering Chemistry Research*, 38(5):1956–1972, 1999.

[8] Claire S Adjiman, Stefan Dallwig, Christodoulos A Floudas, and Arnold Neumaier. A global optimization method, αBB, for general twice-differentiable constrained NLPs—I. theoretical advances. *Computers & Chemical Engineering*, 22(9):1137–1158, 1998.

[9] A Aggarwal and CA Floudas. Synthesis of general distillation sequences—nonsharp separations. *Computers & chemical engineering*, 14(6):631–653, 1990.

[10] Faiz A Al-Khayyal and James E Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8(2):273–286, 1983.

[11] Mohammed Alfaki and Dag Haugland. Strong formulations for the pooling problem. *Journal of Global Optimization*, 56:897–916, 2013.

[12] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

[13] Martin S Andersen, Anders Hansson, and Lieven Vandenberghe. Reduced-complexity semidefinite relaxations of optimal power flow problems. *IEEE Transactions on Power Systems*, 29(4):1855–1863, 2013.

[14] Giuseppe Andreello, Alberto Caprara, and Matteo Fischetti. Embedding {0, ½}-Cuts in a Branch-and-Cut Framework: A Computational Study. *INFORMS Journal on Computing*, 19:229–238, 05 2007.

[15] Ioannis P Androulakis, Costas D Maranas, and Christodoulos A Floudas. αBB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995.

[16] D. Applegate, R. Bixby, V. Chvatal, and B. Cook. Finding cuts in the tsp (a preliminary report). Technical report, 1995.

[17] MOSEK ApS. *Introducing the MOSEK Optimization Suite 10.1.15*, 2022.

[18] Charles Audet, Jack Brimberg, Pierre Hansen, Sébastien Le Digabel, and Nenad Mladenović. Pooling problem: Alternate formulations and solution methods. *Management science*, 50(6):761–776, 2004.

[19] Charles Audet, Pierre Hansen, Brigitte Jaumard, and Gilles Savard. A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Mathematical Programming*, 87(1):131–152, 2000.

[20] Charles Audet, Pierre Hansen, and Frédéric Messine. The small octagon with longest perimeter. *Journal of Combinatorial Theory, Series A*, 114(1):135–150, 2007.

[21] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International conference on machine learning*, pages 344–353. PMLR, 2018.

[22] Mokhtar S Bazaraa, John J Jarvis, and Hanif D Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[23] Pietro Belotti, Timo Berthold, and Kelligton Neves. Algorithms for discrete nonlinear optimization in FICO Xpress. In *2016 IEEE Sensor Array and Multichannel Signal Processing Workshop (SAM)*, pages 1–5. IEEE, 2016.

[24] Pietro Belotti, Sonia Cafieri, Jon Lee, and Leo Liberti. Feasibility-based bounds tightening via fixed points. In *International Conference on Combinatorial Optimization and Applications*, pages 65–76. Springer, 2010.

[25] Pietro Belotti, Jon Lee, Leo Liberti, Francois Margot, and Andreas Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods & Software*, 24(4-5):597–634, 2009.

[26] Aharon Ben-Tal, Gideon Eiger, and Vladimir Gershovitz. Global minimization by reducing the duality gap. *Mathematical programming*, 63(1-3):193–212, 1994.

[27] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and Olivier Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1:76–94, 1971.

[28] Timo Berthold, Ambros M. Gleixner, Stefan Heinz, and Stefan Vigerske. Analyzing the computational impact of MIQCP solver components. *Numerical Algebra, Control and Optimization*, 2(4):739–748, 2012.

[29] Dimitri Bertsekas, Angelia Nedic, and Asuman Ozdaglar. *Convex analysis and optimization*, volume 1. Athena Scientific, 2003.

[30] Dimitris Bertsimas and Ryan Cory-Wright. A scalable algorithm for sparse portfolio selection. *Informs journal on computing*, 34(3):1489–1511, 2022.

[31] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena scientific Belmont, MA, 1997.

[32] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 8.0. Technical report, Optimization Online, December 2021.

[33] Ksenia Bestuzheva, Ambros Gleixner, and Tobias Achterberg. Efficient separation of rlt cuts for implicit and explicit bilinear products. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 14–28. Springer, 2023.

[34] Daniel Bienstock, Chen Chen, and Gonzalo Munoz. Outer-product-free sets for polynomial optimization and oracle-based cuts. *Mathematical Programming*, 183(1-2):105–148, 2020.

[35] Christian Bingane, Miguel F Anjos, and Sébastien Le Digabel. Tight-and-cheap conic relaxation for the ac optimal power flow problem. *IEEE Transactions on Power Systems*, 33(6):7181–7188, 2018.

[36] Johannes Bisschop. *AIMMS optimization modeling*. Lulu. com, 2006.

[37] Robert Bixby and Edward Rothberg. Progress in computational mixed integer programming–a look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37, 2007.

[38] Immanuel M Bomze. Branch-and-bound approaches to standard quadratic optimization problems. *Journal of Global Optimization*, 22(1):17–37, 2002.

[39] Pierre Bonami and Jon Lee. Bonmin user's manual. *Numer Math*, 4:1–32, 2007.

[40] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[41] Christoph Buchheim, Angelika Wiegele, and Lanbo Zheng. Exact algorithms for the quadratic linear ordering problem. *INFORMS Journal on Computing*, 22(1):168–177, 2010.

[42] Samuel Burer and Anureet Saxena. The MILP road to MIQCP. In Jon Lee and Sven Leyffer, editors, *Mixed Integer Nonlinear Programming*, pages 373–405, New York, NY, 2012. Springer New York.

[43] Michael R Bussieck, Arne Stolbjerg Drud, and Alexander Meeraus. MINLPLib—a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003.

[44] Michael R Bussieck and Stefan Vigerske. MINLP Solver Software. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[45] Michael L. Bynum, Gabriel A. Hackebeil, William E. Hart, Carl D. Laird, Bethany L. Nicholson, John D. Siirola, Jean-Paul Watson, and David L. Woodruff. *Pyomo–optimization modeling in python*, volume 67. Springer Science & Business Media, third edition, 2021.

[46] Michael Lee Bynum, Andrea R Castillo, Bernard Knueven, John Daniel Siirola, and Carl Damon Laird. Decomposing Optimization-Based Bounds Tightening Problems Via Graph Partitioning. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2019.

[47] Emilio Carrizosa, Pierre Hansen, and Frédéric Messine. Improving interval analysis bounds by translations. *Journal of Global Optimization*, 29(2):157–172, 2004.

[48] Pedro M Castro, Henrique A Matos, and Augusto Q Novais. An efficient heuristic procedure for the optimal design of wastewater treatment systems. *Resources, conservation and recycling*, 50(2):158–185, 2007.

[49] Pedro M Castro and João P Teles. Comparison of global optimization algorithms for the design of water-using networks. *Computers & chemical engineering*, 52:249–261, 2013.

[50] Pedro M Castro, Joao P Teles, and Augusto Q Novais. Linear program-based algorithm for the optimal design of wastewater treatment systems. *Clean Technologies and Environmental Policy*, 11:83–93, 2009.

[51] Carleton Coffrin, Hassan L Hijazi, and Pascal Van Hentenryck. Strengthening convex relaxations with bound tightening for power network optimization. In *International Conference on Principles and Practice of Constraint Programming*, pages 39–57. Springer, 2015.

[52] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer programming*, volume 271. Springer.

[53] John Horton Conway and Neil James Alexander Sloane. *Sphere packings, lattices and groups*, volume 290. Springer Science & Business Media, 2013.

[54] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[55] Gérard Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical programming*, 112(1):3–44, 2008.

[56] IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[57] Henrik Dahl, Alexander Meeraus, and Stavros A Zenios. *Some financial optimization models: I. risk management*. Fishman-Davidson Center for the Study of the Service Sector, Wharton School . . . , 1989.

[58] Santanu S Dey, Yatharth Dubey, Marco Molinaro, and Prachi Shah. A theoretical and computational analysis of full strong-branching. *Mathematical Programming*, 205(1):303–336, 2024.

[59] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91:201–213, 2002.

[60] Ferenc Domes and Arnold Neumaier. Constraint propagation on quadratic constraints. *Constraints*, 15(3):404–429, 2010.

[61] Ferenc Domes and Arnold Neumaier. Rigorous filtering using linear relaxations. *Journal of Global Optimization*, 53(3):441–473, 2012.

[62] Matteo Fischetti, Ivana Ljubić, and Markus Sinnl. Redesigning benders decomposition for large-scale facility location. *Management Science*, 63(7):2146–2162, 2017.

[63] Matteo Fischetti and Michele Monaci. A branch-and-cut algorithm for mixed-integer bilinear programming. *European Journal of Operational Research*, 282(2):506–514, 2020.

[64] Christodoulos A Floudas, Panos M Pardalos, Claire Adjiman, William R Esposito, Zeynep H Gümüs, Stephen T Harding, John L Klepeis, Clifford A Meyer, and Carl A Schweiger. *Handbook of test problems in local and global optimization*, volume 33. Springer Science & Business Media, 2013.

[65] JJH Forrest, JPH Hirst, and John A Tomlin. Practical solution of large mixed integer programming problems with UMPIRE. *Management Science*, 20(5):736–773, 1974.

[66] John Forrest, Ted Ralphs, Haroldo Gambini Santos, Stefan Vigerske, John Forrest, Lou Hafer, Bjarni Kristjansson, jpfasano, EdwinStraver, Miles Lubin, Jan-Willem, rlougee, jpgoncal1, Samuel Brito, h-i gassmann, Cristina, Matthew Saltzman, tosttost, Bruno Pitrus, Fumiaki MATSUSHIMA, and to st. coin-or/cbc: Release releases/2.10.10, April 2023.

[67] John Forrest, Stefan Vigerske, Ted Ralphs, Lou Hafer, JP Fasano, Haroldo Gambini Santos, Matthew Saltzman, Horand Gassmann, Bjarni Kristjansson, and Alan King. coin-or/Clp: Version 1.17.6 (releases/1.17.6). Zenodo, April 2020.

[68] Robert Fourer, David M Gay, and Brian W Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.

[69] Robert Fourer, David M Gay, and Brian W Kernighan. Design principles and new developments in the AMPL modeling language. *Modeling languages in mathematical optimization*, pages 105–135, 2004.

[70] Antonio Frangioni, Fabio Furini, and Claudio Gentile. Approximated perspective relaxations: a project and lift approach. *Computational Optimization and Applications*, 63:705–735, 2016.

[71] Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[72] Jianjun Gao and Duan Li. Optimal cardinality constrained portfolio selection. *Operations research*, 61(3):745–761, 2013.

[73] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[74] David M Gay. Writing. nl files. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2005.

[75] Bissan Ghaddar, Miguel F Anjos, and Frauke Liers. A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem. *Annals of Operations Research*, 188(1):155–174, 2011.

[76] Bissan Ghaddar, Ignacio Gómez-Casares, Julio González-Díaz, Brais González-Rodríguez, Beatriz Pateiro-López, and Sofía Rodríguez-Ballesteros. Learning for spatial branching: An algorithm selection approach. *INFORMS Journal on Computing*, 35(5):1024–1043, 2023.

[77] Ambros M Gleixner, Timo Berthold, Benjamin Müller, and Stefan Weltge. Three enhancements for optimization-based bound tightening. *Journal of Global Optimization*, 67(4):731–757, 2017.

[78] Ambros M Gleixner, Daniel E Steffy, and Kati Wolter. Iterative refinement for linear programming. *INFORMS Journal on Computing*, 28(3):449–464, 2016.

[79] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275 – 278, 1958.

[80] Ralph E. Gomory. An Algorithm for the mixed Integer Problem. *Report No. P-1885, The Rand Corporation, Santa Monica, CA.*, 1960.

[81] Brais González-Rodríguez, Joaquín Ossorio-Castillo, Julio González-Díaz, Ángel M González-Rueda, David R Penas, and Diego Rodríguez-Martínez. Computational advances in polynomial optimization: RAPOSa, a freely available global solver. *Journal of Global Optimization*, 85(3):541–568, 2023.

[82] Andreas Griewank. Automatic differentiaion of algorithms: Theory, implementation and application. *Siam*, 1991.

[83] Ignacio E Grossmann, Jagadisan Viswanathan, Aldo Vecchietti, Ramesh Raman, Erwin Kalvelagen, et al. Gams/dicopt: A discrete continuous optimization package. *GAMS Corporation Inc*, 37:55, 2002.

[84] Oktay Günlük, Jon Lee, and Robert Weismantel. MINLP strengthening for separable convex quadratic transportation-cost ufl. *IBM Res. Report*, pages 1–16, 2007.

[85] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[86] Iiro Harjunkoski, Ray Pörn, and Tapio Westerlund. *MINLP: trim-loss problem*, pages 1469–1477. Springer US, Boston, MA, 2001.

[87] Iiro Harjunkoski, Tapio Westerlund, Ray Pörn, and Hans Skrifvars. Different transformations for solving non-convex trim-loss problems by minlp. *European Journal of Operational Research*, 105(3):594–603, 1998.

[88] C A Haverly. Studies of the behavior of recursion for the pooling problem. *ACM SIGMAP Bulletin*, (25):19–28, 1978.

[89] Sunderesh S Heragu and Andrew Kusiak. Machine layout problem in flexible manufacturing systems. *Operations research*, 36(2):258–268, 1988.

[90] Hassan Hijazi, Carleton Coffrin, and Pascal Van Hentenryck. Convex quadratic relaxations for mixed-integer nonlinear programs in power systems. *Mathematical Programming Computation*, 9:321–367, 2017.

[91] Reiner Horst and Hoang Tuy. *Global optimization: Deterministic approaches*. Springer Science & Business Media, 2013.

[92] Zeren Huang, Kerong Wang, Furui Liu, Hui-Ling Zhen, Weinan Zhang, Mingxuan Yuan, Jianye Hao, Yong Yu, and Jun Wang. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, 123:108353, 2022.

[93] Qi Huangfu and JA Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

[94] R. C. Jeroslow. There cannot be any algorithm for Integer Programming with Quadratic constraints. *Operations Research*, 21(1):221–224, February 1973.

[95] Tomáš Kaiser, Maria Saumell, and Nico Van Cleemput. 10-Gabriel graphs are Hamiltonian. *Information Processing Letters*, 115(11):877–881, 2015.

[96] Josef Kallrath. Cutting circles and polygons from area-minimizing rectangles. *Journal of Global Optimization*, 43(2):299–328, 2009.

[97] Julia Kallrath, Steffen Rebennack, Josef Kallrath, and Rüdiger Kusche. Solving real-world cutting stock-problems in the paper industry: Mathematical approaches, experience and challenges. *European Journal of Operational Research*, 238(1):374–389, 2014.

[98] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[99] Sergei Kucherenko, Pietro Belotti, Leo Liberti, and Nelson Maculan. New formulations for the kissing number problem. *Discrete Applied Mathematics*, 155(14):1837–1841, 2007.

[100] Xiang Li, Emre Armagan, Asgeir Tomasgard, and Paul I Barton. Stochastic pooling problem for natural gas production network design and operation under uncertainty. *AIChE Journal*, 57(8):2120–2135, 2011.

[101] Xiang Li, Asgeir Tomasgard, and Paul I Barton. Decomposition strategy for the stochastic pooling problem. *Journal of Global Optimization*, 54(4):765–790, 2012.

[102] Leo Liberti. Linearity embedded in nonconvex programs. *Journal of Global Optimization*, 33:157–196, 2005.

[103] Leo Liberti and Constantinos C Pantelides. An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms. *Journal of Global Optimization*, 36(2):161–189, 2006.

[104] Jeff T Linderoth and Martin WP Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[105] Marco Locatelli and Fabio Schoen. *Global optimization: theory, algorithms, and applications*. SIAM, 2013.

[106] Andrea Lodi and Giulia Zarpellon. On learning and branching: a survey. *TOP*, 25:207–236, 2017.

[107] Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, and Juan Pablo Vielma. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 2023.

[108] James Luedtke, Mahdi Namazifar, and Jeff Linderoth. Some results on the strength of relaxations of multilinear functions. *Mathematical programming*, 136(2):325–351, 2012.

[109] Andreas Lundell, Jan Kronqvist, and Tapio Westerlund. The supporting hyperplane optimization toolkit for convex MINLP. *Journal of Global Optimization*, 84(1):1–41, 2022.

[110] Ashutosh Mahajan. Presolving mixed-integer linear programs. *Wiley Encyclopedia of Operations Research and Management Science*, pages 4141–4149, 2010.

[111] Ashutosh Mahajan, Sven Leyffer, Jeff Linderoth, James Luedtke, and Todd Munson. Minotaur: A mixed-integer nonlinear optimization toolkit. *Mathematical Programming Computation*, 13(2):301–338, 2021.

[112] Garth P McCormick. Computability of global solutions to factorable nonconvex programs: Part-I Convex underestimating problems. *Mathematical programming*, 10(1):147–175, 1976.

[113] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601, 1992.

[114] Clifford A Meyer and Christodoulos A Floudas. Convex envelopes for edge-concave functions. *Mathematical programming*, 103(2):207–224, 2005.

[115] Ruth Misener and Christodoulos Floudas. Advances for the pooling problem: Modeling, global optimization, and computational studies Survey. *Applied and Computational Mathematics*, 8, 01 2009.

[116] Ruth Misener and Christodoulos A Floudas. Global optimization of mixed-integer quadratically-constrained quadratic programs (MIQCQP) through piecewise-linear and edge-concave relaxations. *Mathematical Programming*, 136(1):155–182, 2012.

[117] Ruth Misener and Christodoulos A. Floudas. GloMIQO: Global mixed-integer quadratic optimizer. *Journal of Global Optimization*, 57(1):3–50, Sep 2013.

[118] Ruth Misener and Christodoulos A Floudas. ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations. *Journal of Global Optimization*, 59(2-3):503–526, 2014.

[119] Ruth Misener, Chrysanthos E Gounaris, and Christodoulos A Floudas. Mathematical modeling and global optimization of large-scale extended pooling problems with the (EPA) complex emissions constraints. *Computers & chemical engineering*, 34(9):1432–1456, 2010.

[120] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*. SIAM, 2009.

[121] Sylvain Mouret and Ignacio E Grossmann. Crude-oil operations scheduling. *Available from CyberInfrastructure for MINLP at: https://www. minlp. org/library/problem/index. php*, 2010.

[122] TJ Mullin and P Belotti. Using branch-and-bound algorithms to optimize selection of a fixed-size breeding population under a relatedness constraint. *Tree genetics & genomes*, 12(1):4, 2016.

[123] Harsha Nagarajan, Mowen Lu, Site Wang, Russell Bent, and Kaarthik Sundar. An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs. *Journal of Global Optimization*, 2019.

[124] Harsha Nagarajan, Mowen Lu, Emre Yamangil, and Russell Bent. Tightening McCormick relaxations for nonlinear programs via dynamic multivariate partitioning. In *International Conference on Principles and Practice of Constraint Programming*, pages 369–387. Springer, 2016.

[125] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[126] Laurent Perron and Vincent Furnon. Or-tools.

[127] Svatopluk Poljak and Henry Wolkowicz. Convex relaxations of (0, 1)-quadratic programming. *Mathematics of Operations Research*, 20(3):550–561, 1995.

[128] Yash Puranik and Nikolaos V Sahinidis. Domain reduction techniques for global NLP and MINLP optimization. *Constraints*, 22(3):338–376, 2017.

[129] Richard E Rosenthal. A gams tutorial. *GAMS-A User's Guide*, 5(26):649, 2007.

[130] Nikolaos V Sahinidis and Mohit Tawarmalani. Accelerating branch-and-bound through a modeling language construct for relaxation-specific constraints. *Journal of Global Optimization*, 32:259–280, 2005.

[131] Anureet Saxena, Pierre Bonami, and Jon Lee. Disjunctive cuts for non-convex mixed integer quadratically constrained programs. In *Integer Programming and Combinatorial Optimization: 13th International Conference, IPCO 2008 Bertinoro, Italy, May 26-28, 2008 Proceedings 13*, pages 17–33. Springer, 2008.

[132] Hermann Schichl and Arnold Neumaier. Interval analysis on directed acyclic graphs for global optimization. *Journal of Global Optimization*, 33(4):541–562, 2005.

[133] Hanif D Sherali and Warren P Adams. *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*, volume 31. Springer Science & Business Media, 2013.

[134] Hanif D Sherali and Amine Alameddine. A new reformulation-linearization technique for bilinear programming problems. *Journal of Global optimization*, 2(4):379–410, 1992.

[135] Hanif D Sherali, Evrim Dalkiran, and Leo Liberti. Reduced RLT representations for nonconvex polynomial programming problems. *Journal of Global Optimization*, 52:447–469, 2012.

[136] Hanif D Sherali and Cihan H Tuncbilek. A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Journal of Global Optimization*, 2:101–112, 1992.

[137] Naum Z Shor. Quadratic optimization problems. *Soviet Journal of Computer and Systems Sciences*, 25:1–11, 1987.

[138] Mohit Tawarmalani and Nikolaos V Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical programming*, 99(3):563–591, 2004.

[139] João Teles, Pedro M Castro, and Augusto Q Novais. LP-based solution strategies for the optimal design of industrial water networks with multiple contaminants. *Chemical Engineering Science*, 63(2):376–394, 2008.

[140] Joao P Teles, Pedro M Castro, and Henrique A Matos. Global optimization of water networks design using multiparametric disaggregation. *Computers & Chemical Engineering*, 40:132–147, 2012.

[141] Mark Turner, Thorsten Koch, Felipe Serrano, and Michael Winkler. Adaptive cut selection in mixed-integer linear programming. *Open Journal of Mathematical Optimization*, 4:1–28, 2023.

[142] Robert J Vanderbei et al. *Linear programming*. Springer, 2020.

[143] Juan Pablo Vielma, Shabbir Ahmed, and George 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.

[144] Stefan Vigerske and Ambros Gleixner. SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. *Optimization Methods and Software*, 33(3):563–593, 2018.

[145] Franz Wesselmann and U Stuhl. Implementing cutting plane management and selection techniques. *University of Paderborn, Tech. Rep*, 2012.

[146] Tapio Westerlund and Kurt Lundqvist. *Alpha-ECP, version 5.01: An interactive MINLP-solver based on the extended cutting plane method*. Åbo Akademi Turku, Finland, 2001.

[147] Fengqi You and Ignacio E Grossmann. Mixed-integer nonlinear programming models and algorithms for large-scale supply chain design with stochastic inventory management. *Industrial & Engineering Chemistry Research*, 47(20):7802–7817, 2008.

[148] Lu Zhen, Yiwei Wu, Shuaian Wang, Yi Hu, and Wen Yi. Capacitated closed-loop supply chain network design under uncertainty. *Advanced Engineering Informatics*, 38:306–315, 2018.

[149] XJ Zheng, XL Sun, and Duan Li. Nonconvex quadratically constrained quadratic programming: best DC decompositions and their SDP representations. *Journal of Global Optimization*, 50(4):695–712, 2011.

This page was intentionally left blank.

List of Publications and Presentations

- **Mustafa Vora**, and Ashutosh Mahajan. "Cutting planes from the simplex tableau for quadratically constrained optimization problems." *Tech. Rep. Optimization Online, Submitted to INFORMS Journal on Computing* 2023.

- **Mustafa Vora** and Ashutosh Mahajan. "Cutting planes from the Simplex Tableau for Quadratically Constrained Problems." *HUGO 2022 - XV. Workshop on Global Optimization, Szeged, Hungary*, September 6-8 2022.

- **Mustafa Vora** and Ashutosh Mahajan, "Deriving cutting planes for Quadratically Constrained Problems." *Summer School on Large Scale Optimization, IIM Ahmedabad*, May 6-13 2022.

- **Mustafa Vora**, Meenarli Sharma, Prashant Palkar and Ashutosh Mahajan. "Solving Mixed-Integer Nonlinear Optimization Problems Using MINOTAUR." *52nd Annual Convention of ORSI & International Conference, IIM Ahmedabad*, December 15-18 2019

This page was intentionally left blank.

Acknowledgments

There are several people who have made many contributions towards my academic and non-academic life during my PhD journey and whose contributions have made this thesis possible. I would like to take this opportunity to express my deepest gratitude to those who were part of this journey.

Firstly, I would deeply thank my advisor Prof. Ashutosh Mahajan whose guidance and support at every stage of my PhD made me sail through this journey. His cheerful attitude and a knack of cracking pinpointed jokes always helped me stay positive and motivated. His keen remarks with respect to my work and attention to detail allowed me to correct several of my mistakes and provided many new ideas that I can work upon. I have also had the opportunity to work alongside him for Institute timetabling, Teaching Assistant in several courses, and other teaching and non-teaching assignments which broadened my perspective and made me understand practical implications of Operations Research as a whole. It was a great privilege to work with him and his contributions in this thesis are immeasurable.

I would also like to thank Prof. Narayan Rangaraj, Prof. Vishnu Narayanan, and Prof. Avinash Bhardwaj for agreeing to be a part of my Research Progress Committee. I had many insightful discussions with them during my Annual Progress Seminars and otherwise that helped me a lot in my thesis work. I would like to thank my faculty advisor Prof. Veeraruna Kavitha for her guidance during my first year about courses, curriculum and helping me get accustomed to academic life at IITB easily. I would thank all the faculty of IEOR department with whom I have taken many courses. I would also like to thank the office staff of IEOR who helped me greatly with administrative work.

I would like to thank my seniors Prashant Palkar, Meenarli Sharma, Tejas Ghorpade, Swapnesh Subramanian and others for their academic and non-academic advice. I would like to thank my colleagues and friends from IEOR Khushboo, Rishav, Akul, Vanessa, Vartika,

Mufassir, Chhavi, Sambit, Aanchal, Akshay, Prem, Sanket, Simran, Pranav, Abhishek and others. I had a great time with them and they helped me in many academic and personal matters. I would also like to thank my friends Apurva and Shabnam for the many memories that we have shared together during my time in IITB.

I am deeply grateful to my parents, Makbul and Rashida, whose constant support and innumerable sacrifices allowed me to pursue my career. I am deeply humbled by the patience and sacrifices of my wife Sakina during my PhD years. Her support and words of encouragement made me going all this time and I am deeply thankful to have such a partner.

I would also like to thank MHRD Government of India for the financial support during my PhD.

Vora Mustafa Makbul

IIT Bombay

July 10, 2024