Advances in Algorithms for Convex Mixed-Integer Nonlinear Optimization

A Thesis
Submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy
by

Meenarli Sharma
(Roll No. 134190001)

Supervisor:
Prof. Ashutosh Mahajan

Industrial Engineering and Operations Research
Indian Institute of Technology Bombay
Mumbai 400076 (India)

July 8, 2021

Acceptance Certificate

Industrial Engineering and Operations Research

Indian Institute of Technology, Bombay

The thesis entitled "Advances in Algorithms for Convex Mixed-Integer Nonlinear Optimization" submitted by Meenarli Sharma (Roll No. 134190001) may be accepted for being evaluated.

Date: July 8, 2021

Prof. Ashutosh Mahajan

Approval Sheet

This thesis entitled “Advances in Algorithms for Convex Mixed-Integer Nonlinear Optimization” by Meenarli Sharma is approved for the degree of Doctor of Philosophy.

_____

_____

_____

Examiners

_____

_____

_____

Supervisor (s)

_____

Chairman

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 declare that I have properly and accurately acknowledged all sources used in the production of this report. 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 a 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.

Meenarli Sharma

Date: July 8, 2021 (Roll No. 134190001)

Abstract

This thesis focuses on solving a particular class of mathematical optimization problems called convex mixed-integer nonlinear programs (MINLPs). Many exciting and challenging real-world problems are modeled as MINLPs. These problems are known to be theoretically hard, but advancements in the computing infrastructure, algorithms, and solvers have enabled solving these problems faster in practice. Convexity plays a central role in developing useful techniques for handling convex MINLPs. A significant part of this thesis aims to automatically detect specific structures in convex MINLPs for practically enhancing the performance of the methods based on a branch-and-cut framework. These structures help generate tight linear approximations and, thus, tight relaxations and better bounds on the optimal value of the problem.

First, we aim to advance one of the most effective methods, LP/NLP based branch-and-bound, for solving convex MINLPs. While the algorithm is known to take a finite number of steps, careful implementation and control are required for it to be practically useful. We develop schemes for deciding when additional cuts can be generated and at what points. These extra linearizations help strengthen the linear approximation of nonlinear constraints at different nodes in the branch-and-bound tree. Two of the techniques are specifically applicable to commonly found univariate nonlinear functions and are more effective than other general approaches.

Next, we discuss two prevalent structures in convex MINLPs. One is a specific disjunctive set called the 'on-off' set, and the other is the 'separability' property in nonlinear constraints. Reformulations using these structures are known to provide tight polyhedral approximations. However, detecting them in a problem can be as hard as solving the original problem. We present computationally economical ways to automatically recognize these structures and exploit them in a branch-and-cut framework of the LP/NLP based branch-and-bound method.

All the above techniques have been implemented in MINOTAUR, an open-source toolkit for MINLPs.

Finally, we present a highly challenging class of optimization problems called the mixed-integer partial differential equations constrained optimization (MIPDECO) problems. These problems are challenging because of both the combinatorial complexity of integer variables and the computational difficulties of the discretized PDEs. Little is known about this class of problems or solution approaches, and one of the motivations of this work is to experiment with state-of-the-art mixed-integer solvers for solving this class of problems. We present a convection-diffusion inverse problem that aims to identify an unknown number of sources and their locations. We model the sources using a binary function, and we show that the inverse problem can be formulated as a large-scale MINLPs. We show empirically that current state-of-the-art mixed-integer solvers cannot solve this problem, and that applying simple rounding heuristics to solutions of the relaxed problem can fail to identify the correct number and location of the sources. We develop two new rounding heuristics that exploit the value and a physical interpretation of the continuous relaxation solution. We apply a steepest-descent improvement heuristic to obtain satisfactory solutions to both two- and three-dimensional inverse problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We study optimization problems of finding a solution vector that minimizes a given objective function while satisfying a given set of constraints. More specifically, we study problems that can be cast in the following form

$$\left.\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & g_i(x) \leq 0,\ i \in M, \\ & x_i \in \mathbb{Z},\ i \in \mathcal{I}. \end{array}\right\} \tag{P}$$

Here, the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and constraint functions $g_i : \mathbb{R}^n \to \mathbb{R},\ i \in M$ are convex, set $\mathcal{I}$ is the index set of integer constrained decision variables and set M is the index set of constraints. These problems are called convex mixed-integer nonlinear programs, or convex MINLPs in short.

An example of a convex MINLP is the uncapacitated facility location (UFL) problem in the area of supply chain management. In a UFL problem, we are given a set of candidate facilities $\mathcal{F} = \{1, \ldots, m\}$ and a set of customers $C = \{1, \ldots, n\}$ whose demand for a single commodity must be satisfied from open facilities. The objective is to decide which facilities should be opened and how much of each customer's demand be satisfied by an open facility while minimizing the total cost. The cost comprises of a fixed cost associated with opening a facility and a transportation cost incurred in transporting the commodity from an open facility to a customer. A quadratic case of the UFL problem is

modeled as

$$\left.\begin{array}{ll} \underset{x,\, z}{\text{minimize}} & \sum_{i\in\mathcal{F}} c_i z_i + \sum_{i\in\mathcal{F}, j\in C} t_{ij} x_{ij}^2 \\ \text{subject to} & 0 \le x_{ij} \le z_i,\ i \in \mathcal{F},\ j \in C, \\ & \sum_{i\in\mathcal{F}} x_{ij} = 1,\ j \in C, \\ & x_{ij} \ge 0,\ i \in \mathcal{F},\ j \in C, \\ & z_i \in \{0, 1\},\ i \in \mathcal{F}, \end{array}\right\} \tag{UFL}$$

where c and t are given cost parameters. Decision variables $z_i, i \in \mathcal{F}$ are binary, $z_i = 1$ means facility i is opened, and $z_i = 0$ means it is not opened. Variable x_{ij} is the fraction of demand of customer j that is met from facility $i \in \mathcal{F}$. The first component of the objective function is the fixed cost of opening facilities. If a facility i is opened, a fixed cost c_i is incurred. The second term in the objective models the transportation cost. If a quantity is transferred from the facility i to the customer j, transportation cost t_{ij} proportional to the squared of the quantity is incurred. The objective function is a convex nonlinear function. Constraints can be rewritten in the form $g_i(x) \le 0$ with g_i affine (and hence convex) and variables z are integer constrained. The constraint $x_{ij} \le z_i$ ensures that no demand is served by a closed facility.

Convex MINLPs arise in a wide range of real-world applications. Integrality constraints can model the requirement of selection from a discrete set, presence or absence of an entity, logical relationships, fixed charges, disjunctions, piecewise linear functions, non-divisibility of resources, etc. For example, whether to open a facility is modeled using integer variables z in the (UFL) problem. Nonlinearity arises from relationships between the variables and constraints or objective. For example, nonlinear objective function models operational cost in the (UFL) problem. Similarly, a model may have nonlinear relationships, for example, potential loss or gain in an investment over a specified time in financial applications, covariance in data science, gas rate and pressure dependent critical velocity in shale gas production, thermal expansion, heat flow, stress, voltage, pressure in engineering models. Convex MINLPs are also solved as subproblems in other optimization problems like the more general nonconvex MINLPs (Nowak *et al.* (2018); Lundell and Westerlund (2018)), and to approximately solve mixed-integer PDE constrained optimization problems (Chapter 4).

MINLP models have been applied in economics (Kendrick (1982)), process and facility design (Ravemark and Rippin (1998); Iribarren *et al.* (2004)), service system design (Elhedhli (2006); Ahmadi-Javid and Hoseinpour (2019)), medical science (Cao and Lim (2010)), genetics (Mullin and Belotti (2016)), block layout design in manufacturing and service organization (Castillo *et al.* (2005)), supply chain (Agnetis *et al.* (2012); Gün-

lük *et al.* (2007)), portfolio optimization (Vielma *et al.* (2008)), municipal drinking water networks (Laird *et al.* (2006)), space applications (Schlueter *et al.* (2013)), gas distribution network (Mikolajková *et al.* (2018)), shale gas production (Sharma (2013)), electric power systems (Donde *et al.* (2005); Kaur *et al.* (2014); Pourakbari-Kasmaei *et al.* (2019)), design of nuclear plants (Committee (2010)), disaster recovery scenarios (Donovan and Rideout (2003); Fügenschuh *et al.* (2009); You and Leyffer (2010, 2011); Legg *et al.* (2013)), academia (Duives *et al.* (2013); de Souza and Ritt (2018)), etc.

When the functions f and g_i, $i \in M$ are not required to be convex, the problem (P) is called a MINLP or a general MINLP. MINLPs are theoretically hard problems to solve. A result by Jeroslow (1973) states that for the general class of MINLP with quadratic constraints there can not be any method or algorithm that can solve all these problems using available computing architectures. Moreover, a special class of problems of the general MINLPs called nonlinear programs, obtained by removing integrality constraints, are also hard problems (Murty and Kabadi (1985); Floudas and Pardalos (2013)). Convex MINLPs are somewhat easier to solve than general MINLPs, but are still hard theoretically. A subclass of this problem called mixed-integer linear program, obtained when functions f and g are linear, is known to be NP-hard (Kannan and Monma (1978b)). That means, there is no polynomial-time algorithm for these problems. Nevertheless, tremendous advancements in linear, mixed-integer linear, and convex nonlinear optimization and increasing computing power of modern computers have enabled solvers to solve convex MINLPs. Further advancements are required to solve even moderate size convex MINLPs fast.

Even though convex MINLPs are theoretically hard, various useful properties of convex MINLPs can sometimes be exploited to solve them faster. We study such properties and special structures that lead to practically useful enhancements for algorithms for convex MINLPs. This chapter describes the existing methodologies and software for convex MINLPs. Then, we outline the remainder of the thesis and present our contributions.

1.1 Relaxations of Convex MINLPs

Given a convex MINLP of the form (P), a vector x satisfying all the constraints is called a *feasible solution*. The set of all feasible solutions is called the *feasible set* or the *feasible region*, and is denoted by P^o. A feasible solution x^* at which the objective function takes the minimum value among all feasible solutions, that is $f(x^*) \leq f(x), \forall x \in P^o$, is called an *optimal solution*, and the value $f(x^*)$ is called the *optimal value*. We denote the optimal value of problem (P) by Z^*.

Given a point $\bar{x} \in \mathbb{R}^n$, we say that a constraint indexed $i \in M$ is *violated* at $\bar{x}$ if the constraint inequality does not hold at $\bar{x}$, that is, $g_i(\bar{x}) > 0$. A constraint i is called *active* (or tight or binding) at $\bar{x}$ if the constraint holds at equality, that is, $g_i(\bar{x}) = 0$. Let $\pi^\top x \leq \pi_0$ be an inequality denoted by (π, π_0) for a given vector $\pi \in \mathbb{R}^n$ and scalar π_0. It is a *valid linear inequality* for the problem (P), if it is satisfied by all points $x \in P^o$. A valid linear inequality for a problem is also called a *cutting plane*. Finding 'good' cutting planes is one of the common ways of improving algorithms, as we describe later.

Relaxations are another important element of optimization algorithms. Desirable relaxations are those that are easy to obtain and solve, and which provide good lower bounds to Z^*. However, these two goals may conflict with each other. A relaxation that offers a higher lower bound may not be easy to obtain. On the other hand, a relaxation obtained by simply dropping integrality constraints or nonlinear constraints may provide weak lower bounds. Trade-off between ease of obtaining and solving a relaxation on the one hand and the lower bound quality on the other is important. Usually, a *relaxation* of an optimization problem is also an optimization problem albeit of a different class that is easier to solve. More formally,

Definition 1.1.1. *Consider an optimization problem, say (R), with objective function $h(x)$ and feasible region denoted by R^o. Problem (R) is a relaxation of problem (P) if it has the following two properties.*

1. *$h(x) \leq f(x)$ for all $x \in P^o$, where P^o is the feasible region of (P). That is, $h(x)$ underestimates the objective function $f(x)$ on the set P^o.*

2. *$P^o \subseteq R^o$, that is, every point in the feasible region of problem (P) also lies in the feasible region of its relaxation (R).*

Next we present some of the commonly used relaxations of the convex MINLP (*P*).

1.1.1 Linear Programs

The problem of minimizing a linear function over a region defined by linear constraints (and in which all variables are real) is called a *Linear Programming*. Mathematically a linear program, or an LP, can be written as

$$\begin{aligned} &\underset{x}{\text{minimize}} && a^\top x \\ &\text{subject to} && x \in X, \end{aligned}$$

where the vector $a \in \mathbb{R}^n$ and $X = \{x \in \mathbb{R}^n : Cx \leq c, Dx = d\}$. C and D are matrices of coefficients of linear constraints, and c and d are vectors representing the right hand sides. Clearly, the feasible set of an LP is a polyhedron.

If the convex MINLP (P) has a linear objective function, the simplest possible linear programming relaxation of (P) can be obtained by removing all the nonlinear constraints and integrality restrictions. If the objective function is nonlinear, one can replace the nonlinear objective with its linear underestimator (first-order approximation). Instead of removing the nonlinear constraints altogether, one can also replace them with their linear approximations. An LP relaxation can be tightened by adding more linear valid inequalities at the cost of increasing the size of LP. Large-scale (over ten million variables) LPs can be solved currently in an acceptable time using the existing robust, fast, and reliable solvers (Mittelmann (2019)). The availability of efficient LP solvers encourages the use of LP relaxations in solving more complicated optimization problems including convex MINLPs.

The primary methods for solving an LP are the primal simplex method (Dantzig (1998); Bertsimas and Tsitsiklis (1997)), dual-simplex method (Lemke (1954)) and barrier method (or interior-point method) (Karmarkar (1984)), or some combination of these algorithms. The simplex method utilizes the fact that if the feasible region of an LP has at least one extreme point and the LP has an optimal solution, then a basic feasible solution (BFS), an extreme point, is optimal. Thus, it searches among the extreme points by moving along the edges of the feasible region from one BFS to the next. The method terminates when it reaches a BFS at which none of the available edges reduces the objective value, and such a BFS is deemed optimal.

The barrier method moves towards the optimal solution starting from the interior of the feasible region. Both barrier and simplex methods have various implementations in state-of-the-art commercial solvers like CPLEX (2017), Gurobi (2012), FICO-Xpress (2009), MOSEK (2004), etc., and open-source software, CLP (2004). Though the simplex method is one of the most efficient practical approaches for solving LPs, it is not a polynomial-time method. That is, its worst-case running time can be exponential in the size of the problem (Klee *et al.* (1972)). However, interior-point methods are polynomial-time and have theoretically better running time. In general, interior-point and dual simplex methods have been shown to perform better than primal simplex methods on large LPs; however, there are instances on which one algorithm outperforms the others (Junior and Lins (2005); Koberstein (2005)).

Another useful feature of LPs is their warm start capability. Warm starting or re-optimizing signifies using the information obtained by solving an LP to efficiently solve the next LP, which is often a minor perturbation of the earlier problem. These perturbations occur in the context of branch-and-cut frameworks on which most of the convex MINLP algorithms are based. The perturbed LP comprises of either a change in bounds of

variables or constraints or some additional constraints to the already solved LP. The dual simplex methods are particularly suitable for re-optimizing subproblems in the branch-and-cut framework. A solution to the last LP stays dual feasible and can be used directly as a starting point for the next LP (Kostina (2002); Maros (2003); Koberstein (2005)).

1.1.2 Nonlinear Programs

A *nonlinear program* (NLP) is a problem of the form

$$\left.\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & g_i(x) \leq 0,\ i \in M. \end{array}\right\} \tag{NLP}$$

General NLPs are challenging to solve and are considered NP-hard (Murty and Kabadi (1985)), sometimes even intractable (that is, little is known about how difficult they are to solve). However, when the nonlinear functions in the objective and constraints are convex, the resulting problem is called a convex NLP, and there are efficient algorithms for solving them (Vavasis (1991); Nesterov and Nemirovskii (1994); Leyffer and Mahajan (2010)).

State-of-the-art methods for solving convex NLPs include active-set based methods (Forsgren *et al.* (2015); Gill and Wong (2015)), augmented Lagrangian method (Friedlander and Leyffer (2008)), interior point method (Wächter *et al.* (2002); Forsgren *et al.* (2002)), sequential quadratic programming method Gill and Wong (2012); Nocedal and Wright (2000)), generalized reduced gradient method (Lasdon *et al.* (1974)), etc. These methods are implemented in state-of-the-art solvers like CONOPT (2007), filterSQP (Fletcher and Leyffer (1998)), IPOPT (2015), KNITRO (2012), SNOPT (2008). When the NLP is not convex, these methods merely converge to a critical point.

An NLP relaxation of (P) can be obtained by removing integrality constraints on its variables x. It is also commonly referred to as the continuous relaxation of (P).

1.1.3 Mixed-Integer Linear Programs

A *mixed-integer linear program*, or an MILP, is a problem of the form

$$\left.\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & x \in X, \\ & x_j \in \mathbb{Z},\ j \in \mathcal{I}, \end{array}\right\} \tag{MILP}$$

where f is a linear function and X is defined, as earlier, as a set of linear constraints that are to be satisfied by variables x, that is, $X = \{x \in \mathbb{R}^n : Cx \leq c, Dx = d\}$ with C and D be matrices , and c and d be vectors of appropriate dimensions.

Approaches for solving MILPs include LP based branch-and-bound (Land and Doig (2010)), cutting-planes (Gomory (1960)), and branch-and-cut (Caprara and Fischetti (1997); Mitchell (2010)). In LP based branch-and-bound method, LP subproblems are solved in a branch-and-bound framework. Branch-and-cut methods are based on the branch-and-bound framework, and add valid inequalities to subproblems before branching. Cuts are added to strengthen the relaxations. The decision of whether to add cuts has a significant impact on the success of branch-and-cut methods. Typically, several rounds of cuts are added early on (especially at the root node) in the tree than the tree's deeper parts. Cutting-plane methods start from a linear relaxation of (MILP). If the LP optimal solution satisfies all the constraints of the original problem, then the method terminates. Otherwise, it generates a valid inequality violated by the LP optimal solution, thus, strengthening the relaxation, and the process repeats. Cutting-plane methods differ in the type of cuts they use, like the mixed-integer Gomory cuts (Gomory (1963)), intersection cuts (Balas (1971)), lift-and-project cuts (Balas *et al.* (1993)), cover inequalities (Gu *et al.* (1999)), etc.

Some of the state-of-the-art solvers for MILPs are CBC (2004), CPLEX (2017), MINTO, MOSEK (2004), SYMPHONY (Ralphs and Ladányi (2000)), Gurobi (2012), FICO-Xpress (2009), SCIP (Achterberg (2009)), etc. Though MILPs are NP-hard (Kannan and Monma (1978b)), tremendous advancements have taken place in the area of MILPs over the last 60 years. The powerful modeling capability of MILPs allows the mathematical formulation of problems from various applications in many diverse areas. The success of MILP solvers is attributed to its essential components like efficient LP solvers, primal heuristics, cut generation and management routines, and presolving techniques (Lodi (2010); Achterberg and Wunderling (2013); Achterberg *et al.* (2020)).

One can obtain an MILP relaxation of (P) by either dropping all the nonlinear constraints or by replacing them with their linear approximations and by replacing the nonlinear objective function with its first-order approximation.

The areas of LPs, convex NLPs, and MILPs have witnessed tremendous advancements in their theory and software in the last few decades (Leyffer and Mahajan (2010); Jablonskỳ *et al.* (2015); Mittelmann (2017)). In the next section, we will see that solving convex MILPs involves solving a sequence of relaxations, often a large number of times. Thus, advancements in solving these relaxations contribute to solving convex MINLPs.

1.2 Methods for Convex MINLPs

Like MILPs, MINLPs also require a tree-search to resolve the integrality restrictions. Deterministic methods for solving convex MINLPs can be classified into two broad categories: single-tree and multi-tree approaches. Single-tree methods maintain a single branch-and-bound tree, whereas a separate tree for each subproblem is created and solved in multi-tree methods. The nonlinear branch-and-bound (NLP BnB) and the LP/NLP based branch-and-bound (QG) methods belong to the single-tree category, while the generalized Benders decomposition (GBD), the outer approximation (OA), and the extended cutting planes (ECP) methods lie in the second category.

All these methods are iterative and generate lower and upper bounds on the optimal value Z^*. A method terminates when lower and upper bounds fall within an acceptable tolerance. At every iteration, an easier subproblem is solved. As solving progresses, a non-decreasing sequence of lower bounds and a non-increasing sequence of upper bounds are obtained. All these methods are deterministic in the sense that under some assumptions on the problem input, these methods terminate in a finite number of iterations. These assumptions are presented next. For ease of presentation, let X denotes the feasible set formed by all constraints of (P) that are linear, that is,

$$X := \{x : g_i(x) \leq 0,\ g_i \text{ is affine}\}. \tag{1.1}$$

Assumption 1.2.1. *Consider a MINLP problem of the form (P).*

(a) The polyhedral set X is bounded.

(b) The nonlinear functions f and $g_i, i \in M$, are convex and twice continuously differentiable on X.

(c) A constraint qualification holds at every point in the convex hull of the feasible region of (P).

The last Assumption 1.2.1 (c) ensures that the feasible region around any feasible point is well represented by the linear approximations of the nonlinear constraint functions. This assumption is required to guarantee the convergence of the NLP solvers.

1.2.1 Generalized Benders Decomposition (GBD)

The GBD method by Geoffrion (1972) solves a given problem by projecting it onto the space of some of its variables, called 'complicating variables'. For convex MINLPs, integer constrained variables are considered complicating variables because we get a

tractable convex NLP on fixing them to any values. Let the vector of integer constrained variables in (P) be denoted by x_I. The projected problem in the space of x_I variables, expressed using nonlinear duality theory, has infinitely many linear constraints in its description, making it challenging to directly work with it. The GBD algorithm addresses this problem by solving a relaxed version of the problem. More constraints are added sequentially when required. Outer approximation (OA) and LP/NLP based branch-and-bound (QG) can be viewed as refinements of the GBD.

Let x^k denotes an optimal solution of the relaxation problem at the k^{th} iteration of GBD. The starting relaxation (at $k = 0$) is created using the solution of the continuous NLP relaxation of the original problem. If the relaxation solution value lies within an acceptable tolerance of the best known upper bound, the algorithm terminates. Otherwise, its optimal value is a lower bound to Z^*. It then fixes the integer variables $x_I = x_I^k$, and solves the resulting convex NLP, referred to as a 'fixed' NLP. If the fixed-NLP is optimal and has an optimal value better than the incumbent (a feasible point with the best objective value obtained so far), it updates the incumbent and the upper bound. Whether optimal or infeasible, dual multipliers associated with the solution of a fixed-NLP help refine the relaxation problem as explained in the Algorithm 1. The notion of a fixed-NLP is also used in the OA and the QG algorithms. A fixed-NLP is a nonlinear optimization problem obtained by fixing $x_I = x_I^k$ in (P), that is,

$$\left.\begin{array}{lll} \underset{x}{\text{minimize}} & f(x) & \\ \text{subject to} & g_i(x) \leq 0, & i \in M, \\ & x_I = x_I^k. & \end{array}\right\} \qquad \text{(FNLP}(x_I^k)\text{)}$$

When a fixed-NLP (FNLP(x_I^k)) is infeasible, most NLP solvers return a solution of a 'feasibility problem' that aims to find a feasible point by minimizing violation of the constraints (Fletcher and Leyffer (1994)). One such formulation is

$$\left.\begin{array}{ll} \underset{x}{\text{minimize}} & \sum_{i \in M} w_i g_i^+(x) \\ \text{subject to} & x_I = x_I^k. \end{array}\right\} \qquad \text{(FP}(x_I^k)\text{)}$$

Here, $g_i^+(x) = \max\{g_i(x), 0\}$ and $w_i, i \in M$ are non-negative weights chosen as parameters. FP(x_I^k) is feasible regardless of x_I^k and its optimal value is bounded below by zero. FNLP(x_I^k) is infeasible if and only if the optimal value of FP(x_I^k) is strictly positive.

Let x_R be the vector of variables that are not integer constrained, and $x = (x_R, x_I)$. Also, let $X_R \subseteq \mathbb{R}^{n-|I|}$ and $X_I \subseteq \mathbb{Z}^{|I|}$ be the sets of linear inequalities of (P) that define bounds on variables x_I and x_R, respectively. Since set X is assumed to be bounded, X_I and X_R are

also bounded. Also, The projection of (P) on the space of x_I variables is given by

$$\left.\begin{array}{ll} \underset{x_I}{\text{minimize}} & v(x_I) \\ \text{subject to} & x_I \in V \cap X_I, \end{array}\right\} \qquad (\text{Proj}(x_I))$$

where the set V is defined as

$$V = \{x_I^k : \text{there exist some } x \in X \text{ such that } x_I = x_I^k, g_i(x) \leq 0, i \in M\}.$$

Thus, V is a set of all feasible integer values of x_I. A dual representation of set V is given by the following result.

Theorem 1.2.2 (Geoffrion (1972))**.** *A point x_I for some $x \in X$ lies in the set V if and only if it satisfies the following infinite system*

$$\underset{x_R}{\textit{minimize}} \;\; \overline{L}(x_R, x_I, \overline{\lambda}) \leq 0, \forall \overline{\lambda} \in \Lambda, \qquad (1.2)$$

where $\overline{L}(x_R, x_I, \overline{\lambda}) = \sum_{i \in M} \overline{\lambda}_i g_i(x_R, x_I)$ *and* $\Lambda = \{\overline{\lambda} \in \mathbb{R}^{|M|} | \overline{\lambda} \geq 0, \sum_{i \in M} \overline{\lambda}_i = 1\}$.

Theorem 1.2.3 (Geoffrion (1972))**.** *(i) If x^* is optimal to (P), then x_I^* is optimal to (Proj(x_I)).*

(ii) If (P) is infeasible or unbounded, then so is the projected problem (Proj(x_I)).

In (Proj(x_I)), $v(x_I)$ is parametric in variables x_I and is given by

$$\left.\begin{array}{ll} \underset{x_R}{\text{minimize}} & f(x_R, x_I) \\ \text{subject to} & g_i(x_R, x_I) \leq 0, \; i \in M. \end{array}\right\} \qquad (v(x_I))$$

Using dual representation of ($v(x_I)$) and V in terms of Lagrange function and multipliers, the projected problem (Proj(x_I)) can be rewritten as

$$\left.\begin{array}{ll} \underset{x_I \in X_I,\, \mu_B}{\text{minimize}} & \mu_B \\ \text{subject to} & \text{minimize}_{x_R} \;\; L(x_R, x_I, \lambda) \leq \mu_B, \forall \; \lambda \geq 0, \\ & \text{minimize}_{x_R} \;\; \overline{L}(x_R, x_I, \overline{\lambda}) \leq 0, \forall \overline{\lambda} \in \Lambda. \end{array}\right\} \qquad (\text{GBDM})$$

Here, $L(x_R, x_I, \lambda) = f(x_R, x_I) + \sum_{i \in M} \lambda_i (g_i(x_R, x_I) - 0)$ and λ is a vector of dual multipliers. The first set of constraints are associated with $v(x_I)$ and the other constraints define the set V. The problem (GBDM) is called the master problem. Functions ξ and $\overline{\xi}$ are called support functions and defined as

$$\xi(x_I, \lambda) = \underset{x_R}{\text{minimize}} \;\; L(x_R, x_I, \lambda), \qquad (1.3)$$

$$\overline{\xi}(x_I, \overline{\lambda}) = \underset{x_R}{\text{minimize}} \;\; \overline{L}(x_R, x_I, \overline{\lambda}). \qquad (1.4)$$

Finding support functions $\xi(x_I, \lambda^k)$ and $\overline{\xi}(x_I, \overline{\lambda}^k)$ for some λ^k and $\overline{\lambda}^k$ is difficult in general. However, for problems with specific structures support functions are easy to find. For example, problems in which nonlinear functions are separable in real and integer variables and linear in integer variables, support functions are linear in x_I and (GBDM) becomes an MILP with an infinite number of constraints. Also, for general convex MINLPs, one can generate linear approximations to the support functions and still get an MILP master problem. As the master problem (GBDM) contains a infinite number of constraints, it cannot be solved directly. One solves its relaxation (RGBDM) and updates it dynamically at every iteration k.

$$\left.\begin{aligned} \underset{x_I \in X_I,\ \mu_B}{\text{minimize}} \quad & \mu_B, \\ \text{subject to} \quad & \xi(x_I, \lambda^i) \leq \mu_B,\ i \in T^k, \\ & \overline{\xi}(x_I, \overline{\lambda}^i) \leq 0,\ i \in S^k, \end{aligned}\right\} \tag{RGBDM}$$

where T^k and S^k are finite subsets of index of constraints in (GBDM) that are generated upto iteration k. Algorithm 1 presents the pseudocode for the GBD method. The initial vector x_I^1 can be any point in the set X_I. Commonly, the solution of continuous relaxation, (NLP), of the original problem is considered as the starting point.

Algorithm 1: Pseudocode of the Generalized Benders Decomposition Method.

Input: (P), x_I^1 be any point in X_I, upper bound $UB = +\infty$, lower bound $LB = -\infty$, iteration counter $k = 1, T^k = S^k = \emptyset$, and tolerance $\epsilon > 0$.

while $UB - LB > \epsilon$. **do**
  Solve the fixed-NLP (FNLP(x_I^k)).
  if *FNLP(x_I^k) is feasible* **then**
    Let $\bar{x}^k$ be its an optimal solution. Update $UB = \min\{UB, f(\bar{x}^k)\}$.
    Stop if $UB - LB \leq \epsilon$.
    Let λ^k be a vector of optimal multipliers. Generate the support function by solving (1.3) and set $T^k = T^{k-1} \cup \{k\}$.
  else if *FNLP(x_I^k) is infeasible* **then**
    Let $\overline{\lambda}^k$ be multipliers from solving the feasibility problem (FP(x_I^k)).
    Generate the support function by solving (1.4) and set $S^k = S^{k-1} \cup \{k\}$.
  Solve the relaxed master problem (RGBDM). Let its solution be x_I^{k+1}.
  Set $k = k + 1$ and $LB = \mu_B$.
end

Convergence of the GBD method is ensured in a finite number of iterations for any $\epsilon \geq 0$ if Assumption 1.2.1 holds (Geoffrion (1972); Floudas (1995)). Different variants of GBD method are discussed by Floudas (1995). Also, Su *et al.* (2015) present strategies to computationally accelerate the GBD method by adding multiple cuts in an iteration, and Lee *et al.* (2020) employ machine learning tools to manage the size of the master problem by regulating the cuts added to the master problem.

1.2.2 Outer Approximation (OA)

The OA method, in principle, is similar to the GBD method. It also uses the same mathematical concepts of projection, relaxation, and first-order approximation of nonlinear functions. Like GBD, it also projects the original problem on the space of integer constrained variables and solves an alternating sequence of MILP relaxation and a convex NLP. The two methods, however, differ in the way the MILP relaxations are formed. The OA method utilizes the solution obtained from a fixed-NLP (or the feasibility problem) to generate gradient inequalities to the nonlinear constraints. Given a convex differentiable nonlinear function $f : \mathbb{R}^n \to \mathbb{R}$ and a point $\bar{x} \in \mathbb{R}^n$, the following well known gradient inequality (Rockafellar (1970))

$$\nabla f(\bar{x})^\top (x - \bar{x}) + f(\bar{x}) \leq f(x) \tag{1.5}$$

holds for all $x \in \mathbb{R}^n$. A pictorial representation of gradient inequalities to a convex set defined by a single nonlinear inequality is shown in the Figure 1.1.

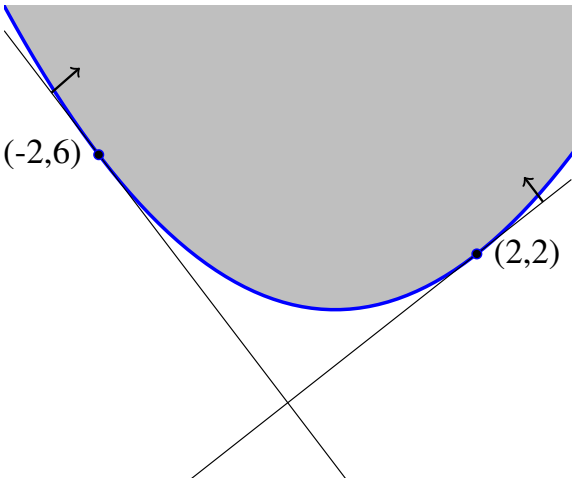


Figure 1.1: A pictorial representation of gradient inequalities (also known as outer-approximation cuts) to the convex set $\{x \in \mathbb{R}^2 : x_1^2 - x_1 - x_2 \leq 0\}$ at points $(-2, 6)$ (left linear inequality) and $(2, 2)$ (right linear inequality).

The OA method proposed by Duran and Grossmann (1986) creates an MILP relaxation of (P) using the outer approximation cuts of the form (1.5) generated at selected points. The points are, in turn, obtained by solving fixed-NLPs (obtained by fixing variables x_I to their values in the solution of the MILP relaxation). This method assumes that the nonlinear functions are separable in integer and real variables, and linear in integer constrained variables x_I. Fletcher and Leyffer (1994) generalized the idea to convex MINLPs under Assumption 1.2.1.

Lemma 1.2.4 (Fletcher and Leyffer (1994))**.** *If the fixed-NLP (FNLP(x_I^k)) is infeasible, and x^k solves FP(x_I^k) with $\sum_{i\in M} w_i^k g_i^+(x^k) > 0$, then $x_I = x_I^k$ is infeasible in constraints $g_i(x^k) + (x - x^k)^\top \nabla g_i(x^k) \leq 0, \forall i \in M, \forall x \in X$.*

Suppose FNLP(x_I^k) is infeasible and $\bar{x}^k$ is a solution to FP(x_I^k), then any point $x \in X$ with $x_I = \bar{x}_I^k$ violates the outer approximations to nonlinear constraints at $\bar{x}^k$. Thus, it is used for characterizing feasible region of the projected problem. Let T (S) be the index set of points in X such that the corresponding fixed-NLPs are feasible (infeasible), defined as

$$\begin{aligned} T &= \{k|\ \text{FNLP}(x_I^k) \text{ is feasible and } \bar{x}^k \text{ is an optimal to FNLP}(x_I^k)\}, \\ S &= \{k|\ \text{FNLP}(x_I^k) \text{ is infeasible and } \bar{x}^k \text{ is an optimal solution to FP}(x_I^k)\}. \end{aligned}$$

An MILP equivalent of the problem (Proj(x_I)) can then be written as

$$\left.\begin{array}{lll} \underset{x,\ \mu_{OA}}{\text{minimize}} & \mu_{OA} & \\ \text{subject to} & f(\bar{x}^k) + (x - \bar{x}^k)^\top \nabla f(\bar{x}^k) \leq \mu_{OA}, & k \in T, \\ & g_i(\bar{x}^k) + (x - \bar{x}^k)^\top \nabla g_i(\bar{x}^k) \leq 0, & i \in M,\ k \in T, \\ & g_i(\bar{x}^k) + (x - \bar{x}^k)^\top \nabla g_i(\bar{x}^k) \leq 0, & i \in M,\ k \in S, \\ & x \in X, x_i \in \mathbb{Z}, & i \in \mathcal{I}. \end{array}\right\} \quad \text{(OAM)}$$

Theorem 1.2.5. *If Assumption 1.2.1 holds, then the problems (OAM) and (P) have the same optimal value. Also, every optimal solution x^* to (P) solves (OAM), but the converse is not true. However, if (x^*, μ^*) is an optimal solution to (OAM), then x_I^* is optimal to (P).*

The proof of Theorem 1.2.5 can be shown using results by Bonami *et al.* (2008b) and Fletcher and Leyffer (1994). Again, solving (OAM) is challenging as it contains a large number of constraints. The OA method starts by solving a relaxation of problem (OAM) and tightens it iteratively using the solutions of fixed-NLPs. The algorithm terminates when the upper and lower bounds on Z^* lie within an acceptable tolerance. Outer approximations added to the nonlinear constraints at the solution $\bar{x}^k$ of the feasibility problem

ensure that the integer combination $\bar{x}_I^k$ does not appear again as the solution of any subsequent MILP relaxation. The MILP relaxation of the (OAM) at any iteration i is given by

$$\left.\begin{aligned} \underset{x,\,\mu_{OA}}{\text{minimize}} \quad & \mu_{OA} \\ \text{subject to} \quad & f(\bar{x}^k) + (x - \bar{x}^k)^\top \nabla f(\bar{x}^k) \leq \mu_{OA},\ k \in T^i, \\ & g_j(\bar{x}^k) + (x - \bar{x}^k)^\top \nabla g_j(\bar{x}^k) \leq 0,\ j \in M,\ k \in T^i, \\ & g_j(\bar{x}^k) + (x - \bar{x}^k)^\top \nabla g_j(\bar{x}^k) \leq 0,\ j \in M,\ k \in S^i, \\ & x \in X, x_j \in \mathbb{Z}, j \in \mathcal{I}, \end{aligned}\right\} \tag{ROAM}$$

where the sets $T^i \subset T$ and $S^i \subset S$ are defined as

$$T^i = \{k|\ k \leq i\ ,\ \text{FNLP}(x_I^k) \text{ is feasible and } \bar{x}^k \text{ is an optimal to FNLP}(x_I^k)\},$$
$$S^i = \{k|\ k \leq i\ ,\ \text{FNLP}(x_I^k) \text{ is infeasible and } \bar{x}^k \text{ is an optimal solution to FP}(x_I^k)\}.$$

Algorithm 2 presents the pseudocode for the OA method. It has been shown by Duran

Algorithm 2: Pseudocode of the Outer Approximation Method.

Input: (P), let x_I^1 be any integer point in X, lower bound $LB = -\infty$, upper bound $UB = +\infty$, counter $k = 1$, $T^k = \emptyset$, $S^k = \emptyset$, and tolerance $\epsilon > 0$.

```
while UB − LB > ε. do
    Solve the fixed-NLP (FNLP(x_I^k)).
    if FNLP(x_I^k) is feasible then
        Let x̄^k be an optimal solution. Linearize nonlinear objective and
          constraints at x̄^k, update UB = min{UB, f(x^k)} and T^k = T^{k−1} ∪ {k}.
    else if FNLP(x_I^k) is infeasible then
        Let x̄^k be an optimal solution to FP(x_I^k). Linearize nonlinear constraints
          at x̄^k and update S^k = S^{k−1} ∪ {k}.
    Solve the update master problem (ROAM), let its solution be x^k.
    Set k = k + 1 and LB = μ_OA.
end
```

and Grossmann (1986) and Fletcher and Leyffer (1994) that the OA method terminates in a finite number of iterations if the number of possible integer combinations in the given problem is finite. The subtlety of the proof is that when X is bounded, no integer assignment, feasible or infeasible, repeats due to the outer approximation cuts at solutions to FP(x_I^k). Duran and Grossmann (1986) also show that for problems in which nonlinear functions are separable in integer and real variables and linear in integer variables, the

lower bound given by the OA method at any iteration is at least as good as the one from the GBD method. This property implies that for these problems, OA may terminate in fewer iterations than GBD. However, it does not mean that OA will converge faster than GBD, since the latter solves a smaller relaxation than OA at each iteration.

1.2.3 Extended Cutting Plane (ECP)

The ECP method by Westerlund and Pettersson (1995) is an extension of Kelley's cutting plane method for convex NLPs. It is considered suitable for problems with a moderate degree of nonlinearity. The ECP method starts from an MILP relaxation of the problem (P). This relaxation is updated at every iteration by generating outer approximations to the nonlinear objective and constraints, in the same way as in the OA method, at a solution to the fixed-NLP. The linearizations added at any point $x^k \notin P^o$ ensure that x^k does not appear again as a solution to subsequent MILPs. However, there is no mechanism for preventing the MILP relaxation from yielding a solution with the same integer assignment x_I^k. This drawback of the ECP method sometimes leads to a large number of iterations until termination. The algorithm terminates when the MILP relaxation solution is feasible to the original problem.

In practice, extended cutting planes are easy to generate and are incorporated in other methods in a branch-and-cut framework (Abhishek *et al.* (2006); Sharma *et al.* (2020b)). The convergence of ECP method can be explained similarly to that of the OA method, but it converges relatively slower. In convex MINLPs where all the variables are integer constrained, OA and ECP methods perform identically. In the worst case, all the three multi-tree methods can take an exponential amount of time (in the size of the problem), as shown by Hijazi *et al.* (2014).

1.2.4 Nonlinear Branch-and-Bound (NLP BnB)

The branch-and-bound based method by Dakin (1965) and Gupta and Ravindran (1985) is one of the earliest approaches for solving integer constrained optimization problems. The idea of a branch-and-bound method is to iteratively divide the feasible region of the original problem into smaller subsets and then optimize over these sets. A problem corresponding to a subset is called a subproblem. The process of dividing is referred to as branching. The subproblems are iteratively generated and solved to obtain lower bounds on the optimal solution value of the original problem. If a solution to any of the subproblems is also feasible to the original problem, it updates the upper bound on the optimal value. All the subproblems with a lower bound value greater than or equal to the best known upper bound value are excluded from further exploration. This exclusion is

referred to as 'pruning'. The method terminates when no further branching is possible for any of the subproblems.

Branch-and-bound can be visualized in the form of a single tree. In this framework, each subproblem is termed as a node. The initial relaxation obtained from the original problem is termed as the root node. When a subproblem is branched into k disjoint subproblems, each subproblem is called a child node, and the subproblem that is branched upon is called the parent node. A node that has no children is called a leaf node. When the lower bound at a node exceeds or becomes equal to the upper bound value, it is excluded from further branching. This exclusion is referred to as pruning by bound. A node is also pruned when it becomes infeasible, in which case it is called pruning by infeasibility.

The NLP BnB method starts with (NLP), the continuous relaxation of the problem (P) obtained by dropping integrality restriction. Every subproblem in this method is a convex NLP. At iteration k, if an optimal solution x^k at a node does not satisfy the integrality restrictions on x_I variables, an integer variable x_j^k, $j \in \mathcal{I}$, that has assumed a nonintegral value, is selected and the node is split into two subproblems using the inequalities $x_j \leq \lfloor x_j \rfloor$ and $x_j \geq \lceil x_j \rceil$. This kind of branching done using an integer constrained variable is called variable branching. Solutions to nonlinear subproblems give the lower bounds, and an integer feasible solution at any node provides an upper bound. The performance of branch-and-bound based approaches relies mainly on branching methods and the order in which nodes are solved (the latter is termed as a node selection rule).

Some examples of variable branching are strong branching, pseudocost branching, and reliability branching (Achterberg *et al.* (2005); Linderoth and Savelsbergh (1999)). The preferable variable branching rule in state-of-the-art solvers is reliability branching. Another important component of branch-and-bound algorithms, the node selection strategy, helps in deciding which node to solve next. These rules aim at improving the lower bound faster and finding good quality upper bounds early in the search. Usually, while choosing a node selection rule, there exist trade-offs between the ease of implementation, the possibility of solving related problems in succession (to exploit warm starting), quality of bounds obtained, memory requirement, etc. Popular node selection strategies are depth-first search, best-bound search, hybrid search, etc. Study and comparison of various node selection rules are discussed in Dakin (1965), Ibaraki (1976), and Linderoth and Savelsbergh (1999). The boundedness of the set X in the Assumption 1.2.1 ensures that the number of possible subproblems in the branch-and-bound (search) tree remains finite, and thus, the algorithm converges in a finite number of iterations.

More details of various algorithmic decisions in a branch-and-bound method can be found in Gupta and Ravindran (1985) and Bonami *et al.*

(2011). Algorithm 3 shows the pseudocode for the NLP BnB method.

Algorithm 3: Pseudocode for the Nonlinear Branch and Bound Method.

Input: Problem (P), the list of subproblems to be solved, $\mathcal{L} = \{(\text{NLP})\}$, upper bound $UB = +\infty$, iteration counter $k = 0$.

```
while L ≠ ∅ do
    Select a problem p_k ∈ L, update L = L\p_k, and solve p_k.
    if p_k is infeasible then
        Prune the node by infeasibility.
    else
        Let x^k be an optimal solution.
        if f(x^k) > UB then
            Prune the node by bound.
        else if x^k is integer feasible then
            Update UB = min{UB, f(x^k)}.
        else
            Branch p_k into subproblems p_{k1} and p_{k2} and update
              L = L ∪ p_{k1} ∪ p_{k2}.
        end
    end
end
```

1.2.5 LP/NLP Based Branch-and-Bound (QG)

This method is proposed by Quesada and Grossmann (1992) and can be seen as a single-tree implementation of the OA method. The fundamental idea of this method is to avoid solving a large number of related MILPs in different iterations from scratch, like in GBD and OA methods. Instead, it tries to merge the changes in subsequent MILPs in a single branch-and-bound tree. Like OA, it aims to solve the MILP equivalent (OAM) of the original problem; however, it creates and maintains a single branch-and-cut tree. It starts from a relaxation of (OAM) and proceeds like LP based branch-and-bound while dynamically updating the MILP relaxation at nodes yielding an integer solution. When a node in the tree yields an integer optimal solution x^k, it solves the fixed-NLP (FNLP(x_I^k)). The solution of FNLP(x_I^k) or FP(x_I^k) is used to generate appropriate linearizations as in OA which are added to all the open nodes of the tree to tighten the relaxations, and branch-and-cut is resumed. The root node can be formed by solving the continuous relaxation of the problem and linearizing the nonlinear objective and constraints at its optimal solution. This MILP relaxation is then relaxed by dropping integrality restriction on variables x_I.

The resulting LP thus forms the root LP relaxation, P^0. A pseudocode of the QG algorithm is presented in Algorithm 4 and Figure 1.2 shows a pictorial depiction of this algorithm.

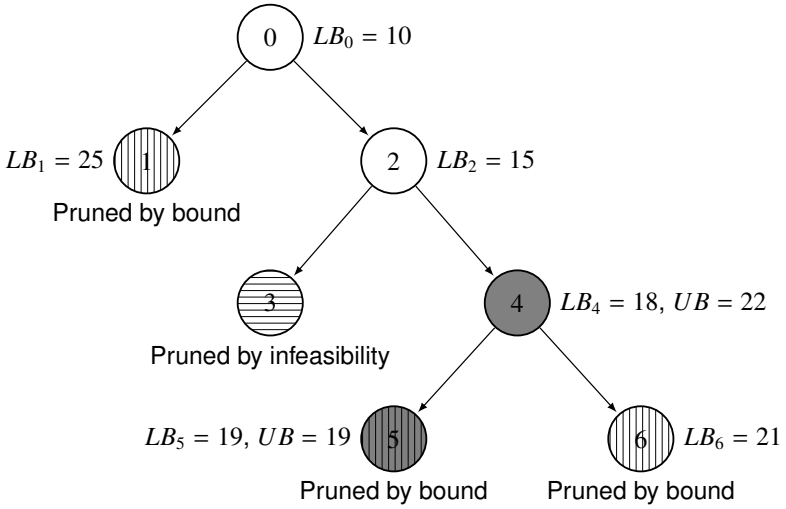


Figure 1.2: An illustration of the QG method. In the given branch-and-bound tree, the search starts from root node 0 that yields a fractional solution with the lower bound of 10, and is branched into nodes 1 and 2. Node 2 is selected next for processing and is further divided into nodes 3 and 4. LB_i indicates the lower bound at node i and UB is the upper bound on the optimal value Z^* of the original problem. Node 3 becomes infeasible and hence pruned by infeasibility. Filled nodes indicate nodes at which associated LPs yield an integer solution, and thus, a fixed-NLP is solved, and the upper bound is updated. Hatched nodes are pruned either due to infeasibility (horizontally hatched) or bound (vertically hatched). At node 4, fixed-NLP is solved, and gradient inequalities are generated at the fixed-NLP solution and added to the open nodes 4 and 5. Node 4 is then resolved. Node 1 is pruned by bound as upper bound is updated to 22 at node 4; and node 6 is pruned when fixed-NLP at node 5 has improved the upper bound to 19.

This method combines the strength of NLP BnB and OA methods, and has a finite convergence. The proof of its convergence follows from the convergence results of OA and NLP BnB methods. The QG method has been shown extremely effective in practice and forms the basis of the most efficient convex MINLP solvers (Abhishek *et al.* (2006); Bonami *et al.* (2008b); Kronqvist *et al.* (2018a); Sharma *et al.* (2020b)).

1.3 Heuristics

In the methods based on a branch-and-bound framework, the availability of a good quality feasible solution is critical in deciding the size of the branch-and-bound tree, mainly because good upper bounds help in pruning the nodes. By the size of a tree, we mean the number of solved (or processed) nodes in the branch-and-bound tree. Better solutions

Algorithm 4: Pseudocode for the LP/NLP Based Branch-and-Bound Method.

Input: Problem (P), the list of subproblems to be solved, $\mathcal{L} = \{P^0\}$, upper bound $UB = +\infty$, iteration counter $k = 0$.

```
while L ≠ ∅ do
    Select a problem p_k ∈ L, update L = L\p_k, and solve p_k.
    if p_k is infeasible then
        Prune the node by infeasibility.
    else
        Let x^k be the optimal solution.
        if f(x^k) > UB then
            Prune the node by bound.
        else if x^k is integer feasible then
            Solve the fixed-NLP (FNLP(x_I^k)).
            if FNLP(x_I^k) is feasible then
                Let x̄^k be an optimal solution. Generate linearizations to
                nonlinear objective and constraints at x̄^k and add to all nodes in
                L and to p_k. Update UB = min{UB, f(x̂^k)}.
            else if FNLP(x_I^k) is infeasible then
                Let x̄^k be an optimal solution of the feasibility problem.
                Generate linearizations to nonlinear constraints at x̄^k and add to
                all nodes in L and to p_k.
            Add p_k back to list L and set K = k + 1.
        else
            Branch p_k into subproblems p_k1 and p_k2 and update
            L = L ∪ p_k1 ∪ p_k2.
        end
    end
end
```

also help tighten constraints and variables bounds and also possibly fixing some of the variables. Smaller branch-and-bound trees lead to shorter computing time and memory requirements.

Heuristics are methods that aim to obtain good quality feasible solutions in a short time. These approaches are prevalent and extensively used in MILP and MINLP solving. Many ideas from the MILP literature on heuristics have been extended to MINLPs.

Two broad classes of heuristics for mixed-integer nonlinear (and linear) programming are primal heuristics and improvement heuristics.

Primal heuristics, also referred to as start heuristics, are applied at the nodes in the early part of the tree (mostly at the root node) to obtain a feasible solution. These methods are focused more on obtaining a feasible solution fast than on finding a good quality solution. Some of the famous primal heuristics are feasibility pump (Bonami *et al.* (2009b)), relaxation enforced neighborhood search (Berthold (2012)), undercover (Berthold and Gleixner (2012)), diving heuristics (Berthold (2006); Bonami and Gonçalves (2012)), etc.

The main idea of an improvement heuristic is to start from a solution and iteratively search for a better solution in terms of the objective value. Sometimes, a solution obtained using some primal heuristic serves as an initial point. Some notable improvement heuristics are guided dives (Danna *et al.* (2005b)), crossover (Berthold (2014)), and large neighborhood searches (LNS) that include local branching (Nannicini *et al.* (2008)), relaxation induced neighborhood search (Danna *et al.* (2005b)), etc. LNS methods search for a better solution in some neighborhood of a given solution (referred to as an initial solution or a reference solution). Both the reference point and the neighborhood influence the performance of LNS methods. A neighborhood signifies a limited search space around a given solution. It is defined either by adding some constraints or fixing some variables to solve easier subproblems while aiming for better quality solutions. We later demonstrate the use of ideas from the local branching heuristic in developing an improvement heuristic for an inverse problem presented in Chapter 4.

Fischetti and Lodi (2002) introduced the local branching heuristic for solving MILPs (with binary integer variables) in a branch-and-bound framework. The neighborhood around a given integer solution is defined using a local branching cut, which is a linear constraint, that restricts the distance from the given point in Manhattan norm on the integer variables. Given a solution $\bar{x} \in \{0,1\}^n$, a local branching constraint is given by

$$\sum_{i:\bar{x}_i=0} x_i + \sum_{i:\bar{x}_i=1} (1 - x_i) \leq k, \tag{1.6}$$

where the expression on the left hand side is denoted by $N(\bar{x}, x)$. Equation (1.6) defines a k-neighborhood around the given solution $\bar{x}$ by allowing at most k flips in the values of binary variables. The local branching constraint is used as a branching rule within a branch-and-bound framework. Given a solution $\bar{x}$, one creates two subproblems using the following branching constraints

$$N(\bar{x}, x) \leq k, \qquad \text{(Left branch)}$$

$$N(\bar{x}, x) \geq k + 1, \qquad \text{(Right branch)}$$

where k is some positive integer and is typically in the range $(10, 20)$ (Fischetti and Lodi (2002); Berthold (2014)). The motivation is that the left subproblem associated with k-neighborhood is smaller and therefore, computationally less expensive than the right subproblem and may still contain a better solution than $\bar{x}$. The left subproblem is solved as a standalone problem using may be a separate branch-and-bound tree. If a better solution is indeed obtained by solving the left subproblem, then using the improved solution and local branching constraint, the right subproblem is further divided as shown in Figure 1.3. The heuristic terminates when solving a left subproblem does not provide a better solution and in this case the regular branch-and-bound is resumed. This heuristic has been modified to start from an infeasible solution by Fischetti and Lodi (2008), and Nannicini *et al.* (2008) present an extension of a local branching heuristic for MINLPs.

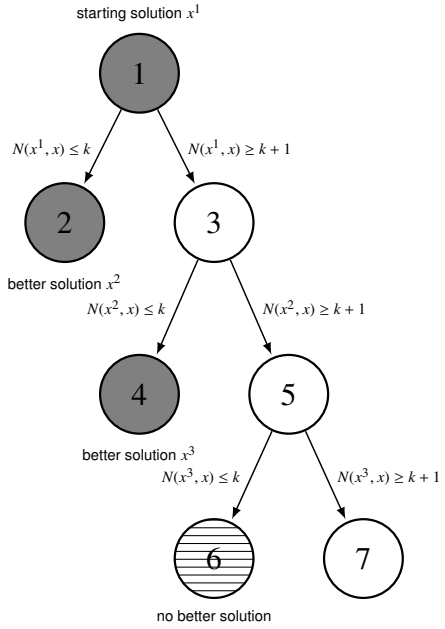


Figure 1.3: A pictorial representation of the local branching scheme. In a given branch-and-bound tree, the local branching has started at node 1 from a given integer feasible solution x^1 and created two nodes 2 and 3. On solving node 2, an improved solution is obtained using which node 3 is further branched into nodes 4 and 5. Similar steps repeat at nodes 4 and 5. As no better solution is found by solving node 6, the local branching scheme terminates, and the original branch-and-bound is resumed at node 7.

For highly challenging problems, as we will see in Chapter 4, many a times, heuristics provide some satisfactory solutions much faster than the state-of-the-art exact algorithms.

1.4 Presolving Techniques

Presolving or preprocessing techniques are those that simplify a given problem by either reformulation, removing redundant constraints, fixing variables, tightening variable bounds and constraint bounds, or similar such modifications before actually solving it. The success of the modern day MILP solvers can be accredited highly to presolving methods. Some popular presolving techniques for MILPs can be found in Savelsbergh (1994), Achterberg (2009), and Mahajan (2010). Many of these techniques are incorporated into solvers for MINLPs too. Presolving helps reduce the size of the problem, tightening the problem/relaxations, generating tighter valid inequalities, and possibly speeding up the convergence to the optimal solution. Many presolving techniques pertaining to MINLPs are presented in Belotti *et al.* (2013).

We briefly explain two impactful presolving techniques in MINLPs, namely, *constraint disaggregation* and *coefficient improvement*. Both these schemes are applicable when a problem has specific structures using which tighter relaxations can be obtained.

Consider the uncapacitated facility location (UFL) problem. The first set of constraints that model the condition that a customer's demand must be satisfied from the facilities that are open is given by

$$0 \leq x_{ij} \leq z_i, \ i \in \mathcal{F}, \ j \in C. \tag{1.7}$$

The form in Equation (1.7) is referred to as a 'disaggregated' form. These constraints can also be written in an 'aggregated' form as

$$\sum_{j \in C} x_{ij} \leq m z_i, \ i \in \mathcal{F}, \tag{1.8}$$

where m is the number of facilities. A (UFL) problem with constraints in aggregated form (instead of disaggregated form) results in a drastically large branch-and-bound tree and solution time (Belotti *et al.* (2013)). This is because the continuous relaxation of a problem having constraints in disaggregated form is tighter than the aggregated form. That is, for any value of variable z_i, set of points satisfying (1.7) is a subset of points satisfying (1.8). Many state-of-art solvers can automatically detect constraints in aggregated form and reformulate it to the disaggregate form (1.7).

Another presolving technique called coefficient tightening in the case of a collection of constraints of the form $\bar{C}$ is described as

$$\left.\begin{aligned} g(x) &\leq M_1(1 - x_1), \\ x_1 &\in \{0, 1\}, x \in \mathbb{R}^n, \end{aligned}\right\} \tag{$\bar{C}$}$$

where $g : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear function, M_1 is some large scalar value. The role of scalar M_1 is to make the nonlinear constraint redundant when $x_1 = 0$ and have the nonlinear constraint effective only when $x_1 = 1$. If the value of scalar M_1 is chosen to be unnecessarily large, it can lead to a poor relaxation and a huge branch-and-bound tree with a large solution time (Belotti *et al.* (2013)). Let the maximum value of the nonlinear constraint for $x_1 = 0$ be given by $g(0)$. If $g(0) < M_1$, then the value of M_1 can be tightened to $g(0)$, which is the best possible value of M_1.

By automatically identifying useful structures in convex MINLPs, a solver's computational performance can be improved significantly. Convexity is the most natural property to exploit in convex MINLPs. In branch-and-bound methods, convexity renders nonlinear relaxations that are convex and can be solved efficiently. In branch-and-cut methods, convexity helps generate polyhedral approximations to the nonlinear functions and the feasible region, resulting in linear or mixed-integer linear programs. In the coming chapters, we will see that further improvements are possible by exploiting more structures and properties if they are present in convex MINLPs. These structures may comprise an individual nonlinear constraint or a collection of linear and nonlinear inequalities. More specifically, these structures help generate tight convex relaxations, better valid inequalities to strengthen polyhedral relaxations, and better bounds on the optimal value of the problem.

1.5 Software

There are several open-source and commercial software available for solving convex MINLPs. These programs, also called solvers, implement algorithms - exact and heuristic methods, preprocessing techniques, cutting planes, branching and node selection rules, and various other components that help solve a problem efficiently. AMPL (Fourer *et al.* (1993)), GAMS (Brooke *et al.* (1992)), AIMMS (Bisschop and Entriken (1993)), Pyomo (Hart *et al.* (2011)), JuMP (Dunning *et al.* (2017)) are some of the widely used languages for modeling optimization problems including convex MINLPs. Solvers for convex MINLPs include AlphaECP (Westerlund and Lundqvist (2001)), ANTIGONE (Misener and Floudas (2014)), BARON (Sahinidis (1996)), BONMIN (Bonami and Lee (2007)), COUENNE (Belotti (2009)), DICOPT (Grossmann *et al.* (2002)), MINOTAUR (Mahajan *et al.* (2020)), Pajarito (Lubin *et al.* (2016)), SCIP (Achterberg (2009)), SHOT (Lundell *et al.* (2018)). Of these, ANTIGONE, BARON, Couenne, and SCIP are designed for global optimization of nonconvex MINLPs. Since solving LPs, NLPs and MILPs (as subproblems) constitutes an integral component of algorithms for convex MINLPs, most

of the convex MINLP solvers require an interface with the solvers for LPs, NLPs, and MILPs. Reviews and comparisons of various solvers for solving MINLPs and convex MINLPs can be found in Bussieck and Pruessner (2003) and Kronqvist *et al.* (2018a).

We have implemented a majority of the methods proposed and discussed in this thesis into the MINOTAUR solver. It is a sophisticated open-source framework for MINLPs and is largely written in C++. MINOTAUR stands for Mixed-Integer Nonlinear Optimization Toolkit: Algorithms, Underestimators, and Relaxations. For convex MINLPs, it provides data structures and basic algorithmic platforms on which a wide range of single-tree and multi-tree methods can be implemented. It also allows access to the nonlinear functions in terms of their computational graphs, providing the flexibility and possibility of exploiting structures of nonlinear functions. It uses the Open-Solver Interface (OSI) library provided by COIN-OR to link to the CLP and CBC solver libraries for solving LPs and MILPs, respectively. There are direct interfaces to solvers like CPLEX for solving LPs and MILPs, BQPD for QPs, and filterSQP and IPOPT for NLPs. Interested readers can refer to Mahajan *et al.* (2020) for more details on various functionalities and features of MINOTAUR.

1.6 Contributions and Outline of the Thesis

The mathematical concepts and notations used throughout are presented in Appendix A. We classify the remainder of the thesis into four chapters.

Chapter 2 focuses on enhancing the performance of QG algorithm for convex MINLPs by creating better relaxations through effective cuts. Recall that the QG algorithm creates an MILP relaxation of the nonlinear feasible region which is solved by branch-and-cut in a single-tree framework. Adding linearizations only when we reach integer feasible points in the branch-and-bound tree may lead to a weak relaxation, and adding too many linearizations early on can slow down the speed. Also, our computational experiments reveal that a very small fraction of the total nodes solved in the branch-and-bound tree yields an integer optimal LP solution. Additionally, in nodes yielding fractional optimal solutions, a large fraction of nonlinear constraints are violated. These observations motivated us to improve the performance of QG algorithm. We make the following contributions.

(i) We propose a set of schemes for tightening the initial LP relaxation at the root node of the branch-and-bound tree. Two of the techniques are specifically applicable to commonly found univariate nonlinear functions and are more effective than other general approaches.

(ii) We provide another set of schemes for generating effective linearization inequalities at other 'selected' nodes that yield a fractional LP solution and have a considerable constraint violation. Two main decisions in the design of these schemes are: (a) whether additional linearization constraints should be added at a given node, and (b) how to determine the points for generating linearization constraints.

(iii) We also study the impact of these schemes in a parallel implementation of the QG method.

In Chapter 3, we present reformulation techniques that exploit specific structures in a convex MINLP (P) to derive tight relaxations. One seeks tight relaxations to obtain better lower bounds on the optimal value of the problem. We address two useful reformulations for convex MINLPs - 'perspective reformulation' (PR), and reformulation using the 'separability' property of nonlinear functions. Our contributions on this front are as follows.

(i) The structures that enable PR are 'on-off' sets, which are in general difficult to find. We present collections of constraints that yield on-off sets in the required form and we automate their identification in MINOTAUR.

(ii) We solve the perspective-reformulated problem in the branch-and-cut framework of QG method using perspective cuts. We develop techniques to automatically find points for generating tight perspective cuts, in different parts of the tree. We also show that, another method, called ϵ-approximation, is naturally applicable to the reformulated problem because of the assumptions on the considered structures.

(iii) Our method for detecting structures amenable to PR also helps in (presolving techniques) variable fixing and finding redundancy in nonlinear constraints.

(iv) We automate the detection of the separability property in nonlinear functions using their computational graphs and solve the reformulated problem using outer-approximation cuts in MINOTAUR.

(v) We present computational results showing that for some instances, reformulation using function separability induces structures amenable to PR, and thus, can be solved much faster.

In Chapter 4, we study a highly challenging class of optimization problems called mixed-integer partial differential equation constrained optimization (MIPDECO). We first

provide some relevant background on the mathematical theory and notations of PDE-constrained optimization and MIPDECO. Then, we discuss a convection-diffusion inverse problem where one wants to determine the number and location of a set of sources by reconciling the differences between measurements and numerical prediction of the concentration. Our contributions on this part are as follows.

(i) We formulate the inverse problem as a large-scale convex MINLP and empirically show that state-of-the-art methods for convex MINLPs fail to give a satisfactory solution to such problems, even in a considerably large amount of time.

(ii) We advance the state-of-the-art in MIPDECO in a number of ways.

  (a) We develop new rounding schemes that take the physics of the problem into account, for example, by preserving the mass of the sources when we move from a relaxation to a rounded solution.

  (b) We apply a simplified version of the trust-region method by Hahn *et al.* (2020), and show that it provides competitive integer solutions.

  (c) We improve the above mentioned trust-region approach by developing a new problem-specific neighborhood that takes the topology of our problem into account, and we use a specialized knapsack problem for the resulting trust-region subproblem. Using this modified trust-region method, we show that one can solve 3-dimensional instances of MIPDECO, that are typically hard to solve, efficiently and in a reasonable amount of time.

(iii) We provide our algorithms coded in Julia (Bezanson *et al.* (2012)) under a permissible open-source license. We also provide the mathematical models of our 2-dimensional instances, coded using the widely used scripting software, AMPL, to promote experimentation with existing MINLP solvers.

Finally in Chapter 5, we discuss the conclusions from our study, and list some future directions of research.

Chapter 2

Linearization Schemes for QG Method

Recall that the QG algorithm creates an MILP relaxation of the nonlinear feasible region which is solved by branch-and-cut. Adding linearizations only when we reach integer feasible points in branch-and-bound tree may lead to a weak relaxation, and adding many of these early on can slow down the speed. While the algorithm is known to take a finite number of steps, careful implementation and control are required for it to be practically useful. Convex MINLPs are known to be NP-hard, and this algorithm, like others, can take a long time to run. We demonstrate effectiveness of some practical ideas that enhance the performance of this algorithm. We propose two sets of schemes - one for tightening the initial LP relaxation at the root node and the other for adding new linearizations later in the branch-and-bound tree. Strategies for generating linearizations based on the change in the lower bound, depth of the nodes in the tree, etc., and using NLP techniques for selecting points for linearizations have previously been proposed by Abhishek (2008) and Kilinç (2011) for use in the FilMINT solver.

There are 374 instances in MINLPLib (Bussieck *et al.* (2003)) that are known to be convex MINLPs. We excluded 40 instances that did not have any nonlinearity (in constraints and objective) or any integer variables after the presolving step in MINOTAUR. We used the remaining 334 instances and refer to them as the TS_l test set in our experiments. Description of these instances is presented in Appendix B. We analyzed performance of default QG in MINOTAUR on 267 instances in test set TS_l which have at least one nonlinear constraint and observed that a large fraction of the nodes processed yield fractional optimal solutions Figure 2.1, many of which also violate a large fraction of nonlinear constraints Figure 2.2. These observations motivated us to add more linearizations at selected nodes. A commonly used technique for creating linear relaxations of convex nonlinear constraints is through a gradient based underestimation. One can thus create a relaxation of (P) by replacing its nonlinear constraints by gradient inequalities of

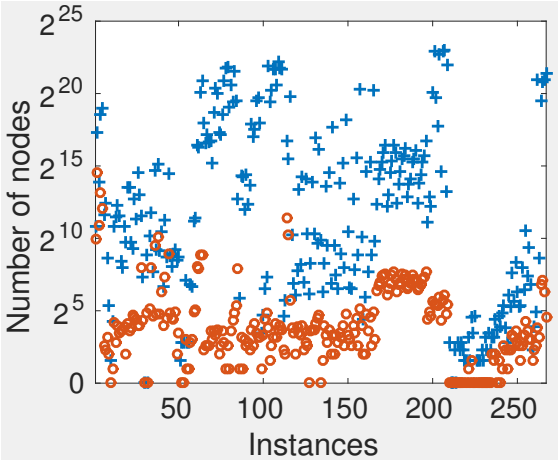


Figure 2.1: Total number of nodes processed (+) and the number of nodes with integer LP optimal solutions (o).

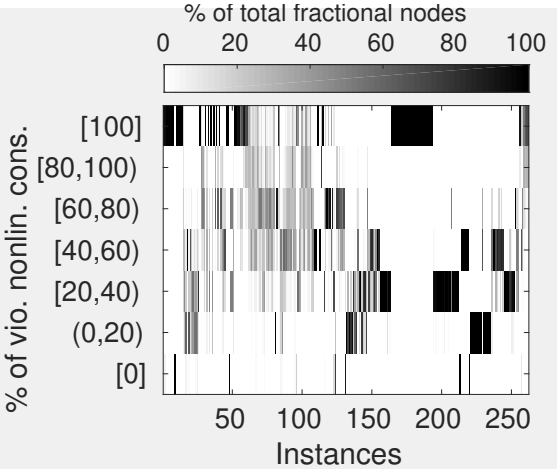


Figure 2.2: Distribution of violated nonlinear constraints at the nodes with fractional LP solutions.

the form

$$\nabla g_i(x')^\top (x - x') + g_i(x') \leq 0. \tag{grad-I}$$

This relaxation can be tightened by adding linearization inequalities obtained from multiple points. We propose schemes that try to identify more effective linearization inequalities by finding suitable points of linearization.

2.1 Experimental Setup

All the computational experiments have been carried out on a system with two 64-bit Intel(R) Xeon(R) E5-2670 v2, 2.50GHz CPUs having 10 cores each and sharing 128GB RAM. Hyperthreading is disabled. Our schemes are available in the development version of MINOTAUR[1]. All codes are complied with GCC-4.9.2 compiler. IPOPT-3.12 with MA97 linear-systems solver is used as the NLP solver. CPLEX-12.8 has been used as the LP solver. We have set a limit of one hour on the wall clock time in all our experiments and reported all the solution times in seconds. Appendix B provides a brief decription of the test instances that are used for the computational experiments in this chapter and in Chapter 3.

[1] Available at http://github.com/minotaur-solver/minotaur

2.2 Linearization Schemes at the Root Node

Given a problem (P) and the solution x^0 of its continuous relaxation (NLP), let $\bar{P}_k$ be a polyhedron corresponding to the k^{th} nonlinear constraint ($g_k(x) \leq 0,\ k \in M$) defined as

$$\bar{P}_k := \{x : \nabla g_k(x^0)^\top (x - x^0) + g_k(x^0) \leq 0\}. \tag{2.1}$$

The feasible region of the root LP relaxation can be interpreted as an intersection of polyhedra $\bar{P}_k, k \in M$, corresponding to the nonlinear constraints, and X. In this section, we propose five schemes that aim to tighten the LP relaxation at the root node by tightening $\bar{P}_k, k \in M$.

The first two schemes are designed for problems in which a nonlinear constraint, $g_k(x) \leq 0$, has a univariate nonlinear structure, i.e., g_k is the sum of a univariate nonlinear function ($h_k(x_i)$) and a linear function ($a_j x_j - b_k$, where b_k is a scalar), and the variable in the linear part of g_k does not appear in its nonlinear part. Mathematically, the nonlinear constraint is of the form

$$a_j x_j + h_k(x_i) \leq b_k, \tag{US}$$

where $a_j \neq 0$ and $j \neq i$. A nonlinear constraint with more than one term in its linear part can be transformed into this structure by replacing the entire linear part using an auxiliary variable.

This univariate structure appears in 126 out of 334 instances in test set TS_l (see Table C.1). Problem classes with this structure are listed in Table 2.1. In 123 of these instances, all the nonlinear constraints have this structure. Three instances, `ex1223a` and two of `synthes*`, have a few other constraints without this structure (US). We refer to the set of these 126 instances as TS_1 and the set of remaining 208 instances in TS_l as TS_2. The structure (US) has also been exploited by Hijazi *et al.* (2014) for building initial relaxation in outer approximation algorithms. They select points at regular intervals along x_i.

The feasible region of (US) can be visualized in the two-dimensional space of x_i and x_j variables. It is easy to see that a linearization (or a gradient inequality) generated at a point (x_i, x_j) touches the nonlinear constraint boundary at some point. More specifically, given a point x', all gradient inequalities at points of the form (x'_i, x_j), $x_j \in \mathbb{R}$ are the same and touch the constraint boundary at $\left(x'_i, \dfrac{b_k - h_k(x_i)}{a_j}\right)$. We utilize this simple fact in the first two schemes.

Table 2.1: Name of classes and number of instances (#) with the univariate structure (US) in a class. * following a name denotes a collection of instances in a class.

Name	#	Name	#	Name	#
cvxnonsep_normcon*r	3	fo8*	6	procurement2mot	1
cvxnonsep_nsig*r	3	fo9*	6	rsyn*m	24
cvxnonsep_pcon*r	3	m*	8	sssd*	13
ex1223a	1	no7*	5	syn*m	24
flay*	10	nvs03	1	synthes*	2
fo7*	7	o7*	9	**Total**	**126**

2.2.1 Root Linearization Scheme 1 (RS1)

Given a nonlinear constraint with the univariate structure (US), this iterative scheme selects a point in each iteration for generating a linearization until the violation of the nonlinear constraint at all points in the updated $\bar{P}_k$ is less than a desired value $\tilde{T}_k$.

The scheme starts by generating linearizations at points $x^L = (l_i, (b_k - h_k(l_i))/a_j)$ and $x^U = (u_i, (b_k - h_k(u_i))/a_j)$, where l_i and u_i are the lower and upper bounds, respectively, on x_i. Both x^L and x^U lie on the boundary of the feasible region of (US). Let us add to $\bar{P}_k$ two linearizations $L(x^L)$ and $L(x^U)$ at these points. Amongst all points in the updated $\bar{P}_k$, the violation of the constraint (US) is maximum at the point of intersection, x^I, of $L(x^L)$ and $L(x^U)$. Let E_k be the set of extreme points of $\bar{P}_k$. At any point $x^l \in E_k$, let $v(x^l)$ be the violation of the nonlinear constraint defined as $v(x^l) = \max\{a_j x^l_j + h_k(x^l_i) - b_k, 0\}$, where x^l_i and x^l_j are the values of variables x_i and x_j in x^l. In each iteration, candidate points for generating a new linearization are those points $x^l \in E_k$ for which $v(x^l) \geq \tilde{T}_k$. Amongst these candidates, the one with maximum violation is selected. Figure 2.3 shows a pictorial depiction of this scheme and Algorithm 5 presents the pseudocode for this scheme.

We compare the default implementation of QG in MINOTAUR, which we refer to as *qg* to that of *qg* with scheme RS1 (denoted as *qgrs1*). Threshold $\tilde{T}_k$ is set to be a fraction K of b_k, if $b_k \neq 0$, otherwise of $v(x^I)$. We tried four different values of K : 0.02, 0.05, 0.10, 0.20. In case any of the bounds, l_i or u_i, on variable x_i is not known, we take $l_i = x^0_i - 50$ and $u_i = x^0_i + 50$, respectively. Table 2.2 shows the impact of this scheme on the overall solution time, size of the tree in terms of the number of nodes processed, and the Euclidean distance of the optimal solution ($\overline{x}$) of the root LP from the feasible region

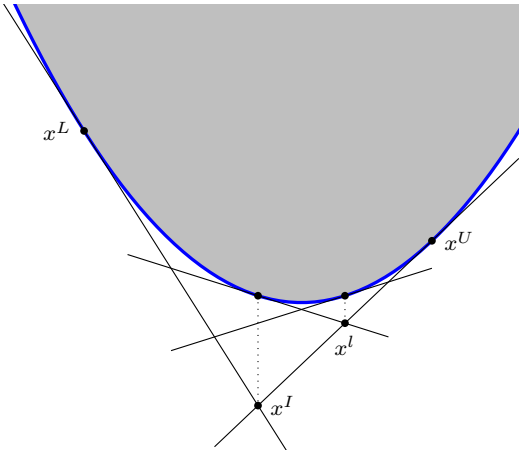


Figure 2.3: Depiction of linearization scheme RS1.

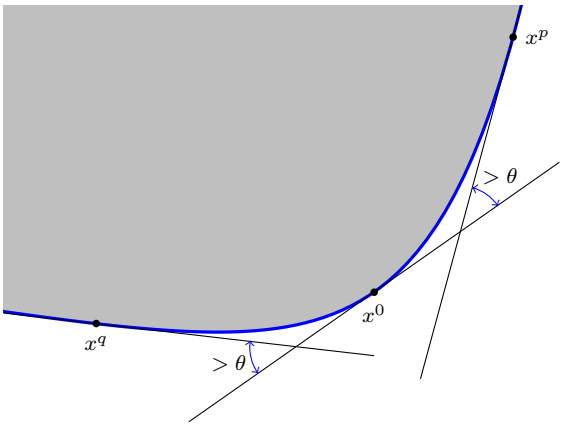


Figure 2.4: Depiction of linearization scheme RS2.

Algorithm 5: Root linearization scheme RS1.

Input: k^{th} nonlinear constraint with structure (US), a scalar K and initial $\bar{P}_k$.

1 Compute points x^L and x^U. Generate linearizations $L(x^L)$ and $L(x^U)$ tor the nonlinear constraint at these points, and add to $\bar{P}_k$.

2 Compute intersection point, x^I, of $L(x^L)$ and $L(x^U)$, and threshold value $\tilde{T}_k$.

3 Construct set $E_k = \{x^L, x^I, x^U\}$ of the extreme points of $\bar{P}_k$.

4 **while** $(\max_{x^l \in E_k}\{v(x^l)\} \geq \tilde{T}_k)$ **do**

5 Select $x^p \in E_k$ with the maximum violation value, generate linearization to the nonlinear constraint at x^p and add to $\bar{P}_k$.

6 Update set E_k by adding newly generated extreme points.

7 **end**

of (NLP). The following nonlinear program is solved for computing this distance.

$$\left.\begin{array}{ll} \underset{x}{\text{minimize}} & \|x - \overline{x}\|_2 \\ \text{subject to} & g_i(x) \leq 0,\ i \in M. \end{array}\right\} \qquad \text{(NLPD)}$$

Here, $\|.\|_2$ stands for Euclidean norm. Problem (NLPD) differs from (NLP) only in the objective function.

Each row of the top table in Table 2.2 corresponds to a parameter setting (K in this case). The column '# solved by' lists the number of instances solved to optimality within the time limit by the proposed method and by both the reference solver (*qg* in this case) as well as the proposed method (under the column 'both'). The first column under the headings 'time' and 'nodes' shows the shifted geometric mean (SGM) of these measures reported by the reference solver for the instances in the column 'both'. The second column

under these headings show the relative SGM ('rel.') of the proposed method for the same instances using the setting corresponding to the row. Similar statistics for the distance measure are computed, but over all the instances, not just for those solved within the time limit. For example, *qgrs1* with $K = 0.20$ is on an average 11% faster and showed an improvement of about 15% and 81% in the number of nodes processed and distance, respectively, over default *qg* on the set of 111 instances that were solved by both *qg* and *qgrs1*. We used a shift of 10 for calculating SGM of time and distance, and 100 for the number of nodes processed. More number of linearizations are added for a smaller value of K, and thus the resulting LP relaxation is tighter but bigger.

As mentioned by Achterberg (2007), using SGM as a performance measure in this context is better compared to the geometric mean, which gets skewed if there are small solution time values, and also compared to the arithmetic mean, which gets gets biased by the presence of large values. Thus, SGM has become a well-accepted statistic of choice to present computational results (Berthold and Csizmadia (2020); Witzig *et al.* (2017)).

Our results report modest improvements in all the considered performance measures under all the settings. We choose $K = 0.20$ as the default setting for this scheme as it solved 2 instances more than *qg* and resulted in about 11% improvement in solution times. The break-up of performance over instances of varying difficulty using the best setting is also included in Table 2.2 (bottom). Each row corresponds to the instances solved by both *qg* and *qgrs1*, but for which at least one of them took more than the specified time. For example, 31 instances were solved to optimality by both *qg* and *qgrs1*, and for each of these instances, at least one of the two solvers took more than 100 seconds. We observed that *qgrs1* is more effective for 'difficult' problems, especially those corresponding to row 3 in the table on the bottom. Similar tables have been used in the rest of this chapter and the next chapter as well.

2.2.2 Root Linearization Scheme 2 (RS2)

Given a nonlinear constraint with univariate structure (US), this scheme iteratively selects points in a way that the successively generated linearization constraints differ in their slope by at least a specified threshold value. Like RS1, the feasible region of (US) can be seen as a two-dimensional region in the space of x_i and x_j variables. We start at x^0, an optimal solution of (NLP). First, x_i^0 is gradually increased using step size δ and variable x_j is determined. If the slope of the linearization at this point differs from the slope of the previously accepted linearization by θ, it is added to the linear relaxation. Otherwise the step size δ is doubled. This process is repeated until x_i^0 exceeds $\min\{u_i, x_i^0 + \Delta\}$ for a scalar parameter $\Delta > 0$. A similar search is carried out in the opposite direction until x_i^0 falls

	# solved by		time		nodes		distance	
K	*qgrs1*	both	*qg*	rel.	*qg*	rel.	*qg*	rel.
0.02	113	111	31.66	0.93	6.9e3	0.79	1.35	0.02
0.05	113	111	31.38	0.86	6.9e3	0.77	1.36	0.05
0.10	113	111	31.54	0.91	6.9e3	0.86	1.37	0.08
0.20	115	111	31.54	0.89	6.9e3	0.85	1.37	0.19

	# solved	time		nodes		distance	
time	by both	*qg*	rel.	*qg*	rel.	*qg*	rel.
> 0	111	31.54	0.89	6.9e3	0.85	1.37	0.19
> 10	53	164.82	0.86	1.5e5	0.88	1.19	0.12
> 100	31	453.82	0.79	4.7e5	0.79	0.89	0.09
> 500	14	1133.83	0.87	1.2e6	0.86	0.68	0.06

Table 2.2: (Top) Comparison of *qg* and *qgrs1* for different values of K on test set TS_1. *qg* could solve 113 instances in the time limit. (Bottom) Performance break-up of *qgrs1* with $K = 0.20$ over instances of varying difficulty in TS_1.

below $\max\{l_i, x_i^0 - \Delta\}$. Figure 2.4 gives a pictorial description of the scheme and Figure 6 presents pseudocode of RS2 along the direction $-e_i$.

Algorithm 6: Root linearization scheme RS2 for the direction $-e_i$.

Input: k^{th} nonlinear constraint with structure (US), x^0, parameters θ, Δ, δ, l_i, u_i, and iteration counter $p = 1$.

if $x_i^0 - \max\{l_i, x_i^0 - \Delta\} < 1$ **then**
  Set $\delta = x_i^0 - \max\{l_i, x_i^0 - \Delta\}$
Set $x_i^1 = x_i^0 - \delta$ and $x_j^1 = (b_k - h_k(x_i^1))/a_j$.
while $x_i^p \geq max\{l_i, x_i^0 - \Delta\}$ **do**
  Compute angle, α, between the linearization at x^p and the last added one.
  if $\alpha \geq \theta$ **then**
    Add linearization at x^p.
  else
    Set $\delta \leftarrow 2\delta$.
  end
  Set $p = p + 1$, $x_i^p = x_i^{p-1} - \delta$ and $x_j^p = (b_k - h_k(x_i^p))/a_j$.
end

Computational performance of *qg* with linearization scheme RS2 (*qgrs2*) on test set TS_1 is presented in Table 2.3. We used four values of $\theta = 2, 5, 10, 20$ with $\Delta = 10$, and $\delta = 0.5$. In our experiments, we observed improvements in solution time and tree-size for the first two settings. Quality of the relaxation improved for all the considered θ values - more for smaller values because more linearizations were added, implying a tighter, but larger LP. Although more instances than *qg* and other settings were solved with $\theta = 10$, it resulted in poor solution times. *qgrs2* with $\theta = 5$ solved two instances less than *qg* but resulted in better performance on 109 instances that were solved by both *qg* and *qgrs2*. Table 2.3 shows the break-up of its performance over instances of varying difficulty.

	# solved by		time		nodes		distance	
θ	*qgrs2*	both	*qg*	rel.	*qg*	rel.	*qg*	rel.
2	112	110	30.44	0.95	6.6e3	0.88	1.37	0.11
5	111	109	28.64	0.87	6.2e3	0.86	1.38	0.17
10	115	112	33.01	1.00	7.3e3	1.00	1.35	0.50
20	113	112	33.01	1.01	7.3e3	1.03	1.35	0.73

	# solved	time		nodes		distance	
time	by both	*qg*	rel.	*qg*	rel.	*qg*	rel.
> 0	109	28.64	0.87	6.2e3	0.86	1.38	0.17
> 10	51	147.57	0.81	1.3e5	0.82	1.20	0.10
> 100	28	432.08	0.74	4.4e5	0.72	0.92	0.07
> 500	13	964.24	0.85	1.0e6	0.83	0.65	0.05

Table 2.3: (Top) Comparison of *qg* and *qgrs2* for different values of θ on test set TS_1. *qg* could solve 113 instances. (Bottom) Break-up of performance over instances of varying difficulty for *qgrs2* with $\theta = 5$.

Time taken within the schemes *qgrs1* and *qgrs2* is negligible (less than 0.5s) in comparison to the total solution time for all the considered instances.

The next three schemes are applicable to a general problem of the form (P).

2.2.3 Root Linearization Scheme 3 (RS3)

Consider the following root LP relaxation of (P). Nonlinear constraints are replaced with gradient inequalities at x^0, an optimal solution of continuous relaxation of the problem. If the objective function in (P) is also nonlinear, the problem is reformulated by replacing the objective with an auxiliary variable, η, and adding the constraint $f(x) \leq \eta$. This new constraint is also replaced with its linearization inequality at x^0 and integrality

constraints are relaxed.

$$\left.\begin{aligned} \underset{x,\eta}{\text{minimize}} \quad & \eta \\ \text{subject to} \quad & \nabla f(x^0)^\top (x - x^0) + f(x^0) \leq \eta, \\ & \nabla g_k(x^0)^\top (x - x^0) + g_k(x^0) \leq 0,\ k \in M. \end{aligned}\right\} \tag{LP}$$

This scheme finds linearization points near an optimal solution of root LP relaxation (LP). First the root LP relaxation is solved and an optimal solution, $\bar{x}$, is obtained. If the solution violates any nonlinear constraint, a line search is performed between $\bar{x}$ and x^C to find a point at the boundary of the feasible region of (NLP). x^C is chosen to be a point inside the feasible region of (NLP). The boundary point is used to generate new linearizations. The updated LP is solved again and the process is continued. The point x^C remains the same at every iteration. We stop when the LP solution is feasible to (NLP) or a preset number (k_{max}) of LPs have been solved.

To obtain an interior point x^C, we solve the following nonlinear problem (NLPI). All the nonlinear inequalities in (NLP) are modified using an auxiliary variable ν, which also forms the objective of (NLPI). All the other (linear) constraints remain unchanged.

$$\left.\begin{aligned} \underset{x,\nu}{\text{minimize}} \quad & \nu \\ \text{subject to} \quad & g_i(x) \leq \nu,\ i \in M, \\ & \nu \leq 0. \end{aligned}\right\} \tag{NLPI}$$

Let the optimal solution of (NLPI) be $(\tilde{\nu}, \tilde{x})$. If $\tilde{\nu} < 0$, then we set $x^C = \tilde{x}$. If $\tilde{\nu} = 0$, then there does not exist any point in the feasible region of (NLP) at which all the nonlinear constraints are inactive. In this case, we simply generate linearizations to nonlinear constraints that are active at $\tilde{x}$, and terminate the scheme. If (NLPI) is unbounded, then we add ν to the linear inequalities in the same way as the nonlinear constraints and re-solve.

Algorithm 7 presents the pseudocode for this scheme. This scheme is similar to root LP generation in the ESH algorithm by Kronqvist *et al.* (2016), but differs in the formulation of initial root LP and the nonlinear problem (NLPI). Unlike in Kronqvist *et al.* (2016), our initial root LP is obtained by linearizing nonlinear constraints at x^0, and we also consider linear equalities to find the required interior point, thus ensuring x^C lies in the feasible region of (NLP). Out of the total 334 instances in the test set TS_l, 67 have nonlinearity only in the objective (see Table C.1). The remaining 267 resulted in an optimal solution with $\tilde{\nu} < 0$. Our computational investigations indicate that the choice of interior point plays an important role in determining the quality of linearizations generated. We experimented first with x^C as obtained from solving (NLPI). Next, we used the center of the line segment between x^C and x^0 as the required interior point, which also

lies in the interior of the feasible region of (NLP). Interior point obtained using the latter way resulted in a better performance.

For 67 instances with nonlinearity only in the objective function (all these instances lie in test set TS_2), root LP solution is also feasible to the problem (NLP). We add objective linearization directly at the LP solution obtained in every iteration. In this case, the algorithm terminates when the current LP solution is the same as the previous solution or when we exhaust a prefixed number of iterations, k_{max}.

Algorithm 7: Root linearization scheme RS3.

Input: An interior point x^C, a maximum iteration limit, k_{max}, the initial root LP and its solution $\bar{x}$.

1 Set the iteration count $k = 1$ and $\bar{x}^1 = \bar{x}$.
2 **while** ($k \leq k_{max}$ *and* $\bar{x} \notin$ *feasible region of* (*NLP*)) **do**
3 Find $0 \leq \lambda \leq 1$ such that $x^B = \lambda x^C + (1 - \lambda)\bar{x}^k$ lies on the boundary of (NLP).
4 Add linearizations of all the nonlinear constraints active at x^B.
5 Set $k = k + 1$, solve the resulting LP and let $\bar{x}^k$ be its optimal solution.
6 **end**

We compare *qg* and *qg* with scheme RS3 (*qgrs3*) using $k_{max} = 5, 10, 20, 40$. Table 2.4 and Table 2.5 report values for different performance metrics on TS_1 and TS_2 respectively. The time taken in this scheme is a very small fraction of the total solution time in all the considered instances. The maximum time taken was close to 2 second for instances with a large number of variables.

We observed only small improvements in the performance metrics over the set TS_1, but reasonable improvements for TS_2. We obtained an improvement of about 5% on TS_1 and of 8% on TS_2 in solution times. Table 2.4 and Table 2.5 (bottom ones) provide a break-up of performance for instances in test sets TS_1 and TS_2, respectively, from the best settings on these sets. Larger improvements in solution times are seen for more difficult and structured instances (Table 2.4, last row in the bottom table). However, we can not predict whether an instance is 'easy' or 'difficult' before solving it. Overall, *qgrs3* is slower than *qgrs1* and *qgrs2* on test set TS_1.

2.2.4 Root Linearization Scheme 4 (RS4)

In this scheme, we search for linearization points by exploring several 'well spread' directions. Starting from an interior point of the feasible region of (NLP), we move along

	# solved by		time		nodes		distance	
k_{max}	*qgrs3*	both	*qg*	rel.	*qg*	rel.	*qg*	rel.
5	113	110	29.94	0.99	6.5e3	1.03	1.38	0.26
10	113	111	31.54	0.95	6.9e3	0.98	1.37	0.12
20	113	111	31.54	1.01	6.9e3	1.01	1.37	0.05
40	112	111	31.66	1.06	6.9e3	1.06	1.35	0.01

	# solved	time		nodes		distance	
time	by both	*qg*	rel.	*qg*	rel.	*qg*	rel.
> 0	111	31.54	0.95	6.9e3	0.98	1.37	0.12
> 10	54	156.81	0.92	1.4e5	0.98	1.17	0.05
> 100	31	451.61	0.84	4.7e5	0.88	0.89	0.02
> 500	17	933.18	0.81	1.0e6	0.83	0.64	0.02

Table 2.4: (Top) Comparison of *qg* and *qgrs3* for different values of k_{max} on test set TS_1. *qg* could solve 113 instances. (Bottom) Break-up of results over instances of varying difficulty with the best setting, $k_{max} = 10$.

	# solved by		time		nodes		distance	
k_{max}	*qgrs3*	both	*qg*	rel.	*qg*	rel.	*qg*	rel.
5	180	177	11.86	0.99	1.0e3	1.00	5.73	0.48
10	179	176	11.33	0.95	1.0e3	0.96	5.78	0.28
20	179	177	11.86	0.96	1.0e3	0.93	5.73	0.19
40	179	177	11.86	0.92	1.0e3	0.88	5.73	0.07

	# solved	time		nodes		distance	
time	by both	*qg*	rel.	*qg*	rel.	*qg*	rel.
> 0	177	11.86	0.92	1.0e3	0.88	5.73	0.07
> 10	64	56.41	0.88	1.0e4	0.82	17.5	0.06
> 100	18	336.23	0.93	1.4e4	0.89	4.61	0.08
> 500	7	1548.07	0.90	3.8e4	0.95	0.56	0.19

Table 2.5: (Top) Comparison of *qg* and *qgrs3* for different values of k_{max} on test set TS_2. *qg* could solve 179 instances. (Bottom) Break-up of results over instances of varying difficulty with the best setting $k_{max} = 40$.

each chosen direction until the boundary of the feasible region of (NLP) is reached. We add linearizations to all the nonlinear constraints that are active at the obtained boundary point. The interior point is computed in the same way as in RS3. For search directions, we

use positive and negative standard basis which consists of directions of the form $\{e_j, -e_j\}$, $\forall j \in D$, where D is the set of indices of variables that appear in the nonlinear part of some constraint or objective and e_j is the j^{th} unit vector. This means that there are at most $2|D|$ directions and points for linearizations.

For problems with nonlinearity only in the objective function, this scheme is changed slightly. In the problem (NLPI) for finding an interior point, linear inequalities are modified in the same way as nonlinear inequalities. If $\tilde{v} < 0$, then the scheme is same as for the problems with nonlinear constraints with the only difference that in the place of nonlinear constraints, linear constraints are used. In the rare case when $\tilde{v} = 0$ or if there exists a linear equality constraint, the point $\tilde{x}$ lies on the boundary of the feasible region of (NLP). In this case, we consider the four equidistant points on the line segment between $\tilde{x}$ and x^0. We generate linearizations at these four points that also lie on the boundary of the feasible region of (NLP). A similar step is performed along the opposite direction, $d = \tilde{x} - x^0$. Starting from $\tilde{x}$, we consider four equidistant points on the line segment between $\tilde{x}$ and $(2\tilde{x} - x^0)$. Out of these four points, the ones which are feasible to (NLP) are selected for generating linearizations.

We observed that in set TS_2, many instances in the class of problems such as ibs2, squfl0*, unitcommit_200_100*, watercontamination*, etc., have a large number of variables in their nonlinear part resulting in a large number of elements in the set D. For such problems, we restrict the size of set D, thus limiting the amount of time spent in this scheme by searching along fewer directions.

In our runs, we limit the size of D to a maximum of 300, selecting only the first 300 directions. First, we chose x^C as defined in RS3 scheme as the interior point and referred to this setting as FC. Then, we used the mid-point of the line segment joining x^C and x^0 as the interior point; this setting is termed as MC. Results from *qg* with RS4 (*qgrs4*) on test sets TS_1 and TS_2 using these two settings are shown in Table 2.6 and Table 2.7 respectively. The time taken within this scheme is again a very small fraction of the total solution time, most of which is spent in solving the nonlinear problem (NLPI) for finding the interior point.

On both the test sets, setting MC has performed better. On TS_1, *qgrs4* solved same number of instances as *qg*, but resulted in an improvement of about 12% in the solution times. Overall, this scheme is inferior to *qgrs1*, but better than both *qgrs2* and *qgrs3* on this test set. On TS_2, *qgrs4* solved two instances fewer than *qg*, but on 177 instances that were solved by both, it showed an improvement of about 6%. Although, *qgrs4* has solved two instances fewer than *qgrs3*, it seems to have performed better on 'difficult' instances (rows corresponding to time > 500 in the respective tables).

	# solved by		time		nodes		distance	
setting	*qgrs4*	both	*qg*	rel.	*qg*	rel.	*qg*	rel.
MC	113	111	31.54	0.88	6.9e3	0.89	1.37	0.61
FC	113	110	29.94	0.92	6.5e3	0.92	1.38	0.76

	# solved	time		nodes		distance	
time	by both	*qg*	rel.	*qg*	rel.	*qg*	rel.
> 0	111	31.54	0.88	6.9e3	0.89	1.37	0.61
> 10	54	156.81	0.82	1.4e5	0.88	1.17	0.47
> 100	33	414.91	0.75	4.3e5	0.79	0.86	0.46
> 500	14	1133.83	0.73	1.2e6	0.77	0.68	0.21

Table 2.6: (Top) Comparison of *qg* and *qgrs4* on test set TS_1. *qg* could solve 113 instances. (Bottom) Break-up of performance over instances of varying difficulty with the best setting, MC.

	# solved by		time		nodes		distance	
setting	*qgrs4*	both	*qg*	rel.	*qg*	rel.	*qg*	rel.
MC	177	177	11.86	0.94	1.0e3	0.93	5.73	0.59
FC	178	177	11.86	1.00	1.0e3	0.96	5.73	0.59

	# solved	time		nodes		distance	
time	by both	*qg*	rel.	*qg*	rel.	*qg*	rel.
> 0	177	11.86	0.94	1.0e3	0.93	5.73	0.59
> 10	63	57.50	0.93	1.1e4	0.94	7.62	0.50
> 100	16	416.56	0.86	2.3e4	0.88	7.56	0.71
> 500	7	1548.07	0.80	3.8e4	0.86	0.56	1.00

Table 2.7: (Top) Comparison of *qg* and *qgrs4* on test set TS_2. *qg* could solve 179 instances. (Bottom) Break-up of performance over instances of varying difficulty with best setting, MC.

2.2.5 Root Linearization Scheme 5 (RS5)

This scheme selects points for linearizations in a neighborhood of x^0, an optimal solution obtained by solving (NLP). Starting from x^0, we move in different directions to find suitable points. We consider two sets of directions. For the first set, we select affinely independent points on the hyperplane passing through x^0 and whose normal is $(x^C - x^0)$, where x^C is an interior point like in RS3. Let this hyperplane be denoted by $a^\top x = r$, where $a = x^C - x^0$, $j = 1, \ldots, n$ and $r = (x^C - x^0)^\top x^0$. Let j be any index such that $a_j \neq 0$,

and define a set of n affinely independent points x^i on this hyperplane as

$$x^i = \begin{cases} (r/a_i)e_i, & \text{if } a_i \neq 0, \\ (r/a_j)e_j + e_i, & \text{otherwise.} \end{cases} \tag{2.2}$$

Let DS_1 be the set of $(n-1)$ linearly independent directions, $x^i - x^1, i = 2, \ldots, n$. Each of these directions has at most two nonzero components.

For each direction d from the set DS_1, we search iteratively along d starting from x^0. At iteration l, we obtain a point $\bar{x}^l = \bar{x}^{l-1} + \delta d$, where δ is a positive step size and $\bar{x}^0 = x^0$. Then, starting from x^C, we perform a line search along direction $(\bar{x}^l - x^C)$ for finding a point x^B on the boundary of the feasible region of (NLP). For every nonlinear constraint active at x^B, we compute the angle between the normals of the linearization drawn at x^B and the previous linearization added to this nonlinear constraint. If this angle is more than a specified threshold θ (in degrees), then we add the linearization generated at x^B to the relaxation. If the objective is also nonlinear, we add an objective linearization at x^B using the same criterion of slope difference. If no linearizations are added at the current point x^B, then we double the step size δ and repeat the search. The search terminates when any component of the point $\bar{x}^l$ violates its bound (lower or upper). This process is repeated for every direction $d \in DS_1$ and also its negative. The whole procedure was tried on another set of directions, DS_2, as a variant of the above method. For each $d \in DS_1$, we replace its negative components by -1 and positive components by 1 to get a new direction. All these $n-1$ directions constitute DS_2. Rest of the procedure remains identical.

In order to choose an initial step size δ along a direction d, we consider the Hessian of the Lagrangian, H, at x^0. If the absolute value of $d^\top Hd$ is below a threshold, we take a step size δ_l, otherwise a smaller step size δ_s is chosen. For problems with nonlinearity only in the objective function, this scheme is modified in the same way as in RS4. However, unlike scheme RS4, if $\tilde{v} = 0$ or if there exists a linear equality constraint, then we consider points at an interval of δ_s on the line segment between the points x^C and x^0.

In our numerical experiments using *qg* with RS5 (*qgrs5*), we first used x^C (referred to as FC) and then modified it as in *qgrs4* (denoted as MC). Along with the two proposed set of directions, DS_1 and DS_2, we obtained four settings: FC-1, MC-1, FC-2, MC-2; for example, FC-1 corresponds to the setting in which interior point is chosen as FC and search directions are from DS_1. For each setting, we used four values for parameter $\theta = 2, 5, 10, 20$, $\delta_s = 0.25$, and $\delta_l = 1$. Out of the four settings, FC-2 with $\theta = 2$ exhibited the best results on both the test sets and are presented in Table 2.8 and Table 2.9. On TS_1, *qgrs5* with this setting solved 2 instances more than *qg* and exhibited an improvement of about 7% in solution times. Overall, on TS_1, *qgrs5* is inferior to all the previous schemes

except *qgrs3* in terms of solution times, but is better than all except *qgrs1* in terms of number of instances solved. On TS_2, it solved one instance more than *qg* and provided an improvement of about 12% in solution times. It also provided better solution times than *qgrs3* and *qgrs4*. Like *qgrs3* and *qgrs4*, most of the time taken by *qgrs5* is spent in solving the nonlinear problem (NLPI) for finding the interior point.

	# solved by		time		nodes		distance	
θ	*qgrs5*	both	*qg*	rel.	*qg*	rel.	*qg*	rel.
2	115	113	34.71	0.93	7.8e3	0.94	1.34	0.69
5	113	112	33.05	0.92	7.4e3	0.96	1.35	0.82
10	112	112	33.05	0.94	7.4e3	0.99	1.35	0.84
20	113	113	34.71	0.95	7.8e3	0.98	1.34	0.93

	# solved	time		nodes		distance	
time	by both	*qg*	rel.	*qg*	rel.	*qg*	rel.
> 0	113	34.71	0.93	7.8e3	0.94	1.34	0.69
> 10	55	183.02	0.88	1.7e5	0.91	1.14	0.64
> 100	33	505.64	0.84	5.5e5	0.86	0.84	0.67
> 500	16	1261.25	0.88	1.4e6	0.89	0.60	0.32

Table 2.8: (Top) Comparison of *qg* and *qgrs5* with FC-2 for different values of θ on test set TS_1. *qg* could solve 113 instances. (Bottom) Break-up of performance over instances of varying difficulty with the best setting $\theta = 2$.

2.3 Linearization Schemes at the Other Nodes

We now consider schemes for nodes (other than the root) that yield a fractional optimal solution in the branch-and-bound tree. Two main decisions in the design of these schemes are: (a) whether additional linearization constraints should be added at a given node, and (b) how to determine points for generating linearization constraints.

2.3.1 Node Linearization Scheme 1 (NS1)

Let x' and Z' be an optimal solution and corresponding optimal value obtained by solving the LP relaxation at a node. Given a nonlinear constraint $g_k(x) \leq 0$, let b_k denotes the constant term in it. We assign a violation based score $V^k = \frac{v_k - b_k}{|b_k|}$, if $b_k \neq 0$, and $V^k = v_k$ otherwise, where $v_k = max\{0, g_k(x')\}$. For a nonlinear objective, $f(x)$, score V^o is defined as $V^o = v_o/|Z'|$, if $Z' \neq 0$, and $V^o = v_o$ otherwise, with $v_o = max\{0, f(x') - Z'\}$.

	# solved by		time		nodes		distance	
θ	*qgrs5*	both	*qg*	rel.	*qg*	rel.	*qg*	rel.
2	180	177	11.86	0.88	1.0e3	0.88	5.73	0.27
5	180	177	11.86	0.95	1.0e3	0.92	5.73	0.56
10	179	177	11.86	0.97	1.0e3	0.94	5.73	0.88
20	178	176	11.33	0.98	1.0e3	0.98	5.78	0.90

	# solved	time		nodes		distance	
time	by both	*qg*	rel.	*qg*	rel.	*qg*	rel.
> 0	177	11.86	0.88	1.0e3	0.88	5.73	0.27
> 10	62	59.15	0.83	1.1e4	0.82	18.08	0.15
> 100	15	451.28	0.81	2.0e4	0.88	4.55	0.34
> 500	7	1548.07	0.87	3.8e4	0.93	0.56	1.11

Table 2.9: (Top) Comparison of *qg* and *qgrs5* with FC-2 for different values of θ on test set TS_2. *qg* could solve 179 instances. (Bottom) Break-up of performance over instances of varying difficulty with the best setting $\theta = 2$.

If the score of a nonlinear constraint is more than a preset threshold value τ, then we generate linearizations at the node. To avoid adding too many cuts, this scheme is applied only up to a certain depth, D, in the branch-and-bound tree.

We employ the following two methods for finding points for generating linearizations for problems that have at least one nonlinear constraint. The first method is based on the extended cutting plane technique (Westerlund and Pettersson (1995)), hence we refer to it as the ECP method. Here, we generate linearizations at x' to all nonlinear constraints whose score $V^k \geq \tau$.

If the objective is nonlinear, then we add a linearization to the objective at x' if $V^o \geq \tilde{K}$. Here, $\tilde{K}$ is initialized with $v_r/|\overline{Z}|$, if $\overline{Z} \neq 0$, and v_r otherwise, where $v_r = max\left\{0, f(\overline{x}) - \overline{Z}\right\}$, and $\overline{x}$ and $\overline{Z}$ denote an optimal solution and corresponding optimal value to the root LP relaxation, respectively. If $\tilde{K} < 0.5$, then we double the value of $\tilde{K}$.

The second method is based on line search and we refer to it as the LS method. Starting from an interior point $\tilde{x}$ in the feasible region of (NLP), we search along the direction $x' - \tilde{x}$ for a point on the boundary of the feasible region of (NLP). Then we generate linearizations at this boundary point to all the active nonlinear constraints. This method ensures that all the linearizations are tight. The chosen interior point, $\tilde{x}$, is the midpoint of the line segment joining x^C (an interior point obtained as in RS3) and x^0. For

problems with a nonlinear objective also, we add a linearization at the obtained boundary point if the criteria mentioned in ECP are met.

For problems that have nonlinearity only in the objective function, a node is selected for adding linearization if $V^o \geq \tilde{K}$, where $\tilde{K}$ is initialized in the same way as above. If $\tilde{K} > 1000$, then we reduce depth D by half, and if $\tilde{K} < 0.5$, we double the value of $\tilde{K}$ and D.

For these problems we employ only ECP method. This treatment to the problems with nonlinearity only in the objective remains the same in the following two schemes, NS2 and NS3, as well.

Using *qg* with scheme NS1 (*qgns1*), we experimented with four values of τ : $\{0.75, 1, 1.5, 2\}$, $D = 10$ for problems with nonlinear constraints, and $D = 5$ for problems with nonlinearity only in the objective. This scheme with both the methods have shown improvements in solution time and the number of nodes processed. We obtained better results with LS method than ECP on both the test sets. Results for test sets TS_1 and TS_2 are reported in Table 2.10 and Table 2.11 respectively. On TS_2, although *qgns1* solved one instance less than *qg*, fair improvements are seen in solution times. Best results are obtained using $\tau = 2$ in which we obtained an improvement of about 7% and 11% on TS_1 and TS_2 respectively in solution times. On TS_1, this scheme is inferior to all root schemes in terms of solution times, but comparable to *qgrs1* and *qgrs5* in terms of the number of instances solved. On TS_2, this scheme has performed better than root schemes *qgrs3* and *qgrs4*, but inferior to *qgrs5*.

2.3.2 Node Linearization Scheme 2 (NS2)

This scheme is similar to NS1 but differs in the nonlinear constraints that are analyzed at a given node. Here, we analyze violation of *important* nonlinear constraints only. A nonlinear constraint with index k is considered important based on a surrogate value for its dual multiplier. Let I be the index set of important constraints and is constructed as follows. Given a feasible solution x^l to (NLP), let d_k be the dual multiplier of the nonlinear constraint with index k at x^l, $d_{max} = \max_{k \in M}\{d_k\}$ be the maximum dual value among all the nonlinear constraints, and $\tilde{d}$ (≤ 1) be a positive parameter. We include indices of those nonlinear constraints in set I whose associated dual values are at least $\tilde{d}$ times of the maximum dual value d_{max}. Initially, set I is populated using x^0, an optimal solution of (NLP), and is recomputed every time the upper bound is updated using the corresponding solution. Since, the same dual multiplier values are used until a better solution is obtained, we call these values surrogate. For determining points for generating linearizations, the same two methods, ECP and LS, as in *qgns1* are used. The ECP method is slightly mod-

	# solved by		time		nodes	
τ	*qgns1*	both	*qg*	rel.	*qg*	rel.
0.75	115	113	34.71	0.98	7.8e3	1.00
1	114	113	34.71	0.99	7.8e3	1.01
1.5	114	113	34.71	0.95	7.8e3	0.98
2	113	113	34.71	0.93	7.8e3	0.98

	# solved	time		nodes	
time	by both	*qg*	rel.	*qg*	rel.
> 0	113	34.71	0.93	7.8e3	0.98
> 10	54	191.54	0.90	1.8e5	0.98
> 100	32	535.64	0.88	5.8e5	0.97
> 500	16	1261.25	0.90	1.4e6	0.98

Table 2.10: (Top) Comparing *qg* and *qgns1* using LS method for different values of τ on TS_1. *qg* could solve 113 instances. (Bottom) Break-up of results over instances of varying difficulty for the best setting, $\tau = 2$.

	# solved by		time		nodes	
τ	*qgns1*	both	*qg*	rel.	*qg*	rel.
0.75	178	177	11.86	0.91	1.0e3	0.89
1	178	177	11.86	0.90	1.0e3	0.90
1.5	178	177	11.86	0.91	1.0e3	0.92
2	178	177	11.86	0.89	1.0e3	0.92

	# solved	time		nodes	
time	by both	*qg*	rel.	*qg*	rel.
> 0	177	11.86	0.89	1.0e3	0.92
> 10	63	57.76	0.83	1.1e4	0.81
> 100	16	406.47	0.84	1.8e4	0.87
> 500	7	1548.07	0.85	3.8e4	1.03

Table 2.11: (Top) Comparing *qg* and *qgns1* using LS method for different values of τ on TS_2. *qg* could solve 179 instances. (Bottom) Break-up of performance over instances of varying difficulty for the best setting $\tau = 2$.

ified to consider only important constraints (in set I) for generating linearizations. Also, problems with nonlinearity only in the objective function are treated as in NS1.

In experiments using *qg* with NS2 (*qgns2*), we used the same values for parameter τ, D, and $\tilde{K}$. We used $\tilde{d} = 0.5$ for constructing set I. Again, on both the test sets, LS method for selecting points for linearizations has performed better than ECP.

Table 2.12 and Table 2.13 illustrate results from *qgns2* with LS method on TS_1 and TS_2 respectively. On TS_1, we obtained an improvement of about 7% and of about 13% on TS_2 in solution times. *qgns2* has performed better than *qgns1* on both the test sets TS_1 and TS_2.

	# solved by		time		nodes	
τ	*qgns2*	both	*qg*	rel.	*qg*	rel.
0.75	114	113	34.71	0.93	7.8e3	0.98
1	113	113	34.71	0.93	7.8e3	0.98
1.5	113	113	34.71	0.93	7.8e3	0.98
2	113	113	34.71	0.92	7.8e3	0.98

	# solved	time		nodes	
time	by both	*qg*	rel.	*qg*	rel.
> 0	113	34.71	0.93	7.8e3	0.98
> 10	54	191.54	0.89	1.8e5	0.98
> 100	32	535.64	0.86	5.8e5	0.97
> 500	16	1261.25	0.92	1.4e6	1.02

Table 2.12: (Top) Comparing *qg* and *qgns2* with LS method and various values of τ on TS_1. *qg* could solve 113 instances. (Bottom) Break-up of results over instances of varying difficulty for best setting $\tau = 0.75$.

2.3.3 Node Linearization Scheme 3 (NS3)

In this scheme, we use both the violation of nonlinear constraints, and their dual multipliers for deciding whether to select a given node for generating linearizations. First, we compute a score, $\hat{s}$, for the node as $\hat{s} = \sum_{k:v_k>0} (V^k + v_k \times d_k)/N$ where V^k, v_k and d_k are as defined in schemes NS1 and NS2, and N is the number of violated nonlinear constraints ($v_k > 0$) at x', an optimal solution to the LP relaxation of the node. If the score of the node is more than τ times the score of its parent ($\hat{p}$), then we consider the node for generating linearizations. First, parameter τ is initialized by a preset value. As the tree grows, τ is updated at every selected node (for adding linearizations) by taking its average with $\tilde{\tau} = \hat{s}/(\hat{p} + \epsilon)$, where ϵ is a small tolerance value which in our experiments is 0.001. This scheme is also implemented up to a depth D in the search-tree. Methods for finding

	# solved by		time		nodes	
τ	*qgns2*	both	*qg*	rel.	*qg*	rel.
0.75	178	177	11.86	0.87	1.0e3	0.89
1	178	177	11.86	0.87	1.0e3	0.90
1.5	178	177	11.86	0.89	1.0e3	0.91
2	178	177	11.86	0.88	1.0e3	0.91

	# solved	time		nodes	
time	by both	*qg*	rel.	*qg*	rel.
> 0	177	11.86	0.87	1.0e3	0.89
> 10	63	57.76	0.81	1.6e4	0.81
> 100	15	463.33	0.82	2.2e4	0.93
> 500	7	1548.07	0.81	3.8e4	1.03

Table 2.13: (Top) Comparing *qg* and *qgns2* with LS method and various values of τ on TS_2. *qg* could solve 179 instances. (Bottom) Break-up of results over instances of varying difficulty for best setting $\tau = 0.75$.

linearization points and treatment to problems with nonlinearity only in the objective remain same as in NS1.

In *qg* with NS3 (*qgns3*), we used $\tau = 0.5, 0.75, 1, 1.5$ and the same D as in *qgns1* and *qgns2*. We observed that the ECP method performed better on test set TS_1 and LS performed better on TS_2. In the former case, best performance is obtained using $\tau = 1.5$ where *qgns3* with ECP solved one instance more than *qg* with an improvement of about 11% in the solution time and of about 10% in the number of nodes processed. On TS_2, using $\tau = 0.5$ and LS method, *qgns3* solved two more instances than *qg* with an improvement of about 10% in solution times. These results are presented in Table 2.14 and Table 2.15. This scheme has performed better than *qgns1* and *qgns2* on both TS_1 and TS_2.

2.4 A Hybrid Linearization Scheme

We studied the effects of schemes obtained by combining linearization schemes at the root node with those for fractional nodes. We present a hybrid scheme (Hyb) that automatically identifies structure (US) in a problem and applies linearization schemes RS1 and NS3. For instances (in TS_2) without this structure, Hyb scheme employs RS5 and NS3. Results from *qg* using the hybrid scheme Hyb (denoted *qgHyb*) on TS_1 and TS_2 are detailed in Table 2.16 and Table 2.17 respectively. These results show that on TS_1, *qgHyb* (with

	# solved by		time		nodes	
τ	*qgns3*	both	*qg*	rel.	*qg*	rel.
0.5	113	111	31.54	0.94	6.9e3	0.89
0.75	113	110	30.21	0.93	6.9e3	0.89
1	114	111	31.54	0.93	6.9e3	0.90
1.5	114	111	31.54	0.89	6.9e3	0.90

	# solved	time		nodes	
time	by both	*qg*	rel.	*qg*	rel.
> 0	111	31.54	0.89	6.9e3	0.90
> 10	53	163.83	0.86	1.5e5	0.88
> 100	32	430.23	0.82	4.5e5	0.83
> 500	16	1000.65	0.85	1.1e6	0.84

Table 2.14: (Top) Comparing *qg* and *qgns3* with ECP method and different values of τ on TS_1. *qg* could solve 113 instances. (Bottom) Break-up of results over instances of varying difficulty for best setting, $\tau = 1.5$.

	# solved by		time		nodes	
τ	*qgns3*	both	*qg*	rel.	*qg*	rel.
0.5	181	179	11.72	0.90	1.0e3	0.96
0.75	180	179	11.72	0.90	1.0e3	0.95
1	180	179	11.72	0.92	1.0e3	0.96
1.5	180	179	11.72	0.93	1.0e3	0.95

	# solved	time		nodes	
time	by both	*qg*	rel.	*qg*	rel.
> 0	179	11.72	0.90	1.0e3	0.96
> 10	63	58.02	0.87	1.1e4	0.94
> 100	15	463.33	0.77	2.2e4	0.91
> 500	7	1548.07	0.81	3.8e4	1.02

Table 2.15: (Top) Comparing *qg* and *qgns3* with LS method and different values of τ on TS_2. *qg* could solve 179 instances. (Bottom) Break-up of results over instances of varying difficulty for best setting, $\tau = 0.5$.

$K = 0.05$ and $\tau = 1.5$) has solved 2 instances more than *qg* with an improvement of about 11% in solution times and about 22% in the number of nodes processed. On TS_2, *qgHyb* (with $\theta = 2$ and $\tau = 1.5$) has solved one instance more than *qg* with about 13% reduction

in both solution times and nodes processed. Overall, on test set TS_l, we solved 3 more instances and obtained an improvement of about 12% in the solution times over *qg*.

# solved by		time		nodes	
qgHyb	both	*qg*	rel.	*qg*	rel.
115	112	33.01	0.89	7.3e3	0.78

time	# solved by both	time *qg*	time rel.	nodes *qg*	nodes rel.
> 0	112	33.01	0.89	7.3e3	0.78
> 10	54	172.97	0.86	1.6e5	0.82
> 100	34	430.69	0.81	4.7e5	0.75
> 500	16	1115.30	0.89	1.2e6	0.80

Table 2.16: (Top) Comparison of *qg* and *qgHyb* on TS_1. *qg* could solve 113 instances. (Bottom) Break-up of performance of *qgHyb* over instances of varying difficulty.

# solved by		time		nodes	
qgHyb	both	*qg*	rel.	*qg*	rel.
180	177	11.86	0.87	1.0e3	0.86

time	# solved by both	time *qg*	time rel.	nodes *qg*	nodes rel.
> 0	177	11.86	0.87	1.0e3	0.86
> 10	64	56.37	0.81	1.1e4	0.84
> 100	16	408.04	0.82	2.1e4	0.85
> 500	7	1548.07	0.69	3.8e4	0.85

Table 2.17: (Top) Comparison of *qg* and *qgHyb* on TS_2. *qg* could solve 179 instances. (Bottom) Break-up of performance of *qgHyb* over instances of varying difficulty.

2.5 Effect of Linearization Schemes in Parallel Implementation of QG

We study the impact of proposed linearization schemes on a parallel implementation of QG in MINOTAUR, referred to as *mcqg*. The basic idea of *mcqg* is to solve multiple LP subproblems in the tree simultaneously using multiple available processors on a shared-memory system. Also, information generated in the tree by different processors, like

feasible solutions, pseudocosts etc. are shared. More details on the implementation of the considered parallel implementation can be found in Sharma *et al.* (2020b). Our numerical experiments show that deploying linearization schemes within parallel tree-search further enhances the performance of *qg*. We show the performance of *mcqg* with the hybrid linearization scheme Hyb presented in Section 2.4. We refer to the combination of *mcqg* with Hyb as *mcqgHyb* and compare it to both *qg* and (plain) *mcqg*. For experiments using *mcqg* and *mcqgHyb*, we have used 16 threads. Table 2.18 and Table 2.19 show the performance of *mcqgHyb* on test sets TS_1 and TS_2, respectively. Note that the wall clock time taken by the sequential algorithm (*qg*) is the same as the CPU time. Using *mcqgHyb* on TS_1, we observed a significant improvement of about 52% in the solution times and solved 6 instances more than *qg*. On TS_2, we solved 2 more instances and obtained an improvement of about 38% in the solution times.

method	# solved by		wall time		nodes	
(M)	M	both	*qg*	rel.	*qg*	rel.
mcqg	115	111	31.54	0.54	6.9e3	1.35
mcqgHyb	119	112	33.01	0.48	7.3e3	1.07

	# solved	wall time		nodes	
time	by both	*qg*	rel.	*qg*	rel.
> 0	112	33.01	0.48	7.3e3	1.07
> 10	53	181.01	0.32	1.7e5	1.09
> 100	31	504.06	0.27	5.3e5	0.94
> 500	16	1021.32	0.31	1.1e6	1.19

Table 2.18: (Top) Comparison of *mcqg* and *mcqgHyb* to *qg* on test set TS_1. *qg* could solve 113 instances. (Bottom) Break-up of results of *mcqgHyb* over instances of varying difficulty.

2.6 Conclusions

To conclude, the serial implementation of QG sees about 12% and 15% improvements in the solution time and tree size, respectively, by using the proposed linearization schemes. The schemes reduce the distance between the root LP solution and the feasible region of the continuous relaxation at the root node by a far greater extent than the reduction in the solution time. Exploiting the univariate structure in nonlinear constraints has a bigger impact as compared to general purpose routines. Proposed linearization schemes

method	# solved by		wall time		nodes	
(M)	M	both	*qg*	rel.	*qg*	rel.
mcqg	179	177	11.86	0.88	1.0e3	1.45
mcqgHyb	181	176	11.42	0.62	1.0e3	1.19

	# solved	wall time		nodes	
time	by both	*qg*	rel.	*qg*	rel.
> 0	176	11.42	0.62	1.0e3	1.19
> 10	64	52.15	0.47	1.1e4	1.08
> 100	14	445.50	0.35	2.4e4	0.86
> 500	6	1727.45	0.28	5.2e4	0.81

Table 2.19: (Top) Comparison of *qg* and *mcqgHyb* on test set TS_2. *qg* could solve 179 instances. (Bottom) Break-up of results of *mcqgHyb* over instances of varying difficulty.

are shown to have favorable impact on the performance of a parallel implementation of the QG method as well.

Chapter 3

Automatic Reformulations

One of the main algorithmic ideas in solving a convex MINLP (P) is to generate and strengthen bounds (lower and upper) on its optimal objective value, Z^*. The quality of these bounds influences the convergence of an algorithm, where tight (good quality) bounds often contribute to solving a problem faster. Iteratively tightened relaxations are solved in a branch-and-bound framework to obtain lower bounds on Z^*. Since tight relaxations often give good bounds, one seeks to generate tight relaxations at different nodes in a tree-search, especially at the root node. Reformulation is one of the ways to tighten a problem's relaxation. The tightest possible continuous relaxation of (P) is its convex hull. But, finding a convex hull can be as hard as solving the original problem.

In some cases, finding a convex hull description of some relaxation of (P) may be easy, and using such a description, one can reformulate (P) to strengthen its overall continuous relaxation. In this chapter, we study one such reformulation, called the Perspective Reformulation (PR) by Frangioni and Gentile (2006) and Günlük and Linderoth (2010). This reformulation uses a special disjunctive set, called the 'on-off' set, whose convex hull can be defined using the perspective function in the space of the original variables.

Another useful problem reformulation results from utilizing the 'separability' property of the nonlinear function defining a constraint or the objective in (P). Function separability is widely used for approximating the values of the Jacobian matrix of a function, for solving nonconvex MINLPs (especially factorable programs) by creating a separable equivalent of the original problem (Ryoo and Sahinidis (1996); Shectman and Sahinidis (1998)), in methods like the interval branch-and-bound for global optimization (Berenguel *et al.* (2013)), etc.

For convex MINLPs, using function separability, one can form an extended formulation (with extra variables and constraints) of (P), which helps generate tight polyhedral approximations in cutting-plane, and branch-and-cut based methods (Kronqvist *et al.*

(2018b)). The effectiveness of exploiting separability is demonstrated by Hijazi *et al.* (2014) through example (exBall) of a convex MINLP that involves one nonlinear constraint with a separable quadratic function.

$$\left.\begin{array}{ll} \underset{x}{\text{minimize}} & 0^\top x \\ \text{subject to} & \sum_{j=1}^{n} \left(x_j - \frac{1}{2}\right)^2 \leq \frac{n-1}{4}\Big\}, \\ & x \in \{0,1\}^n. \end{array}\right\} \tag{exBall}$$

Here, the feasible region is defined as an intersection of a hyperball of radius $\sqrt{(n-1)}/2$ centered at $(1/2, \ldots, 1/2)$, with the vertices of the unit hypercube $\{0,1\}^n$. Moreover, the feasible region is empty. It is shown that outer-approximation based algorithms take an exponential number (2^n) of iterations to prove the infeasibility of this problem. On the other hand, using the separability of the nonlinear function in (exBall), the number of cuts required by an algorithm to converge reduces to $2n$.

On another front, the use of function transformations can induce separability. For example, log and power transformations help reduce some general nonlinear functions to separable nonlinear functions. These transformations enable reformulation of general MINLPs to convex MINLPs, which can then be solved by exploiting function separability in a branch-and-cut framework (Bonami *et al.* (2008b); Kronqvist *et al.* (2018b)).

'On-off' sets and separable nonlinear functions appear in many optimization problems and the reformulations utilizing these structures help solve these problems faster. With this motivation, we automatically detect such structures in a given mathematical formulation (and exploiting them) and show their significance in a computational optimization environment using MINOTAUR. More specifically, this chapter presents how to detect structures that signify separability in nonlinear functions, the applicability of the perspective reformulation, and how to solve the reformulated problems in a branch-and-cut framework.

3.1 Perspective Reformulation

First, we present the definition of the perspective function used in describing a convex hull of the sets of our interest. Given a function $f(x) : \mathbb{R}^n \to \mathbb{R}$, let a function $\check{f}(x,\lambda) : \mathbb{R}^{n+1} \to \mathbb{R}$ be defined as

$$\check{f}(x,\lambda) = \begin{cases} \lambda f\Big(\dfrac{x-(1-\lambda)\hat{x}}{\lambda}\Big), & \text{if } \lambda > 0, \\ 0, & \text{if } \lambda = 0, \\ \infty, & \text{otherwise,} \end{cases} \tag{PF}$$

where $\hat{x}$ is some fixed vector. It can be easily shown that if f is convex, then $\check{f}$ is also convex. When $\hat{x} = 0$, the function $\check{f}$ is known as the perspective function of f.

Proposition 3.1.1. *$\check{f}(x, \lambda)$ is convex if and only if $f(x)$ is convex.*

Ceria and Soares (1999) extend the results of Balas (1985) and Balas (1998) on disjunctive programming involving polyhedral sets to disjunctive convex programming. Theorem 3.1.2 by Ceria and Soares (1999) presents convex hull description of the union of a finite number of closed convex sets (defined using convex functions) in a higher-dimensional space using the perspective function. Let a set K be defined as

$$K = conv\Big(\bigcup_{j \in J} K^j \Big), \tag{3.1}$$

where every set K^j is a closed convex set and is given by $K^j := \Big\{ x \in \mathbb{R}^n : F^j(x) \leq 0 \Big\}$ and $F^j : \mathbb{R}^n \to \mathbb{R}^{m_j}$ is a vector-valued function whose every component is a closed convex function. Perspective functions F^j and their closure, denoted as $\check{F}^j$ and $\Big(cl\ F^j\Big)$ respectively, are given by

$$\check{F}^j(x^j, \lambda_j) = \begin{cases} \lambda_j F^j\Big(\dfrac{x^j}{\lambda_j}\Big), \text{ if } \lambda_j > 0, \\ 0, \text{ if } \lambda_j = 0, \\ \infty, \text{ otherwise,} \end{cases}$$

and,

$$\Big(cl\ \check{F}^j\Big)(x^j, \lambda_j) = \begin{cases} \lambda_j F^j\Big(\dfrac{x^j}{\lambda_j}\Big), \text{ if } \lambda_j > 0, \\ F^{j'}_{\infty}, \text{ if } \lambda_j = 0, \\ \infty, \text{ otherwise,} \end{cases}$$

where $F^{j'}_{\infty}$ denotes the recession function of F^j (Ceria and Soares (1999)). The recession function of a function captures the asymptotic behavior of this function along a direction.

Theorem 3.1.2 (Ceria and Soares (1999))**.** *Let K be defined as in (3.1). A vector x belongs to K if and only if there exist vectors (x^j, λ_j) for every $j \in J$, such that the following nonlinear system is feasible*

$$x = \sum_{j \in J} x^i, \tag{3.2}$$

$$\Big(cl\ \check{F}^j\Big)(x^j, \lambda_i) \leq 0,\ j \in J, \tag{3.3}$$

$$\sum_{j \in J} \lambda_j = 1, \quad \lambda_j \geq 0, \tag{3.4}$$

where $\Big(cl\ \check{F}^j\Big)(x^j, \lambda_i)$ denotes the closure of the perspective mapping of F^j at (x^j, λ_i).

It is shown by Günlük and Linderoth (2010) and Frangioni and Gentile (2006) that, for some disjunctive sets of the following form (S), convex hull description can be given in the space of the original variables using the perspective function. The set (S) is a union of two sets corresponding to different values of an associated binary variable, say z.

$$\left\{(x, z) \in \mathbb{R}^n \times \{0, 1\} \quad \middle| \quad \begin{array}{l} x \in \Gamma_0, \text{ if } z = 0 \\ x \in \Gamma_1, \text{ if } z = 1 \end{array} \right\}. \tag{S}$$

If Γ_0 is a singleton set and Γ_1 is a bounded convex set, then the set S is called an indicator-induced $\{0, 1\}$ set or an 'on-off' set. The roles of $z = 0$ and $z = 1$ can be swapped without losing generality.

A binary variable z is said to control variables x, if by setting $z = 0$, x can be fixed to some vector $\hat{x}$, $\hat{x} \in \mathbb{R}^n$, and $z = 1$ implies that x lies in a compact convex set. Such variables x are called semi-continuous variables (Frangioni and Gentile (2006, 2009)). Every binary variable trivially controls itself. Semi-continuous variables appear in optimization models of many real-world problems like the synthesis of heat exchanger networks (Zamora and Grossmann (1998)), hydro-thermal unit commitment problems (Frangioni *et al.* (2008)), the portfolio selection model (Jobst *et al.* (2001)), etc. These variables take a fixed value representing a certain condition, otherwise, a value from some interval. For example, the variable x_{ij} in the uncapacitated facility location problem (UFL) presented in Chapter 1 is a semi-continuous variable. It takes the value 0 when facility i is closed (indicated by $z_i = 0$), and if the facility is open ($z_i = 1$), x_{ij} lies in the range $[0, 1]$.

Let the problem (P) contains disjunctions that are either an 'on-off' set or a set (S) in which Γ_0 is a ray. Suppose such disjunctions are replaced with their convex hull represented using the well-known perspective function (4.6) in the space of original variables. In that case, the resulting reformulation is called a perspective reformulation of (P).

In this chapter, we present structures in the form of a collection of constraints representing disjunctive sets that are amenable to perspective reformulation. We automatically detect and use these structures in a branch-and-cut framework for generating gradient inequalities to the nonlinear constraints in their convex hull description. These inequalities are called perspective cuts Frangioni and Gentile (2006)).

3.1.1 Sets of the Form (S)

We present sets that are amenable to perspective reformulation. If these sets are present in a problem, they can be replaced by their convex hull descriptions to obtain tighter relaxations. Sets discussed in this section can be viewed as generalizations of sets described by Günlük and Linderoth (2010).

1. This set is referred to as (S_1) and is given by $\Gamma_0 \cup \Gamma_1$ with

$$\left.\begin{aligned} \Gamma_0 &:= \{(x,z) \in \mathbb{R}^p \times \{0,1\} : x = \hat{x},\ z = 0\}. \\ \Gamma_1 &:= \{(x,z) \in \mathbb{R}^p \times \{0,1\} : g_i(x) \leq 0,\ i \in M',\ Ax \leq a, z = 1\}. \end{aligned}\right\} \quad (S_1)$$

Where, $\hat{x}$ is some fixed assignment of x, M' is an index set of nonlinear constraints, a is a vector, A is a matrix of appropriate dimensions such that the polyhedral set $Ax \leq a$ is compact, every g_i is a convex function, and p is a positive integer. Here, Γ_0 is a singleton and Γ_1 is a compact convex set.

From the first principle, the convex hull of the set (S_1), denoted as $conv(S_1)$, can be shown to be in the space of original variables x and z.

Lemma 3.1.3. *$conv(S_1) = \Gamma_0 \cup \tilde{S}_1$, where*

$$\tilde{S}_1 = \Big\{(x,z) \in \mathbb{R}^{p+1} : g_i\Big(\frac{x - (1-z)\hat{x}}{z}\Big) \leq 0,\ i \in M', \\ A(x - (1-z)\hat{x}) \leq az,\ 0 < z \leq 1\Big\}.$$

Proof. The extended reformulation of $conv(S_1)$ can be derived by taking convex combination of points $(x^0, z^0) \in \Gamma_0$ and $(x^1, z^1) \in \Gamma_1$ as

$$conv(S_1) = \Big\{(x,z,\lambda) \in \mathbb{R}^{p+2} : x = (1-\lambda)x^0 + \lambda x^1, \\ z = (1-\lambda)z^0 + \lambda z^1, \\ g_i(x^1) \leq 0,\ i \in M', \\ Ax^1 \leq a, z^1 = 1, \\ x^0 = \hat{x}, z^0 = 0, \lambda \in [0,1]\Big\}.$$

By substituting the values of x^0, z^0, and z^1, we get $z = \lambda$ and $x = (1-z)\hat{x} + zx^1$. Furthermore, x^1 can be substituted as $x^1 = \dfrac{x - (1-z)\hat{x}}{z}$ for $z > 0$. We define

$$\tilde{S}_1 = \Big\{(x,z) \in \mathbb{R}^{p+1} : g_i\Big(\frac{x - (1-z)\hat{x}}{z}\Big) \leq 0,\ i \in M', \\ A(x - (1-z)\hat{x}) \leq az,\ 0 < z \leq 1\Big\}.$$

Also, if $z = 0$, then $(x, 0) \in conv(S_1)$ if and only if $(x, 0) \in \Gamma_0$. □

Also, using results from Günlük and Linderoth (2008), Γ_0 can be shown to lie in the closure of the set $\tilde{S}_1$, denoted by $cl\,(\tilde{S}_1)$. Therefore, $conv(S_1) = cl\,(\tilde{S}_1)$. The function defining $\tilde{S}_1$ may not be convex even when g_i is convex. To ensure that

the nonlinear functions defining the set $\tilde{S}_1$ are convex and given that $z > 0$, we can write the following equivalence

$$g_i\Big(\frac{x-(1-z)\hat{x}}{z}\Big) \leq 0 \Leftrightarrow zg_i\Big(\frac{x-(1-z)\hat{x}}{z}\Big) \leq 0,\ i \in M'.$$

Thus, $\tilde{S}_1$ can be rewritten as

$$\tilde{S}_1 = \Big\{(x,z) \in \mathbb{R}^{p+1} : zg_i\Big(\frac{x-(1-z)\hat{x}}{z}\Big) \leq 0,\ i \in M', \\ A(x-(1-z)\hat{x}) \leq az,\ 0 < z \leq 1\Big\}.$$

2. This set, referred to as (S_2), is defined as $\Gamma_0 \cup \Gamma_1$, with

$$\left.\begin{aligned} &\Gamma_0 := \{(x,v,z) \in \mathbb{R}^{p+q} \times \{0,1\} : x = \hat{x}, d^T v \leq 0, z = 0\}. \\ &\Gamma_1 := \{(x,v,z) \in \mathbb{R}^{p+q} \times \{0,1\} : g_i(x) \leq 0,\ i \in M', g_j(x) + d^T v \leq 0, Ax \leq a, z = 1\}. \end{aligned}\right\} \quad (S_2)$$

Where, $\hat{x}$ is some fixed x, d, a are vectors and A is a matrix of appropriate dimensions, polyhedral set defined by $Ax \leq a$ is compact, and every g_i and g_j are convex functions such that $g_j(\hat{x}) \geq 0$. In this case, Γ_0 constitutes a half-space and Γ_1 is a compact convex set.

Lemma 3.1.4. *$conv(S_2) = \Gamma_0 \cup \tilde{S}_2$, where*

$$\tilde{S}_2 = \Big\{(x,v,z) \in \mathbb{R}^{p+q+1} : g_i\Big(\frac{x-(1-z)\hat{x}}{z}\Big) \leq 0,\ i \in M', \\ g_j\Big(\frac{x-(1-z)\hat{x}}{z}\Big) + \frac{d^T v}{z} \leq 0, \\ A(x-(1-z)\hat{x}) \leq az,\ 0 < z \leq 1\Big\}.$$

Proof. Convex hull of the set S_2, $conv(S_2)$, can be expressed in the extended space by taking convex combination of points $(x^0, v^0, z^0) \in \Gamma^0$ and $(x^1, v^1, z^1) \in \Gamma_1$ as

$$conv(S_2) = \Big\{(x,v,z,\lambda) \in \mathbb{R}^{p+q+2} : x = (1-\lambda)x^0 + \lambda x^1 \\ v = (1-\lambda)v^0 + \lambda v^1, \\ z = (1-\lambda)z^0 + \lambda z^1, \\ g_i(x^1) \leq 0,\ i \in M', \\ g_j(x^1) + d^\top v^1 \leq 0, \\ Ax^1 \leq a, z^1 = 1, \\ x^0 = \hat{x},\ z^0 = 0,\ d^\top v^0 \leq 0,\ \lambda \in [0,1]\Big\}.$$

By substituting the values of $x^0,\ x^1,\ v^1,\ z^0$ and z^1, we get $z = \lambda$, and for $z > 0$, we also get $x^1 = \dfrac{x-(1-z)\hat{x}}{z}$ and $v^1 = \dfrac{v-(1-z)v^0}{z}$. On substituting the values of x^1 and v^1 in the nonlinear constraints in the description of $conv(S_2)$, we get

$$g_j\Big(\frac{x-(1-z)\hat{x}}{z}\Big) + \frac{1}{z}d^\top v - \frac{1-z}{z}d^\top v^0 \leq 0. \tag{3.5}$$

Since, $d^\top v^0 \leq 0$, we have that $\Big(\dfrac{1-z}{z}\Big)d^\top v^0 \leq 0$ and the third term in (3.5) can be dropped. For $z = 0$, the point $(x, v, 0)$ belongs to $conv(S_2)$ if and only if $(x, v, 0)$ belongs to Γ_0. □

Like earlier, $conv(S_2)$ can be shown equivalent to the closure of the set $\tilde{S}_2$, denoted by $cl\,(\tilde{S}_2)$. Again, to ensure convexity of the nonlinear functions in the description of $\tilde{S}_2$, we multiply both the sides by z as it is positive, and obtain

$$\begin{aligned}\tilde{S}_2 = \Big\{(x,v,z) \in \mathbb{R}^{p+q+1} : & \ z g_i\Big(\frac{x-(1-z)\hat{x}}{z}\Big) \leq 0,\ i \in M', \\ & z g_j\Big(\frac{x-(1-z)\hat{x}}{z}\Big) + d^\top v \leq 0, \\ & A(x-(1-z)\hat{x}) \leq az,\ 0 < z \leq 1\Big\}.\end{aligned}$$

Also, the $conv(S_2) = cl\,(\tilde{S}_2)$.

3.1.2 Difficulty in Finding Sets (S_1) and (S_2)

Variables x in the sets (S_1) and (S_2) are semi-continuous variables. Given a convex MINLP of the form (P), finding semi-continuous variables and hence the disjunctive sets (S_1) or (S_2), is NP-hard in general. This is because, for every pair (x_i, x_j) of variables of interest, where $i \neq j,\ x_j \in \{0,1\},\ j \in \mathcal{I}$, the following two MINLPs (P_1 and P_2) have to be solved with $x_j = 0$ to check if x_i gets fixed to a value $\hat{x}_i$.

$$\left.\begin{array}{lll} \underset{x}{\text{minimize}} & x_i & \\ \text{subject to} & g_k(x) \leq 0, & k \in M, \\ & x_j = 0,\ x_k \in \mathbb{Z}, & k \in \mathcal{I}. \end{array}\right\} \tag{P_1}$$

Problem (P_2) is the same as (P_1) except that the objective sense is maximization. If the optimal values of these two MINLPs exist and are equal to say $\hat{x}_i$, then it means that $x_j = 0$ fixes x_i to $\hat{x}_i$. Since finding on-off sets can be as challenging as solving the original MINLP, there is a trade-off between the number of structures detected and the time spent finding them. A less time-consuming alternative is to find collections of constraints that conform with sets (S_1) and (S_2). As reported in the next section, such collections appear as small blocks in the mathematical formulation of many applications, as evident from certain benchmarking instances from MINLPLib (Bussieck *et al.* (2003)).

3.1.3 Structures Implying Semi-Continuous Variables

As mentioned earlier, finding whether a variable is semi-continuous or not requires solving two MINLPs, (P_1) and (P_2). This section presents some collections of constraints that indicate semi-continuous variables and binary variables controlling them. These collections may appear as small blocks in the problem (P). In this and the coming sections, we also use the notation z for binary variables that control other variables.

1. Collection, (C_1), of linear inequalities in at most two variables of the form,

$$\left.\begin{aligned} & l^1 z + l^0(1-z) \le x \le u^1 z + u^0(1-z), \\ & z \in \{0,1\},\ x \in \mathbb{R}, \end{aligned}\right\} \qquad (C_1)$$

   where $l^0, l^1, u^0, u^1 \in \mathbb{R}$ and $l^j \le u^j$, $j = 0, 1$. If $l^0 = u^0$, then x is a semi-continuous variable controlled by z. Similarly, if $l^1 = u^1$, then $(1 - z)$ controls x. A simple example of (C_1) that appears in many problems is

$$\begin{aligned} & lz \le x \le uz, \\ & z \in \{0,1\},\ x \in \mathbb{R}, \end{aligned}$$

   where $l^0 = u^0 = 0$, l, u are lower and upper bounds on x, and $\hat{x} = 0$.

2. Collection (C_2) of the following constraints

$$\left.\begin{aligned} & a^T x + d_1 z \le d_2, \\ & l \le x \le u, \\ & z \in \{0,1\}, \end{aligned}\right\} \qquad (C_2)$$

   where d_1 and d_2 are scalars, and $l, u \in \mathbb{R}^p$ with $l \le u$. For a variable x_i, if $a_i > 0$, then $l_i = 0$, and if $a_i < 0$, then $u_i = 0$.

   (a) When $d_2 = 0$ if $d_1 < 0$, every component of x is semi-continuous variable controlled by z such that $\hat{x} = 0$. If $d_1 > 0$, $z = 1$ is infeasible and therefore, z can be fixed to 0.

   (b) When $d_1 = d_2$, every component of x is semi-continuous variable controlled by $1 - z$ and $\hat{x} = 0$. However, if $d_1 < 0$, then $z = 0$ becomes infeasible and z can be fixed to 1. A simple example of (C_2) is

$$\begin{aligned} & \sum_{i=1}^{p} z_i \le 1, \\ & z \in \{0,1\}^p. \end{aligned}$$

   In this example, $(1 - z_i)$ controls all the other variables in the constraint.

3. Collection (C_3) of constraints that involves at least one of the previous two collections and an extra equality constraint.

$$\left.\begin{aligned} & d^T\overline{x} + d_3\tilde{x} = d_4, \\ & \tilde{x} \in \mathbb{R},\ \overline{x} \in C_1 \text{ or } C_2, \end{aligned}\right\} \qquad (C_3)$$

where $d \in \mathbb{R}^p$, and d_3 and d_4 are any scalars. If $\overline{x}$ is controlled by z or $(1-z)$, then so is $\tilde{x}$ if $\tilde{l} \leq \dfrac{d_4 - d^T\hat{\overline{x}}}{d_3} \leq \tilde{u}$, where $\tilde{l}$ and $\tilde{u}$ are lower and upper bounds respectively on $\tilde{x}$, otherwise, z can be fixed to 1 or 0, respectively.

The experimental setup used in this chapter is the same as in Section 2.1 in the Chapter 2. Out of 374 convex MINLP instances in MINLPLib (Bussieck *et al.* (2003)), 274 instances have at least one binary variable remaining after MINOTAUR's presolve routine. We refer to the set of these 274 instances as TS_b. Table 3.1 reports the number of instances in TS_b with above mentioned collections. While computing the number of semi-continuous variables in an instance, we do not count a binary variable that controls itself. We found that 220 instances in TS_b have at least one of collections ($C_1 + C_2 + C_3$). We refer to the set of these instances as TS_c. Test set TS_c is used for detecting structure amenable to PR. More details on instances in test set TS_c are presented in the Table D.1 in Appendix D.

type	# inst.	# inst. with semi-continuous variables ≥ 50%	≤ 10%
C_1	194	151	9
C_2	132	41	5
$C_1 + C_2$	220	203	0
$C_1 + C_3$	194	154	7
$C_2 + C_3$	132	43	5
$C_1 + C_2 + C_3$	220	208	0

Table 3.1: Summary of instances in test set TS_c that contain collections of type $C_i, i = 1, 2, 3$ indicating the presence of semi-continuous variables in them. The first column shows the type of collection. The second column reports the number of instances containing at least one collection of the type mentioned in the first column. In the last column, the first sub-column corresponds to the number of instances (out of the number of instances mentioned under the second column) in which at least 50% of the total number of variables are found to be semi-continuous. The second sub-column shows the number of instances in which the total number of semi-continuous variables is less than 10%.

3.1.4 Structures Amenable to Perspective Reformulation

A reformulation of the problem that results by replacing the on-off sets of the form (S_1) or (S_2) by their convex hull description (as mentioned in Lemma 3.1.3 and Lemma 3.1.4, respectively) is referred to as a perspective reformulation. In this section, given the problem (P), we present some structures that conform with sets of the form (S_1) or (S_2), and thus, are amenable to perspective reformulation.

1. A nonlinear constraint of the form

$$g_i(x) \leq 0, \tag{3.6}$$

conforms with (S_1) and thus, is amenable to perspective reformulation, if the following two conditions hold.

(a) variables x are semi-continuous and controlled by a binary variable x_j, $j \in \mathcal{I}$, and are related by constraints of the form (C_1) or (C_2) or (C_3).

(b)

$$g_i(\hat{x}) \leq 0. \tag{3.7}$$

If inequality (3.7) does not hold, then $x_j = 0$ is infeasible (there does not exist a feasible solution in which x_j takes the value 0) and x_j can be fixed to 1.

The overall structure can be written as

$$\left.\begin{aligned} g_i(x) &\leq 0, \\ (x, z) &\in C_k, \end{aligned}\right\} \tag{PS_1}$$

where $k \in \{1, 2, 3\}$.

An example of structure (PS_1) and its PR is

$$\begin{aligned} &e^{x_1+x_2} - 1000 \leq 0, \\ &z + 1 \leq x_1 \leq 1 + 5z, \\ &z + 2 \leq x_2 \leq 2 + 7z, \\ &z \in \{0, 1\}. \end{aligned}$$

Here, $e^3 < 1000$ therefore, the perspective reformulation is given by

$$\left.\begin{aligned} &ze^{\left(\frac{x_1+x_2}{z}\right)} - 1000z \leq 0, \\ &z + 1 \leq x_1 \leq 1 + 5z, \\ &z + 2 \leq x_2 \leq 2 + 7z, \\ &z \in [0, 1]. \end{aligned}\right\} \tag{PEx-1}$$

Remark 3.1.5.

(a) *Suppose all the semi-continuous variables result from collection (C_1). In that case, only the nonlinear constraint is modified, and the linear constraints remain unchanged in the reformulated problem.*

(b) *When the semi-continuous variables x and the associated binary variable z are defined using linear constraints $Dx \leq d$ (where d is a vector and D is a matrix of appropriate dimensions) such that $\hat{x} = 0$ and $d = 0$, then the linear constraints remain unchanged in the reformulated problem. For example, see (C_2) (2a).*

(c) *Structure (PS_1) can have additional linear constraints of the form $Dx \leq d$ and nonlinear constraints $g_k(x) \leq b_k, k \in M'$ such that $D(\hat{x}) \leq d$ and $g_k(\hat{x}) \leq b_k, k \in M'$, and PR is still applicable.*

2. In this structure, all variables in the nonlinear part of a nonlinear constraint function g_j are semi-continuous, and there is at least one variable in the linear part that is not controlled by the same binary variable x_j. Given a nonlinear constraint function g_j, let $\widetilde{g}_j$ and $\overline{g}_j$ denote nonlinear and linear parts in disjoint set of variables $\tilde{x}$ and $\overline{x}$, respectively. If z exists in the function g_j, it should be considered a part of $\tilde{g}_j$. This collection (PS_2) is defined as

$$\left.\begin{aligned} &\widetilde{g}_j(\tilde{x}) + \overline{g}_j(\overline{x}) \leq 0, \\ &(\tilde{x}, z) \in C_k, \end{aligned}\right\} \qquad (PS_2)$$

where $k \in \{1, 2, 3\}$ conforms with set (S_2) if $g_j(\hat{\tilde{x}}) \geq 0$ and the nonlinear constraint (3.6) is called amenable to PR. Remark 3.1.5 regarding the reformulated problem hold in this case as well.

An example of structure (PS_2) that may appear on reformulating a problem by moving the nonlinear objective to the constraint set using an auxiliary variable and its PR is

$$\begin{aligned} &e^{x_1+x_2} \leq \eta, \\ &z + 1 \leq x_1 \leq 1 + 5z, \\ &z + 2 \leq x_2 \leq 2 + 7z, \\ &z \in \{0, 1\}. \end{aligned}$$

Here, $e^3 > 0$ and therefore, the perspective reformulation is given by

$$\left.\begin{aligned} & ze^{\left(\frac{x_1 + x_2}{z}\right)} \leq \eta, \\ & z + 1 \leq x_1 \leq 1 + 5z, \\ & z + 2 \leq x_2 \leq 2 + 7z, \\ & z \in [0, 1]. \end{aligned}\right\} \tag{PEx-2}$$

Perspective reformulations of the structures (PS_1) and (PS_2) represent the convex hull descriptions of these sets and thus, contained by any convex nonlinear relaxation of these sets. Therefore, these reformulations are at least as tight as their continuous relaxations (result from relaxing the integer constrained variables in these sets).

3.1.5 Detecting Structures Amenable to Perspective Reformulation

Given a problem (P), we have a straightforward two-phase algorithm for detecting nonlinear constraints amenable to perspective reformulation. In the first phase, the algorithm first iterates through linear inequalities to find blocks of constraints (C_1) and (C_2). Then it iterates through all the linear equalities to detect (C_3). The outcome of the first phase is either a set of semi-continuous variables (and binary variables controlling them) or an indication that there are none. If there are semi-continuous variables, then in the second phase, the algorithm iterates through every nonlinear constraint and checks if it is amenable to PR. In case a nonlinear constraint conforms to either of the sets, it is declared amenable to perspective reformulation.

Our computational results on the test set TS_c show that 104 instances (all mixed-binary nonlinear programs) have structures amenable to perspective reformulation. We refer to the set of these 104 instances as TS_{pr}. Full details of the instances in TS_{pr} are shown in the Table D.2 in Appendix D. All instances (except `synthes2` and `synthes3`) in TS_{pr} have all nonlinear constraints amenable to perspective reformulation. Out of these, 103 instances have all *PR* amenable constraints of type (S_1), and the instance `synthes3` has one constraint each of type (S_1) and (S_2).

As this algorithm iterates through linear constraints for finding semi-continuous variables, it might take more time on instances with a large number of linear constraints. The time taken to detect structures amenable to perspective reformulation (including detection of semi-continuous variables) in any instance in the set TS_{pr} is negligible (less than half a second).

3.1.6 PR as a Presolving Technique

The identification of semi-continuous variables and constraints amenable to PR can also be viewed as a preprocessing step. We may also see some variable fixings as a byproduct of finding structures for PR. While detecting semi-continuous variables, collections (C_1), (C_2), and (C_3) are used to fix the value of a variable that is controlled by z or $1 - z$. When $z = 0$, if the lower bound and the upper bound on a variable $x \in \mathbb{R}$, say l and u respectively, are equal, then x is said to be controlled by z. If, however, $l > u$, then z can be fixed to 1. In the cases, $C_2 + C_3$ and $C_1 + C_2 + C_3$ as shown in Table 3.1, we found that a binary variable could be fixed to 0 or 1 in three instances `batch0812`, `batchdes`, and `batch`.

Another possibility of variable-fixing arises when a nonlinear constraint satisfies the first requirement but not the second (3.7) of the set (S_1). Although we do not see fixing of this kind in the instances in test set TS_c, it does not undermine such possibility.

Moreover, if a nonlinear constraint conforming with (S_1) is such that all of its variables are controlled by both z and $1 - z$, then the nonlinear constraint becomes redundant and can be removed from the problem. If the same holds for a nonlinear constraint conforming with (S_2), then the nonlinear constraint can be transformed to a linear one as

$$\overline{g}_j(\overline{x}) + (1 - z)\widetilde{g}_j(\tilde{x}^0) + z\widetilde{g}_j(\tilde{x}^1) \leq 0,$$

where $\tilde{x} = \tilde{x}^0$ when $z = 0$, and $\tilde{x} = \tilde{x}^1$ when $z = 1$.

3.1.7 Solving Perspective Reformulation

The main difficulty in solving a perspective reformulation problem arises (due to indivisibility by zero) at points where binary variables associated with PR amenable nonlinear constraints take the value 0. Therefore, the existing methods for solving the reformulated problem focus on handling the division by 0. One can solve the perspective reformulated problem (1) by ϵ-approximation method that perturbs the reformulated constraint to take care of indivisibility by 0 (Furman *et al.* (2020)) (2) as a second-order cone program (SOCP) where nonlinear inequalities in the reformulated problem can be written as equivalent second-order cone constraints and solved using efficient SOCP solvers Frangioni and Gentile (2009); Günlük and Linderoth (2010)) (3) using perspective cuts, which are outer-approximation cuts to the reformulated nonlinear constraints, introduced by Frangioni and Gentile (2006). We focus on the last method - solving the reformulated problem using perspective cuts - where we generate and automatically add perspective cuts in a branch-and-cut framework, particularly in the QG method. We also briefly discuss ϵ-approximation approach in the context of the structures of our interest.

Consider the nonlinear constraint (3.8) in the perspective reformulation of the structure (PS_1)

$$z g_i\Big(\frac{x-(1-z)\hat{x}}{z}\Big) \leq 0. \tag{3.8}$$

Let

$$h_i(x,z) = z g_i\Big(\frac{x-(1-z)\hat{x}}{z}\Big).$$

Outer-approximating the constraint in (3.8) at a point (x', z') with $z' > 0$ gives

$$h_i(x',z') + (s_1,s_2)^\top (x-x', z-z') \leq 0, \quad \forall (s_1,s_2) \in \partial h_i(x',z'). \tag{3.9}$$

Let Γ_1 be the set obtained corresponding to $z = 1$ in the structure (PS_1). For any fixed $x'' \in \Gamma_1$ and $z \in [0,1]$, $\partial h_i(x,z)$ remains constant on the line of the form $x = (1-z)\hat{x} + zx''$ and thus, linearizations obtained at all such points (x,z) are the same. This implies that the outer-approximation cut at (x',z') is the same as at one at the point $\Big(\frac{x'}{z'}, 1\Big)$ and is given by

$$x^\top s + z\Big(g_i\Big(\frac{x'}{z'}\Big) + s\Big(\hat{x} - \frac{x'}{z'}\Big)\Big) \leq s^\top \hat{x}, \;\; s \in \partial_x g_i\Big(\frac{x'}{z'}\Big). \tag{3.10}$$

The valid inequality (3.10) is called a perspective cut. It is shown by Günlük and Linderoth (2010) that perspective cuts are outer-approximation cuts to the constraints amenable to perspective reformulation. And adding infinitely many perspective cuts to the defining function in a structure amenable to perspective reformulation gives its convex hull. Also, note that all perspective cuts pass through the point $(\hat{x}, 0)$. Similarly, for a reformulated constraint corresponding to (PS_2), a perspective cut is given by

$$\tilde{x}^\top s + z\Big(\widetilde{g}_j\Big(\frac{\tilde{x}'}{z'}\Big) + \tilde{s}\Big(\hat{\tilde{x}} - \frac{\tilde{x}'}{z'}\Big)\Big) + \overline{g}_i(\overline{x}) \leq s^T \hat{\tilde{x}}, \;\; s \in \partial_{\tilde{x}} \widetilde{g}_i\Big(\frac{x'}{z'}\Big). \tag{3.11}$$

In a cutting-plane based method or a branch-and-cut framework, instead of working directly with the algebraic description of the reformulated constraints, one starts from its relaxation. This relaxation is iteratively strengthened by dynamically generating perspective cuts as the algorithm progresses.

3.1.8 Perspective Cuts in a Branch-and-Cut Framework

This section discusses when one should generate perspective cuts and at which points in the branch-and-cut framework of the QG algorithm. In the QG method, cuts are added at nodes where associated linear programs yield integer optimal solutions. Additionally, as discussed in Chapter 2, one tends to start with a tight relaxation at the root

node as the quality of relaxation at the early stages of the tree influences branching decisions and the overall size of the tree. Traditionally, the continuous relaxation solution at the root node, say (x^0, z^0), is used to create initial linear relaxation by linearizing the nonlinear constraints at (x^0, z^0). Here, z^0 represents the vector of binary variables associated with semi-continuous variables appearing in the structures amenable to PR. These linearizations (gradient inequalities or cuts) to the constraints active at (x^0, z^0) are supporting for P^c (the feasible region of the continuous relaxation of problem (P)). However, these inequalities may not support P^r (the feasible region of the continuous relaxation of the perspective reformulated problem). This scenario arises when z^0_j belongs to $(0, 1)$ for some j and satisfies the original nonlinear constraint but not the reformulated constraint. Moreover, this can also happen at other nodes yielding fractional solutions in the branch-and-bound tree.

We found that in 68 instances in test set TS_{pr}, at least one reformulated nonlinear constraint get violated at the root relaxation solution (x^0, z^0), and 20 of these instances have more than 50% of the reformulated constraints violated. This observation motivated us to generate tight perspective cuts for the reformulated problem from a point (x', z') that is not in P^r. We study the problem of generating perspective cuts from such a point (x', z') under the following two cases.

1. In this case, (x', z') lies in the P^r, feasible region of the continuous relaxation of the original problem (P), but not in the P^c, continuous relaxation of the reformulated problem. That is, $(x', z') \in P^c$ but $(x', z') \notin P^r$. An example of this case is when (x^0, z^0), the root relaxation solution, does not lie in P^r.

2. In this case, (x', z') does not lie in either of the sets P^c or P^r.

Given a $(x', z') \notin P^r$, we are interested in finding a point (x'', z'') such that (x'', z'') lies in P^r (or at least at the boundary of the violated constraint) and linearizations generated at (x'', z'') are tight and cut off (x', z'). We propose the following methods that are easy to implement and are computationally effective in obtaining such a point (x'', z'').

1. This method, named SimLS (stands for simple line search,) considers each violated constraint indexed j and search for a point that satisfies the reformulated constraint indexed j at equality. That is, given

$$z'_i g_j\Big(\frac{x' - (1 - z'_i)\hat{x}}{z'_i}\Big) > 0,$$

where z_i is the binary variable controlling variables x, this method finds a point (x'', z'') such that

$$z_i'' g_j\Big(\frac{x'' - (1 - z_i'')\hat{x}}{z_i''}\Big) = 0.$$

Given the point $(x', z') \in P^c$, if $(x', 1) \in P^r$, then (x'', z'') is such that $x'' = x'$ and $z''_i = (1 - \lambda)z'_i + \lambda$ for some $\lambda \in (0, 1]$.

Also, if $(\hat{x}, 1) \in P^c$ (and thus, in P^r), then for every $(x', z' \in P^c)$, $(x', 1) \in P^c$ (and thus, in P^r). Verifying $(\hat{x}, 1) \in P^c$ amounts to evaluating whether the nonlinear constraint satisfies at $(x', 1)$. Also, for the structure (PS_1), if the associated binary variable does not exist in the defining nonlinear constraint, then $(\hat{x}, 1) \in P^c$.

We found that in 98 instances in TS_{pr}, $(\hat{x}, 1) \in P^c$ for all the PR amenable constraints. In 50 out of these 98 instances, binary variables controlling semi-continuous variables do not appear in the constraint function. The 6 instances in which this condition is not satisfied for any of the PR amenable constraints are `clay*`.

2. This method, referred to as CenLS, uses an approximation of the center (x^C, z^C) of P^c, if (x^C, z^C) also lies P^r, continuous relaxation of the reformulated problem. To find (x^C, z^C), we solve the problem (NLPI) as in Chapter 2. If such a point (x^C, z^C) exists and lies in P^r, then we perform a line search between the given point and the center point (x^C, z^C) to obtain boundary point as the desired point (x'', z'').

Adding Perspective Cuts at Root Node

In the first set of computational experiments, we generate perspective cuts at only the root node in the QG algorithm. We categorize these under three settings *root_reg*, *root_cenls*, and *root_bothls*, based on the above methods for generating perspective cuts.

1. *root_reg*: In this setting, we generate a perspective cut to every nonlinear constraint violated at point (x^0, z^0) (solution to the root node) in the reformulated problem.

2. *root_cenls*: This setting generates perspective cuts to violated nonlinear constraints in the reformulated problem using CenLS method.

3. *root_bothls*: This setting generates additional perspective cuts using SimLS method, wherever applicable, in the setting *root_cenls*.

In all these experiments, we add gradient inequalities to nonlinear constraints defining structures amenable to PR at corresponding points $(\hat{x}, 0)$, where $z = 0$ sets $x = \hat{x}$. Collection of these cuts are termed as *initCuts*.

We compare the default implementation of QG in MINOTAUR, referred to as *qg*, to *qg* with the settings *root_reg*, *root_cenls*, and *root_bothls*. We experimented with a variant of the last two settings. In this variant, we incorporated additional perspective

cuts generated at (x^0, z^0) and obtained better performance than the corresponding original setting. We see further improvement by adding perspective cuts to constraints that were inactive at the root relaxation solution. If a nonlinear constraint in the reformulated problem is inactive at (x^0, z^0), then every direction from (x^0, z^0) is feasible concerning the region defined by the constraint. If $(x^0, 1)$ does not lie in P^r, then we find a point on the boundary by moving along direction $-e_z$, where e_z is a vector whose components associated with z is one and rest are 0. Thus, results reported for the settings *root_cenls* and *root_bothls* are from this final version. The performance measures used for comparison are overall solution time and size of the tree in terms of the number of nodes processed, as described in Section 2.2.

Each row of Table 3.2 corresponds to an experimental setting ($s \in$ {*root_reg*, *root_cenls*, *root_bothls*}). The column '# solved by' lists the number of instances solved to optimality within the time limit under setting s and by both the reference solver (*qg* in this case) as well as setting s. The first column under the headings 'time' and 'nodes' shows the shifted geometric mean (SGM) of these measures reported by the reference solver (*qg* in this case) for the instances solved by both. The second column under these headings show the relative SGM ('rel.') under setting s for the same instances. Similarly for Table 3.3. Table 3.2 and Table 3.3 present a comparison of default *qg* and *qg* with different settings s for adding perspective cuts at the root node of the tree of instances from test set TS_{pr}. Analysis is presented for instances in the test set TS_{pr} that are solved by all the methods compared. The once instance (`rsyn0830m04m`) that reached time limit by *qg* took under setting *root_reg* 56.22 s, *root_cenls* 53.93 s, and *root_bothls* 47.17 s.

Our computational results show improvements in both the considered measures under all three settings. The highest improvement is reported by *qg* with *root_bothls*. Overall, it improved the solution time and tree size by about 43.19% and 41.45%, respectively. Even higher improvements (about 81% and 95%for both the measures) are observed for instances in with default *qg* took more than 100 seconds and 500 seconds, respectively.

Furthermore, we use performance profiles Dolan and Moré (2002) that graphically demonstrate the relative performance of different solvers for a particular performance measure over a given set of instances. Let S be a set of solvers to be compared, I be a given set of instances, and $t_{i,s}$ be the solution time of instance $i \in I$ by solver s. The performance ratio $r_{i,s}$ of solver s on instance i compared to the best solver for this instance is given by

$$r_{i,s} = \frac{t_{i,s}}{\min\limits_{j \in S} t_{i,j}},$$

Table 3.2: (Top) Comparison of *qg* and *qg* with setting *s* on 103 instances in TS_{pr} that are solved by both the methods. (Bottom) Performance on instances (26 for *qg* and *qg* with first two settings, and 25 with *root_bothls*) that are solved by both but at least one method took more than 10 seconds.

	time		nodes	
setting (*s*)	*qg*	rel.	*qg*	rel.
root_reg	8.60	0.67	505.44	0.69
root_cenls	8.60	0.63	505.44	0.69
root_bothls	8.60	0.57	505.44	0.59

	time		nodes	
setting (*s*)	*qg*	rel.	*qg*	rel.
root_reg	67.23	0.48	14956.08	0.41
root_cenls	67.23	0.43	14956.08	0.41
root_bothls	72.39	0.34	17005.06	0.29

Table 3.3: (Top) Comparison of *qg* and *qg* with setting *s* on nine instances in TS_{pr} that are solved by both the methods but at least one method took more than 100 seconds. (Bottom) Similar comparison on two instances in TS_{pr} that are solved by both the methods, but at least one method took more than 500 seconds.

	time		nodes	
	qg	rel.	*qg*	rel.
root_reg	294.93	0.28	55751.27	0.24
root_cenls	294.93	0.24	55751.27	0.26
root_bothls	294.93	0.19	55751.27	0.19

	time		nodes	
	qg	rel.	*qg*	rel.
root_reg	1249.47	0.10	595448.0	0.09
root_cenls	1249.47	0.10	595448.0	0.09
root_bothls	1249.47	0.05	595448.0	0.04

and $\rho_s(\tau) : \mathbb{R} \rightarrow [0, 1]$, a cumulative distribution function for the performance ratio of solver *s*, is defined as

$$\rho_s(\tau) = \frac{|i \in I : t_{i,s} \leq \tau|}{|I|}.$$

$\rho_s(\tau)$ is a nondecreasing function indicating that solver *s* is at most τ times slower than the best solver on an instance. In particular, the value $\rho_s(1)$ gives the fraction of the instances on which a solver *s* performs the best. Figure 3.1 shows the performance profiles of *qg* and *qg* with settings *root_reg*, *root_cenls*, *root_bothls* using the solution times of the instances in test set TS_{pr}. It shows that on nearly 45% of these instances, *qgrs1* is faster than *qg* and two times faster on more than 10% of the instances. We use similar profiles for reporting the results of the other schemes in this section.

Adding Perspective Cuts at Other Nodes

In this second set of experiments, we generate perspective cuts at other nodes yielding integer feasible solutions in addition to the root node. The nodes we have selected for generating perspective cuts are the same as in default *qg* (the ones yielding integer optimal LP solution). But using the fixed-NLP solution as in QG algorithm may not produce tight

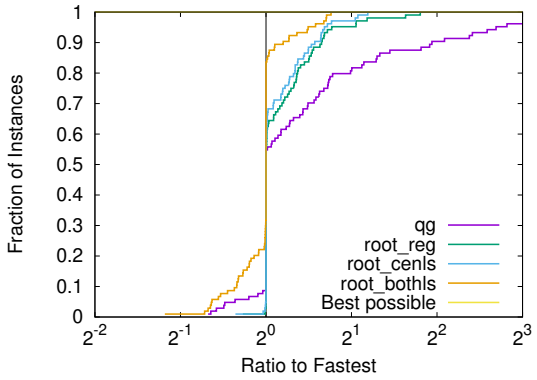


Figure 3.1: Performance profiles comparing solution times of *qg* and *qg* with settings *root_reg*, *root_cenls*, *root_bothls* on instances in test set TS_{pr}.

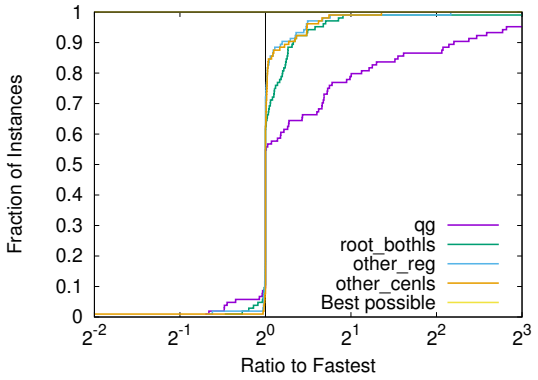


Figure 3.2: Performance profiles comparing solution times of *qg* and *qg* with settings *root_bothls*, *other_reg*, *other_cenls* on instances in test set TS_{pr}.

inequalities for the reformulated problem for the same reason as mentioned for the case of the root node. Here, we employ CenLS method for finding points for generating tighter perspective cuts. Let (x', z') be an integer solution of the fixed-NLP at any node. If the fixed-NLP is infeasible, (x', z') is a solution to the feasibility problem. When the fixed-NLP is optimal, a reformulated constraint is always feasible. However, some constraints could be inactive. In the test set TS_{pr}, in 6 instances, at least one reformulated nonlinear constraint is violated at some node yielding an integer optimal solution. In 10 instance, at least one reformulated constraint is inactive at the fixed-NLP optimal solution. Thus, keeping the best setting at the root node, we have performed the following computational experiments for adding perspective cuts at nodes that yield an integer optimal solution.

- *other_reg*: This setting generates perspective cuts to every nonlinear constraint in the reformulated problem at (x', z') when $z' \neq 0$.
- *other_cenls*: This setting employs CenLS method for generating perspective cuts.

We have experimented *other_cenls* method with and without adding constraints for inactive perspective amenable constraints in the same manner as in the root node. We found better results by not adding additional constraints for inactive constraints and thus report the same. Table 3.4 and Table 3.5 summarize the results of *qg* with these settings in comparison to the default *qg* on instances in the test set TS_{pr}.

Our computational results show improvements in both the considered measures under all the three settings. The highest improvement is reported by *qg* with *other_cenls*. Overall, it improved the solution time and the tree size by about 47.40% and 44.62%, respectively, but even higher improvements (around 82% and 95% for both the measures)

Table 3.4: (Top) Comparison of *qg* and *qg* with setting *s* on 103 instances in TS_{pr} that are solved by both the methods. (Bottom) Performance on instances (25 for *qg* and *qg* with first two settings, and 26 with *other_cenls*) that are solved by both but at least one method took more than 10 seconds.

	time		nodes	
setting (*s*)	*qg*	rel.	*qg*	rel.
root_bothls	8.60	0.57	505.44	0.59
other_reg	8.60	0.53	505.44	0.56
other_cenls	8.60	0.53	505.44	0.55

	time		nodes	
setting (*s*)	*qg*	rel.	*qg*	rel.
root_bothls	72.39	0.34	17005.06	0.29
other_reg	72.39	0.30	17005.06	0.26
other_cenls	67.61	0.31	15645.84	0.27

Table 3.5: (Top) Comparison of *qg* and *qg* with setting *s* on nine instances in TS_{pr} that are solved by both the methods but at least one method took more than 100 seconds. (Bottom) Similar comparison on two instances in TS_{pr} that are solved by both the methods, but at least one method took more than 500 seconds.

time		nodes	
qg	rel.	*qg*	rel.
294.93	0.19	55751.27	0.19
294.93	0.18	55751.27	0.17
294.93	0.19	55751.27	0.17

time		nodes	
qg	rel.	*qg*	rel.
1249.47	0.05	595448.0	0.04
1249.47	0.06	595448.0	0.05
1249.47	0.06	595448.0	0.05

are observed for instances in with default *qg* took more than 100 seconds and 500 seconds, respectively.

As discussed in Chapter 2, generating additional cuts at some fractional nodes may also reduce the size of the tree and solution time (Sharma *et al.* (2020b)). One can also add the perspective cuts at the fractional nodes in the tree. However, since we compare with traditional QG, we limit our PR related computational experiments to the root node and the nodes yielding integer optimal solutions only.

3.1.9 ϵ-Approximation of Perspective Reformulation

In this section, we briefly discuss the $\epsilon-$approximation method of perspective reformulation. Grossmann and Lee (2003) proposed to solve the perspective reformulated problem in disjunctive convex programming by using the following approximation of the reformulated nonlinear constraints

$$(z_i + \epsilon) g_i\Big(\frac{x - (1 - (z_i + \epsilon))\hat{x}}{z_i + \epsilon}\Big) \leq 0, \tag{Ref1}$$

where ϵ is a small tolerance value. For $z_i = 0$, (Ref1) reduces to $g_i(\hat{x}) \leq 0$. However, for $z_i = 1$, it gives $(1 + \epsilon) g_i\Big(\frac{x - \epsilon\hat{x}}{1 + \epsilon}\Big) \leq 0$. The smaller the value of ϵ, the closer the

approximated region is to the feasible region of the reformulated problem. But choosing a very small value of tolerance leads to numerical instability in terms of round-off errors. This approximation can be carried out in a similar way for the nonlinear constraint in the structure (PS_2).

An approximation proposed by Furman (2009) overcomes the difficulty at $z_i = 1$ in the previous approximation (Ref1) and is given by

$$((1-\epsilon)z_i + \epsilon)g_i\Big(\frac{x-(1-\epsilon)(1-z_i)\hat{x}}{(1-\epsilon)z_i+\epsilon}\Big) \leq 0. \tag{Ref2}$$

The approximation (Ref2) is the same as the original constraint for $z_i \in \{0,1\}$. Similarly, an approximation can be written for the structure (PS_2) by replacing z_i with $(1-\epsilon)z_i + \epsilon$.

Since, we assume that $g_i(\hat{x}) \leq 0$, and $(1-x_i)\epsilon \geq 0$, inequality (Ref2) can be tightened by modifying its right-hand side as

$$((1-\epsilon)z_i + \epsilon)g_i\Big(\frac{x-(1-\epsilon)(1-z_i)\hat{x}}{(1-\epsilon)x_i+\epsilon}\Big) \leq g_i(\hat{x})(1-z_i)\epsilon. \tag{3.12}$$

Inequality (3.12) is tighter than inequality (Ref2) for any $0 < z_i < 1$. Furman *et al.* (2020) describes the conditions under which the approximation (3.12) is applicable in general. However, for the structures (PS_1) and (PS_2) both the ϵ-approximations, (Ref1) and (Ref2), are directly applicable for solving the perspective reformulation.

3.2 Reformulation Based on Function Separability

Separability is a property of a function that allows the function to be written as a sum of functions with a disjoint set of variables. Every linear function, by its very definition, is separable. In Wright *et al.* (2009), a function $f : \mathbb{R}^n \to \mathbb{R}$ is called 'group separable' or separable if there exist functions $f^i : \mathbb{R}^{n_i} \to \mathbb{R},\ i = 1,\ldots,m$, such that

$$f(x) = \sum_{i=1}^{m} f^i(x^i), \tag{3.13}$$

where $x^i \in \mathbb{R}^{n_i}$, and f^i and f^j for $i \neq j$ do not have any variables in common. This means that f can be written as a sum of functions of disjoint set of variables. A function is fully separable if every f^i in (3.13) is a univariate function; and f is partially separable if some f^i are univariate functions and some are not. Fully and partially separable functions can be seen as special cases of group separable functions. A fully separable function is also referred to as completely additively separable function and a general group separable function is also called as partially additively separable function (Griewank and Toint (1982); Kronqvist *et al.* (2018b)).

In addition to the motivating example (exBall), it is shown by Tawarmalani and Sahinidis (2005) that if a convex function is decomposed into its convex sub-expressions then outer-approximating these decomposed components separately gives better approximation of the original function than outer-approximating the original function directly. Let $\hat{f} : \mathbb{R}^k \to \mathbb{R}^m$ and $\hat{g} : \mathbb{R}^m \to \mathbb{R}^n$ be vectors of functions such that $\hat{f}(x) = (\hat{f}_1(x), \ldots, \hat{f}_m(x))$ and $\hat{g}(u) = (\hat{g}_1(u), \ldots, \hat{g}_n(u))$. Let $\hat{f}_i(x)$ be convex for $i = 1, \ldots, m$, $\hat{g}_j(u)$ be convex for $j = 1, \ldots, n$, $I = \{1, ..., m\}$, and $I_L = \{i \in I | \hat{f}_i(x)$ is affine$\}$. Assume $\hat{g}_j(u)$, $\forall j$ is non-decreasing in the range of $\hat{f}_i(x)$ for each $i \in I \setminus I_L$. Let functions $\hat{h}_l = \hat{g}_l \circ \hat{f}, l = 1, \ldots, n$ be defined as the composition of $\hat{g}$ and $\hat{f}$, as $\hat{h}(x) = \hat{g}(\hat{f}(x))$. It is shown by Tawarmalani and Sahinidis (2005) that for each $l = 1, \ldots, n$, function $\hat{h}_l$ is convex and Theorem 3.2.1 shows that the set obtained by outer-approximating the functions $\hat{f}$ and $\hat{g}$ separately is a subset of the set obtained by outer-approximating $\hat{h}$ directly.

Theorem 3.2.1 (Tawarmalani and Sahinidis (2005))**.** *Consider a set of points x^j, $j = 1, \ldots, r$. Let*

$$H_1(\hat{h}) = \{(\gamma, x) : \gamma \geq \hat{h}(x^j) + \nabla\hat{h}((x^j(x - x^j)),\ j = 1, \ldots, r\},$$

and,

$$H_2(\hat{h}) = \begin{cases} (\gamma, x): & \gamma \geq \hat{g}(\hat{f}(x^j)) + \nabla\hat{g}(\hat{f}(x^j))(\phi - \hat{f}(x^j)),\ j = 1, \ldots, r, \\ & \phi_i \geq \hat{f}_i(x^j) + \nabla\hat{f}_i(x^j(x - x^j)),\ j = 1, \ldots, r,\ i \in I \setminus I_L, \\ & \phi_i = \hat{f}_i(x^j) + \nabla\hat{f}_i(x^j(x - x^j)),\ j = 1, \ldots, r,\ i \in I. \end{cases}$$

Then, $H_2(\hat{h}) \subseteq H_1(\hat{h})$.

Given $\hat{h}(x)$, determining if it is a composition of functions $\hat{g}$ and $\hat{f}$ with the properties as mentioned above is not easy. However, a relatively easy-to-identify case is where $\hat{g}$ is a linear function (of the form $g(u) = u$) and $\hat{f}$ is a convex separable function. From Proposition 3.2.2), it is clear that with $\hat{g}$ being linear and $\hat{f}$ being convex and separable, $\hat{h}(x) = \hat{g}(\hat{f}(x))$ is convex with $\hat{f}$ and $\hat{g}$ satisfying the required properties. Many of the instances in MINLPLib (Bussieck *et al.* (2003)) have functions of this form in their mathematical formulations.

Proposition 3.2.2. *If f is a convex and separable function as in (3.13), then every separable part $f^i(x^i), i = 1, \ldots, m$ of it is also a convex function.*

Suppose a nonlinear separable constraint is of the form

$$\sum_{i=1}^{m} f^i(x^i) \leq b, \tag{3.14}$$

where $b \in \mathbb{R}$ is a scalar. The reformulation of (3.14) utilizing the nonlinear function separability can be written as

$$\left.\begin{aligned} &\textstyle\sum_{i=1}^{m} \tilde{\gamma}_i \leq b, \\ &f^i(x^i) \leq \tilde{\gamma}_i,\ i = 1, \ldots, m, \\ &\tilde{\gamma}_i \in \mathbb{R}, i = 1, \ldots, m. \end{aligned}\right\} \tag{SepCon}$$

Practically, to detect separability of a nonlinear function, we use its computational graph representation. A computational graph of a nonlinear function represents the function as a directed acyclic graph (DAG) for computational purposes. Figure 3.3 shows an example of a computational graph of the nonlinear function $f = e^{x_1+x_2} + {x_2}^2 + {x_3}^4$.

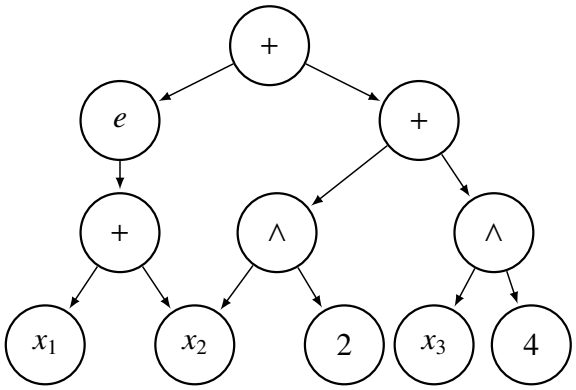


Figure 3.3: A computational graph of the nonlinear function $f = e^{x_1+x_2} + {x_2}^2 + {x_3}^4$.

A nonlinear separability can be detected using its Hessian matrix, but the computation of the Hessian matrix is expensive. On the other hand, a computational graph is readily available in many solvers, as it is used to store nonlinear functions, compute gradients and Hessians, etc. The computational graph we use is slightly different from the one used by Iri (1984) in algorithms for automatic differentiation, but is similar to the one used by Yamamura and Kiyoi (1992) in algorithms for identifying fully separable functions.

A computational graph of a nonlinear function f is constructed as a sequence of unary, binary, or other operations carried out on the input variables, constants, and intermediate variables, which themselves are created using these operations. Operations that we consider are binary operations (addition, subtraction, multiplication, and division), unary operations (ceiling, floor, root extraction, $(.)^k$, etc.), functions like transcendental, logarithmic and exponential, etc. Depending upon how intermediate variables are formed, a function can have multiple computational processes. Example (3.2.3) shows a computational process of a nonlinear function.

Example 3.2.3. *Given nonlinear convex function*

$$f = e^{x_1+x_2} + {x_2}^2 + {x_3}^4,$$

a computational process can be described as

$$v_1 = x_3^4,\ v_2 = x_2^2,\ v_3 = v_1 + v_2,$$
$$v_4 = x_1 + x_2, v_5 = e^{v_4},$$
$$f = v_3 + v_5.$$

Where, $v_i,\ i = 1, \ldots, 5$ *are intermediate variables such that*

$$v_i = \phi_i(v_{i_1}, \ldots, v_{i_n}),\ i_1, \ldots, i_n < i,$$

and, ϕ_i *is a basic operation on* $v_{i_1}, \ldots, v_{i_n}$ *that yields* v_i*.*

A node in a computational graph represents either a variable, a constant, or an operation. An edge e_{ij} from node i to node j implies that i is a parent of j, or j is an operand of operation represented by i, or j is a child of node i. A node with no child is called a leaf node or an independent node, and it represents either a constant or a variable. Other nodes are called dependent nodes. A node i that represents a binary operation has two child nodes- left ${n_i}^l$ and right ${n_i}^r$. A node i representing a unary operation has only one child, ${n_i}^l$. In expressions $a - b$ and $a \div b$, a is the left child and b is the right child of the node representing the operation $-$ or $\div$. Figure 3.4 shows an example of a node with two children. Let E_i^o denotes the set of edges originating from node i, and E_i^t denotes the set of edges terminating at i. Node i with $E_i^t = \emptyset$ is called the root node and there is only one root node in a computational graph. If $E_i^o = \emptyset$ then i is a leaf node. For an edge e_{ij}, let N_{ij}^o represents the origin node and N_{ij}^t represents the terminal node. In our implementation, a node representing a constant can have only one parent.

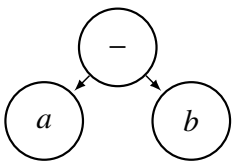


Figure 3.4: Left (a) and right (b) child nodes of node (-) representing the expression $a - b$.

Computational Subgraph

We introduce a notion of the computational subgraph that we use in finding separable parts in a computational graph of the given nonlinear function.

Let $G_f(C, E)$ be a computational graph of some nonlinear function f with C and E be the sets of all the nodes and edges, respectively. A graph $G_f^s(V, F)$ is called a subgraph of $G_f(C, E)$ if the following conditions hold.

1. $V \subseteq C$ and $F \subseteq E$.
2. For each $v \in V$, $E_v^o \in F$, and for each $e_{ij} \in F$, $N_{ij}^t \in V$ and $N_{ij}^o \in V$.
3. A node with no parent node does not represent binary operations + or −.
4. If we consider an undirected version of $G_f^s(V, F)$, then there exists a path between every pair of nodes $i,\ j\ \in V$.

A subgraph can have more than one node with no parent. Every computational graph is a subgraph.

We define 'maximal subgraph' as a subgraph that is not a part of any subgraph other than the original computational graph. Figure 3.5 shows the computational graph of a separable function depicted in Figure 3.3 and its maximal subgraphs. Following

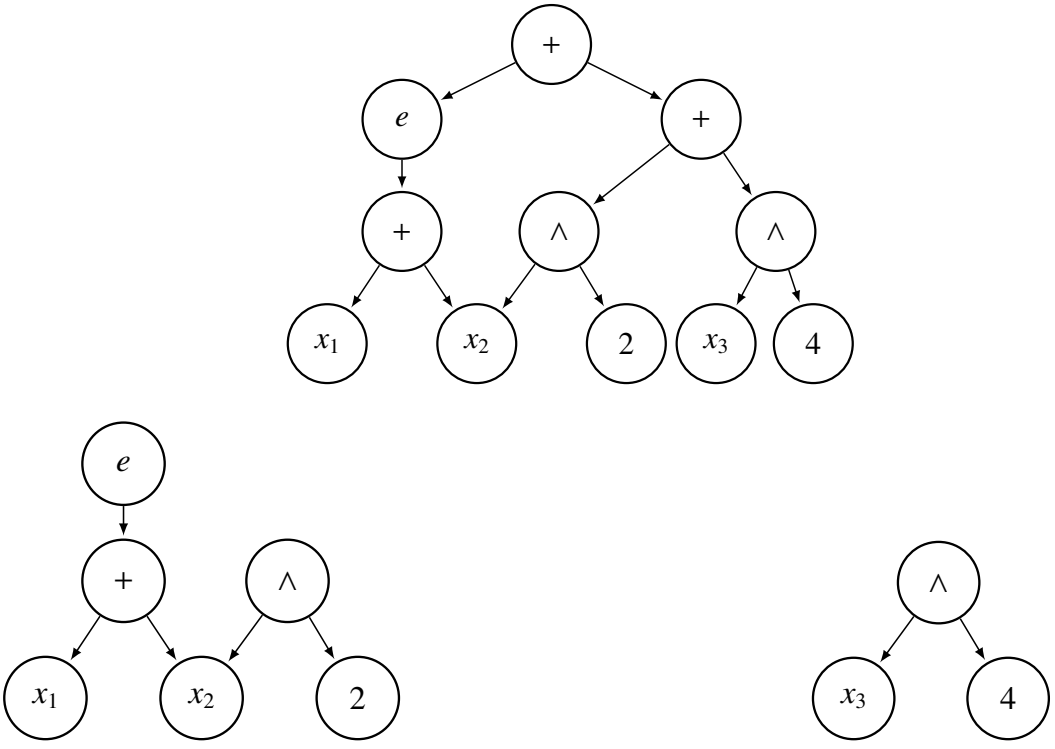


Figure 3.5: Computational graph (top) and maximal subgraphs (bottom) of $f = e^{x_1+x_2} + {x_2}^2 + {x_3}^4$.

Yamamura and Kiyoi (1992), we call a function implicitly separable if it is separable but is not evident from the given function expression. For example, $f = (x_1 + x_2 - x_2)^2 + {x_2}^2$ is separable in variables x_1 and x_2 but it is not evident until the first term in the expression is reduced to x_1^2. We call a function explicitly separable if its separability is not hidden in

the presence of other terms in the function expression. Let f be an explicitly separable function and let G_f be its computational graph.

Proposition 3.2.4. *The number of maximal subgraphs in G_f is equal to the number of separable parts in the function f and vice-versa.*

Proposition 3.2.5. *Function f is not separable if and only if G_f has only one maximal subgraph.*

3.2.1 Detection of Function Separability

Given a nonlinear convex function $f : \mathbb{R}^n \to \mathbb{R}$ and its computational graph G_f, checking whether f is not separable is more straightforward than checking otherwise. Thus, we start with employing simple rules to check if a function is not explicitly separable. If a function disobeys these rules, we use more extensive checking for separability.

Let r be the root node of G_f. Function f may not be separable if any of the below conditions is true:

1. r represents a unary operation.
2. r represents the binary operation $\times$, and both of its children represent either an operation or a variable.
3. r represents the binary operation $\div$, and its right child represents either an operation or a variable.

If r represents the operation $\times$ with any of its children representing a constant, we compute the tree rooted at the non-constant node and check again. Similarly, if r represents $\div$ with the right child representing a constant, we further analyze the graph rooted at its left child.

When a function fails to meet the above conditions, we search for maximal subgraphs in the computational graph G_f. If the number of maximal subgraphs is one, then the function is not separable. Otherwise, it is separable in as many parts as the number of maximal subgraphs. In this regard, we iteratively traverse G_f starting from node r and construct a list O of nodes that are either themselves or have children that are root nodes of some maximal subgraph. If a node $i \in O$ represents an operation other than binary operations $+$ and $-$, it is a root node of a maximal subgraph. Thus, a graph rooted at i belongs to a separable part, and an iteration of the algorithm amounts to traversing the graph rooted at such a node i. The graph rooted at a node i is a graph $G(C_i, E_i)$ in which:

1. i is the root node,

2. for each $k \in C_i$, $E_k^o \subseteq E_i$,

3. for each $e \in E_i$, $N_e^t \in C_i$.

While traversing the graph rooted at node i, if a node k is encountered that is already traversed in some previous iteration l, we do not traverse the graph rooted at k. Instead, we merge the information from iteration l with that in the current iteration j, as they belong to the same separable part. Every iteration j has a number m_j associated with it. This number represents the iteration number to which iteration j is merged to. Merging the iterations prevents revisiting the graph rooted at a node. At iteration j, N_j represents the set of nodes that are root nodes of maximal subgraphs. The number v_j represents the number of nodes representing the variables found while traversing the graph rooted at nodes in N_j. Merging iteration l to j means updating $N_j = N_j \cup N_l, v_j = v_j + v_l$, $N_l = \emptyset$, and $v_l = 0$. If v_j is equal to the number of variables in the function f, then f has only one maximal subgraph. In this case, the algorithm terminates with the output that the function is not separable. Otherwise, the algorithm gives the number of separable parts: the number of nonempty sets N_j. If $N_j = \emptyset$, it means that the iteration j is merged to some other iteration.

3.2.2 Some Implementation Details

As our algorithm relies on the computational graph of the function to detect its separability, we prefer to have a computational graph representing the function expression as explicitly as possible; sometimes at the expense of some additional computational effort. Different separable function expressions (in an explicit form) that can be identified by our algorithms are (1) $a \times (\sum_{i=1}^m f^i(x^i))$ (2) $\sum_{i=1}^m a_i \times f^i(x^i)$ (3) $\frac{\sum_{i=1}^m f^i(x^i)}{a}$ (4) $\sum_{i=1}^m \frac{f^i(x^i)}{a_i}$ (5) $(\sum_{i=1}^m f^i(x^i))^1$, where $a,\ a_i,\ \ b \in \mathbb{R}$.

A constraint of the form

$$\sum_{i=1}^{m} a_i \times f^i(x^i) \leq b,$$

is reformulated as

$$\sum_{i=1}^{m} a_i \gamma^i \leq b,$$
$$f^i(x^i) \leq \gamma^i,\ i = 1, \ldots, m.$$

And, a constraint of the form

$$a \times (\sum_{i=1}^{m} f^i(x^i)) \leq b,$$

is reformulated as

$$\sum_{i=1}^{m} a\gamma^i \leq b,$$
$$f^i(x^i) \leq \gamma^i,\ i = 1, \ldots, m.$$

Similar reformulations are applied to the other forms. This is a default reformulation under separability detection in MINOTAUR. However, in some instances of the benchmarking library, we found that different separable constraints have common separable parts (f^i). For example, if there are two constraints of the form

$$a_1 f^1(x^1) + a_2 f^2(x^2) \leq b_1,$$
$$d_1 f^1(x^1) + d_2 f^3(x^3) \leq b_2,$$

then, we reformulate them as

$$a_1\gamma_1 + a_2\gamma_2 \leq b_1,$$
$$d_1\gamma_1 + d_2\gamma_3 \leq b_2,$$
$$f^1(x^1) \leq \gamma_1,$$
$$f^2(x^2) \leq \gamma_2,$$
$$f^3(x^3) \leq \gamma_3.$$

Our implementation reuses variables corresponding to different separable parts in other constraint expressions, thus avoiding to create an extra variable and a constraint.

In MINOTAUR, we carry out separability detection before the presolving step. We have found that 126 instances have at least one separable nonlinear function (either in constraint or objective) out of 374 convex MINLP instances in the benchmarking library. Out of 126 such instances, 79 have separability only in the nonlinear objective function. In the remaining 47 instances, 45 instances have all the nonlinear constraints separable, and 2 have 40% of the nonlinear constraints with separability property. Also, 108 out of these 126 instances have at least one integer constrained variable. These 108 instances constitute our test set, TS_{sep}, for analyzing the impact of exploiting separability in the QG method. More details on these instances are provided in the Table D.3 in Appendix D. Using the same performance measures as before, Table 3.6 reports a comparison of default *qg* and *qg* with separability based reformulation on instances in test set TS_{pr}. Overall, we achieve about 40% improvement in the solution time and the tree size; even better improvements are seen in difficult instances. Using this reformulation, *qg* could solve 8 extra instances that reached the time limit earlier with default *qg*.

Table 3.6: Comparison of *qg* and *qgsep* (*qg* using separability based reformulation) on instances in test set TS_{sep}.

		time		nodes	
time	# of inst.	*qg*	rel.	*qg*	rel.
>= 0	76	13.68	0.60	700.04	0.50
>= 10	26	94.54	0.42	5902.23	0.32
>= 100	11	714.61	0.20	24311.92	0.28
>= 500	7	1900.22	0.12	54480.5	0.26

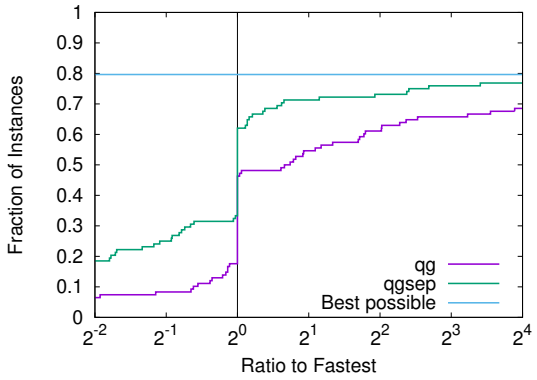


Figure 3.6: Performance profiles comparing solution times of *qg* and *qgsep* on instances in the test set TS_{sep}.

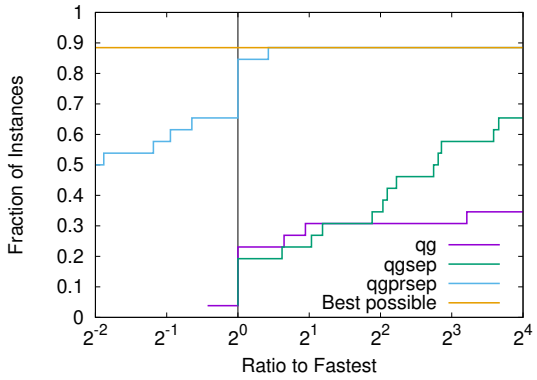


Figure 3.7: Performance profiles comparing solution times of *qg*, *qgsep*, and *qgprsep* on instances in the test set TS_{ps}.

3.3 Combined Effects of the Two Reformulations

In some cases, when a problem (P) is reformulated using function separability, we may get structures that are amenable to perspective reformulation. Let us reconsider the uncapacitated facility relocation problem (UFL) introduced in Chapter 1. This problem can be reformulated by moving the nonlinear objective to the constraint set using an auxiliary variable η as

$$\left.\begin{array}{lll} \underset{x,\, z,\, \eta}{\text{minimize}} & \eta & \\ \text{subject to} & \sum_{i\in\mathcal{F}} c_i z_i + \sum_{i\in\mathcal{F}, j\in C} t_{ij} x_{ij}^2 \le \eta, & \\ & 0 \le x_{ij} \le z_i,\ i \in \mathcal{F},\ j \in C, & \\ & \sum_{i\in\mathcal{F}} x_{ij} = 1,\ j \in C, & \\ & x_{ij} \ge 0, z_i \in \{0,1\},\ i \in \mathcal{F},\ j \in C. & \end{array}\right\} \qquad (UFL_1)$$

The function in the nonlinear constraint is separable and on reformulating (UFL_1), we get

$$\left.\begin{array}{ll} \underset{x,\, z,\, \eta}{\text{minimize}} & \eta \\ \text{subject to} & \sum_{i\in\mathcal{F}} c_i z_i + \sum_{i\in\mathcal{F}, j\in C} t_{ij}\gamma_{ij} \leq \eta, \\ & x_{ij}^2 \leq \gamma_{ij}, i \in \mathcal{F},\ j \in C, \\ & 0 \leq x_{ij} \leq z_i,\ i \in \mathcal{F},\ j \in C, \\ & \sum_{i\in\mathcal{F}} x_{ij} = 1,\ j \in C, \\ & x_{ij} \geq 0, z_i \in \{0,1\},\ i \in \mathcal{F},\ j \in C. \end{array}\right\} \quad (UFL_2)$$

Clearly, (UFL_2) now has structures of the form (PS_2) and thus, becomes amenable to perspective reformulation.

We observed that in the test set TS_{sep}, on reformulating using separable nonlinear functions, 26 instances have become amenable to perspective reformulation. These instances comprise our test set TS_{ps} and are reported in Table D.3 in Appendix D. We solved instances in TS_{ps} with default *qg*, *qgsep* (*qg* with separability based reformulation), and *qgprsep* (*qg* with both separability and perspective reformulations), and results are reported in Table 3.7 and Table 3.8 . Overall, there is a significant improvement of about 88% in both the solution times and the tree size, and 4 more instances that reached the time limit with even separability based reformulation could now be solved.

Table 3.7: (Top) Comparison of *qg* and methods (M) (*qg* with reformulation) on 15 instances in TS_{ps} that are solved by both the techniques. (Bottom) Performance on ten instances that are solved by both, but at least one technique took more than 10 seconds.

	time		nodes	
Method (M)	*qg*	rel.	*qg*	rel.
qgsep	78.24	0.32	1213.03	0.42
qgprsep	78.24	0.12	1213.03	0.12

	time		nodes	
Method (M)	*qg*	rel.	*qg*	rel.
qgsep	251.01	0.22	4418.17	0.31
qgprsep	251.01	0.07	4418.17	0.06

3.4 Conclusions

Our study concludes that perspective reformulation and exploitation of separability of nonlinear constraint functions help generate better polyhedral-approximations of the fea-

Table 3.8: (Top) Performance break-up on instances solved by both the methods (M) and *qg*, but where at least one of them took more than 100s (Top) and 500s (Bottom).(Top) Comparison of *qg* and methods (M) on instances in TS_{ps} that are solved by both the techniques, but at least one took more than 100 seconds. (Bottom) Similar comparisons using instances that are solved by both the techniques, but at least one took more than 500 seconds.

# solved	time		nodes	
by both	*qg*	rel.	*qg*	rel.
7	654.78	0.15	10605.48	0.27
5	1454.86	0.01	15711.76	0.02

# solved	time		nodes	
by both	*qg*	rel.	*qg*	rel.
4	2389.38	0.04	27068.51	0.15
4	2389.38	0.01	27068.51	0.01

sible region. We see improvement in both the solution time and the size of the tree in the branch-and-cut framework of the QG method on our test instances. Using the proposed schemes for generating perspective cuts at the root node, we reported around 10% improvement in the solution times and tree size over default *qg*. Overall, about 45% and 40% enhancements are observed in the considered performance measure with PR and exploiting separability, respectively; more improvements are recorded for difficult instances. Reformulation based on separability solved six more instances than the default *qg* setting. Test instances that became amenable to PR after separability based reformulation are solved even faster. About 20% improvement is recorded in both the considered performance measures over *qgsep* (*qg* with separability) and around 88% over default *qg*. Out of 26 such instances, *qg* solved 15, *qgsep* 19, and *qgprsep* (*qg* with PR and separability) 23.

Chapter 4

Mixed-Integer Partial Differential Equation Constrained Optimization

In this chapter, we study a problem from a highly challenging class of optimization problems called mixed-integer partial differential equation constrained optimization (MIPDECO). The combination of discrete optimization variables and PDE constraints presents difficult problems, providing a range of new mathematical challenges; see, for example, Leyffer *et al.* (2013). These challenges arise out of the combination of the combinatorial complexity of integer variables and the computational difficulties of the discretized PDE. Little is known about this class of problems or solution approaches, and one motivation of this work is to experiment with state-of-the-art mixed-integer solvers for solving this class of problems.

We consider an inverse problem in which the optimization variables are discrete and constrained by partial differential equations (PDEs), specifically in our case to describe convection-diffusion. We are interested in determining the number and location of a set of sources by reconciling the difference between measurements and numerical prediction of the concentration. Our work is motivated by applications in groundwater flow, where we want to find the location of pollutants in the subsurface; see, for example, Ozdogan (2004) and Fipki and Celi (2008), for more detailed background. We chose the steady-state convection-diffusion equation as our model problem because it allows us to solve the resulting mixed-integer PDE-constrained optimization (MIPDECO) problems in a reasonable amount of time. We stress, however, that our results and approaches generalize to time-dependent convection-diffusion processes. This work is done with Dr. Sven Leyffer (Argonne National Laboratory, USA), Prof. Lars Ruthotto (Emory University, USA), and Dr. Bart van Bloemen Waanders (Sandia National Laboratory, USA).

We advance the state of the art in MIPDECO in a number of ways. First, we develop new rounding schemes that take the physics of the problem into account by preserving the mass of the source when we move from a relaxation to a rounded solution. Second, we apply a simplified version of the trust-region method (Hahn *et al.* (2020)), and show that it already provides competitive integer solutions. Third, we improve the trust-region approach by developing a new problem-specific neighborhood that takes the topology of our problem into account, and we use a specialized knapsack solve for the resulting trust-region subproblem. Using the modified trust-region method we show that we can solve 3D instances of MIPDECO efficiently and in a reasonable amount of time, and we provide our Julia (Bezanson *et al.* (2012)) code under a permissible open-source license.

In the next section, we provide background on PDECO that is relevant to our developments, then we discuss the challenges of MIPDECO in more detail.

4.1 Background

Mixed-integer PDE-constrained optimization brings together complicated elements from two algorithmic areas to solve relevant application problems. In this section, we provide a general overview to highlight the most relevant features needed to introduce MIPDECO.

MINLPs are a challenging class of problems in their own right: they are in general NP-hard (Kannan and Monma (1978a)) and in the worst case undecidable (Jeroslow (1973)). Adding PDE-constraints leads to a range of computational and conceptual challenges for MINLPs. First, the computational expense of solving PDE-constrained optimization problems is much higher than the computational expense of solving standard NLP. Second, the number of optimization variables is typically very large, and if the integer variables are functions defined over the computational domain, then the number of integer variables typically also grows as we refine the discretization, resulting in huge combinatorial search spaces. Third, the solutions of the relaxations of the PDE-constrained optimization problem are typically only locally optimal and do not provide valid lower bounds for nonlinear PDE-constrained optimization problems.

4.1.1 PDE-Constrained Optimization

PDE-constrained optimization (PDECO) refers to the optimization of systems governed by partial differential equations. In most cases the goal is to optimize an objective function with respect to a quantity that is defined on subregions or everywhere in the computational domain. The inversion for initial conditions and the reconstruction of material properties are examples of typical optimization problems.

The PDE-constrained optimization problem is an infinite-dimensional optimization problem, and two approaches exist for obtaining a finite-dimensional approximation: the optimize-then-discretize approach and the discretize-then-optimize approach. In the former, one finds the necessary optimality conditions of the PDECO in function spaces and then discretizes this system of equations. In the latter, one first discretizes the PDE and then uses nonlinear optimization techniques to solve the large-scale optimization problem. Both approaches lead to an optimization problem with a large number of variables and a large system of equations (the discretized PDEs) that describe the underlying physics. The large-scale nature of these problems dictates the use of efficient sensitivities (adjoints), Newton based methods to handle the nonlinearity of the optimization formulation, the coordination of globalization, and the use of parallel matrix-vector operators to address the computational requirements (Nocedal and Wright (2000); Biegler *et al.* (2001, 2007)). The combination of these technologies poses formidable challenges to achieve efficient and accurate solutions. Considerable research and development have been conducted; the interested reader is referred to (Vogel (1999); Ascher and Haber (2001); Haber and Ascher (2001); Vogel (2002); Laird *et al.* (2005); Hintermuller and Vicente (2005); Hazra and Schulz (2006); Borzi (2007); Hinze *et al.* (2009)). Advances have been made to accelerate the convergence of these algorithms, with recent examples in special preconditioners, reduced-space methods, full-space algorithms, and multigrid approaches (Biros and Ghattas (2005a,b); Akcelik *et al.* (2005); Bartlett *et al.* (2006); Heinkenschloss and Ridzal (2008); Borzì and Schulz (2009); Simon (2008)).

A full-space solution algorithm forms the Lagrangian function and takes variations with respect to the state variables, the adjoint variables, and the optimization variables. This approach results in the first-order optimality conditions, which form a nonlinear system of equations. This system is typically solved by using Newton's method, requiring the solution of large structured systems of equations.

Alternatively, a reduced-space method eliminates the state variables by using the discretized PDE, resulting in an optimization problem in the optimization or control variables only, where the effect of the states in represented implicitly. The gradient of the objective function can be computed via the chain rule; and the solution process consists of an iterative process in which forward, adjoint, and gradient equations are successively solved. Even though there are advantages to using the full-space methods, in particular when the solution of the forward solve is slow to converge, we use the reduced-space method here because it has advantages in the mixed-integer case, resulting in an easy-to-solve subproblem.

PDE-constrained optimization problems with integer optimization variables have been solved with PDE-constrained optimization methods. Topological optimization is an example where the discrete optimization variables are approximated with continuous variables with the unfortunate consequence of errors (nonintegral values, or gray areas) at the boundaries of the topological solution. Although excellent practical results are obtained for a host of problems, establishing rigorous optimality proofs for this approach remains a challenge (Bendsøe and Sigmund (2004); Sigmund and Maute (2013a)).

4.1.2 MIPDECO

Practical approaches to MIPDECO must tackle the challenges posed by the number of integer variables in the discretized problem and the computational complexity of solving PDECO problems. We briefly discuss how recent approaches to MIPDECOs tackle these challenges.

The two classical approaches for solving PDECO problems are optimize-then-discretize and discretize-then-optimize. We do not believe that it is possible to apply the optimize-then-discretize to obtain first-order conditions for MIPDECOs, because such a generalization would also imply a set of first-order conditions for the (global) optimality of integer solutions for MINLPs, which seems unlikely. Hence, we consider only the discretize-then-optimize approach as a practical way to solve MIPDECOs.

If the integer controls are functions over the computational domain, then the discretization of the PDE and the controls results in MINLPs with a large number of integer variables, which has implications for standard MINLP solvers. Current state-of-the-art solution methods for MINLP employ a branch-and-bound tree search (Belotti *et al.* (2013); Bonami *et al.* (2008a)) at some stage of the solution process and the large number of integer variables arising in discretized MIPDECOs means that this tree can become huge, even for coarse discretization levels. In some cases, the tree search can be customized for solving discretized MIPDECO by using problem specific branching rules; see, for example, Hahn *et al.* (2017), which introduces a new branching rule and backtracking strategy that work well for time-dependent control problems.

Another solution approach for discretized MIPDECO is penalty based methods, which avoid the branch-and-bound tree altogether by penalizing the violation of integrality. The resulting nonlinear optimization problem is solved iteratively for an increasing penalty parameter, until all integer variables are integral. Unfortunately, the penalty is in general nonconvex, and stationary points of the nonlinear optimization problem correspond only to feasible points of the discretized MIPDECO without any optimality guar-

antees. See Costa *et al.* (2016), Lucidi and Rinaldi (2013), and Garmatter *et al.* (2019) for penalty based methods in the context of MINLP and MIPDECO.

There are other notable solution approaches for MIPDECOs that avoid a tree search, including decomposition methods that solve a relaxed form of the original problem and approximate the effect of the relaxed optimal control with that of an integer-valued one. For ODE- and DAE-constrained problems, heuristics such as sum-up rounding (SUR) (Sager (2006, 2009)) and next-forced rounding (NFR) (Jung (2013)) can produce arbitrarily good approximations of the optimal relaxed behavior given sufficiently high grid resolutions (Gerdts and Sager (2012); Sager *et al.* (2012)). These heuristics are special cases of the combinatorial integral approximation (CIA) approach (Sager *et al.* (2011)), which formulates the approximation problem as an MILP. Efficient problem-specific tree based solvers have also been developed to solve the CIA problem directly (Bürger *et al.* (2020); Jung *et al.* (2015)).

Early extensions of CIA heuristics to PDE-constrained problems (Hante and Sager (2013); Hante (2017)) limited themselves to rounding purely time-distributed controls where the arrow of time can be exploited as part of rounding heuristics such as SUR and NFR. More recent results, however, have shown that SUR can be applied to elliptic PDEs with spatially distributed controls by imposing an order through space-filling Hilbert curves (Manns and Kirches (2020, 2018, 2019)), though other orders have also been used successfully. Rounding methods for topology optimization problems are also explored in Garmatter *et al.* (2019).

These methods are collectively based on the recognition that spatially distributed integer-valued controls are not truly discrete, but form a continuum. This is explored in Hahn *et al.* (2020), where a trust-region steepest-descent method for binary optimal control is developed that is closely related to our approach. The authors show convergence to first-order stationarity in a topological sense, provided that the mesh is refined. This is a remarkable result because it replaces the combinatorial challenge of MIPDECO by a set based approach. This method avoids the combinatorial complexity of the tree search and instead solves a sequence of knapsack problems that can be interpreted as a local improvement strategy. A similar model to ours has been studied in Guo *et al.* (2019), where pollution sources are identified and a genetic algorithm heuristic is employed to resolve the integrality restrictions.

4.2 Mathematical Formulation

We formulate our constrained source inversion problem as a mixed-integer PDE-constrained optimization problem with binary inversion parameters. We discretize the PDEs with finite elements and present the resulting finite-dimensional formulation. We also present an alternative finite-difference discretization that provides self-contained AMPL models to run state-of-the-art MINLP solvers.

4.2.1 Variational Formulation of Source Inversion Problem

The goal of the inverse problem is to estimate the source w from measurements b assuming that the properties of the PDE are known and assuming a sparse set of sensors. The problem can be written as

$$\left.\begin{array}{lll} \underset{u,w}{\text{minimize}} & \mathcal{J}(u,w) = \frac{1}{2\sigma}\sum_{i=1}^{m} \|\langle p(r_i), u\rangle - b_i\|^2 + \alpha\mathcal{R}(w) & \\ \text{subject to} & -c\Delta u + v^\top \nabla u = w, & \text{in } \Omega, \\ & \frac{\partial u}{\partial n} = 0, & \text{on } \Gamma_N, \\ & u = g, & \text{on } \Gamma_D, \end{array}\right\} \tag{VF}$$

where $c > 0$ is the diffusion coefficient, $v : \Omega \to \mathbb{R}^d$ is the velocity vector, and $w : \Omega \to \{0, 1\}$ represents the source terms. The boundary of the domain, Γ, is partitioned into Γ_N and Γ_D for Neumann and Dirichlet conditions, respectively, $n : \Gamma \to \mathbb{R}^d$ denotes the outward normal vector; and $g : \Gamma_D \to \mathbb{R}$ defines the Dirichlet condition. Let $\Omega \subset \mathbb{R}^d$ denote the computational domain, where in this work $d = 2, 3$. Discrete measurements $b \in \mathbb{R}^m$ are given as

$$b_i = \langle p(r_i), u\rangle + \epsilon_i, \quad i = 1, 2, \ldots, m, \tag{4.1}$$

where u is the concentration of the pollutant, $p(r_i) : \Omega \to \mathbb{R}$ are the receiver functions at the locations r_i for $i = 1, \ldots, m$, $\langle \cdot, \cdot \rangle$ denotes the $\mathcal{L}_2$ inner product, and $\epsilon_1, \epsilon_2, \ldots, \epsilon_m$ represent measurement noise. To model point measurements of the PDEs, we consider the receiver function $p(r)$ to be a Dirac δ-function centered at r. In our numerical demonstrations, we generate the sensor data synthetically and assume that the measurement noise is independent and identically distributed (iid) Gaussian noise with constant, known standard deviation $\sigma > 0$.

$\mathcal{R}$ is a regularization functional that promotes the existence and regularity of solutions. This is important because the inverse problem is underdetermined and ill-conditioned as a result of data sparsity and noise. The term $\mathcal{R}$ can also be used to penalize undesirable features. The regularization parameter $\alpha > 0$ balances between fitting the data (for small values of α) and ensuring regularity of the solution (for large values of α).

Choosing an "optimal" α, is both crucial and nontrivial. No general rule exists for picking α; however, strategies using generalized cross validation (Golub *et al.* (1979); Haber and Oldenburg (2000)), and L-curve (Hansen (1998)) are commonly used. In practice, any of these methods will require us to approximately solve (VF) for a set of regularization parameters. Because of the binary constraints, the source w will be continuous only in trivial cases. In general, we expect piecewise constant solutions with finite edge measure. This guides our choice of the regularization function and motivates us to use the total variation (TV) semi-norm. With this choice, the solution of the relaxed problem will have bounded variation (Chan and Shen (2005); Rudin *et al.* (1992); Vogel (2002)). In our numerical experiments, we formally write the total variation regularizer as

$$\mathcal{R}(w) = \int_{\Omega} \|\nabla w(x)\|_2 \, dx, \tag{4.2}$$

which is well-defined for all $w : \Omega \to [0, 1]$ with bounded variation; for non-differentiable functions (e.g., binary-valued functions) $\mathcal{R}(w)$ can be computed using the co-area formula (Scherzer, O and Grasmair, M and Grossauer, H and Haltmeier, M and Lenzen, F, 2013, Thm 9.75). This regularizer is *isotropic*, which means that its value does not depend on the orientation the source w. In other words, its value is invariant to rotations of the coordinate system. This is an advantage over the also commonly used *anisotropic* version of TV (obtained by replacing the Euclidean norm with the ℓ_1-norm in (4.2)), which is sensitive to rotations of the domain. Problem (VF) is not a quadratic program, because the TV regularization term involves a square-root. It is easy to show that (4.2) is second-order-cone representable. However, we do not exploit this fact in our results.

4.2.2 Finite-Dimensional Approximations of Source Inversion Problem

In the following, we briefly outline a finite-element discretization of the PDE and boundary condition and comment on the structure these discretizations imply for the finite-dimensional MINLPs. Detailed mathematical analysis and implementation details are discussed in Sharma *et al.* (2020a). For ease of presentation, we assume a rectangular domain $\Omega = [0, 1]^d$.

The finite-element approach partitions the domain Ω into a mesh Ω_N containing N^d congruent elements (pixels or voxels; see an additional explanation in Sharma *et al.* (2020a)) in $d = 2$ and $d = 3$, respectively. We note however that a strength of the finite-element method is that it extends easily to a wide class of non-rectangular domains. We then approximate the concentration u in a finite-dimensional subspace consisting of globally continuous and piecewise bi/trilinear functions on Ω_N. Similarly, we approximate the

sources w in the space of all piecewise constant functions. This approach allows us to replace u and w by finite-dimensional vectors, $u \in \mathbb{R}^{(N+1)^d}$ and $w \in \{0,1\}^{N^d}$, respectively, where u_i corresponds to the value of u at node i of the finite-element mesh and w_i is the value of w inside element i. Even though the basis functions are nonlinear, the expansion of u and w is *linear* in the coefficients of the basis functions, and results in a set of linear equality constraints, leading to the finite-dimensional PDE constraint

$$S\,u = M\,w, \tag{4.3}$$

where S is the stiffness matrix and M is the mass matrix, whose entries are obtained by integrating the weak form of the PDE constraint in (VF); see Sharma *et al.* (2020a) for details. Because of the compact support of the Ansatz functions for u and w, both matrices are sparse. The discrete objective function is obtained after discretizing the receiver functions and gradient operators on the finite-element mesh. To discretize the inner products in (4.1), we use a midpoint rule. The resulting finite-dimensional convex MINLP is

$$\left.\begin{array}{ll} \underset{u,\,w}{\text{minimize}} & \frac{1}{2\sigma}\,\|Pu-b\|^2 + \alpha R(w) \\ \text{subject to} & S\,u = M\,w, \\ & w \in \{0,1\}^{N^d}, \end{array}\right\} \tag{CFP}$$

which is a finite-dimensional MINLP with a convex objective and linear constraints. Here, the matrix P is the interpolation of the states to the point-measurement locations. In particular, the ith row of the matrix $P \in \mathbb{R}^{m\times(N+1)^d}$ contains the discretization of the receiver function $p(r_i)$ on the mesh. In our experiments below, we assume point measurements at the locations $r_1, \ldots, r_m$ and use a bi/trilinear spline interpolation to obtain the PDE solutions at those points. Hence, each row of P contains only four/eight nonzero elements that correspond to the interpolation weights. More information on the discretized regularizer, $R(w)$, can be found in Sharma *et al.* (2020a).

We note that the MINLP (CFP) contains special structure that is not typically present in standard MINLPs, and we exploit this structure in the solution of the problem. Because the stiffness matrix, S, is nonsingular by design, we can solve (4.3) uniquely for u, given any choice of the discretized controls, w. Formally, we obtain $u = S^{-1}Mw$ and eliminate u, resulting in a pure integer nonlinear program over the controls, w, only. This corresponds to a reduced-space approach. The feasibility of this approach hinges on an efficient way to solve the PDE (operate with S^{-1}). For some problems, one may be able to obtain a factorization of the stiffness matrix and reuse it to evaluate the reduced-space objective and gradients. For large-scale problems the reduced-space approach may still be

feasible if an effective iterative method is available. This approach leads to the discretized MIPDECO

$$\underset{w}{\text{minimize}} \quad J(w) = \frac{1}{2\sigma}\left\|PS^{-1}Mw - b\right\|^2 + \alpha R(w) \tag{4.4a}$$

$$\text{subject to} \quad w \in \{0,1\}^{N^d}. \tag{4.4b}$$

We note that the regularization term in problem (4.4) can be reformulated so that (4.4) becomes a mixed-integer second-order cone problem. The relaxation of (4.4) is

$$\underset{w}{\text{minimize}} \quad J(w) = \frac{1}{2\sigma}\left\|PS^{-1}Mw - b\right\|^2 + \alpha R(w) \tag{4.5a}$$

$$\text{subject to} \quad w \in [0,1]^{N^d}, \tag{4.5b}$$

which we solve using a projected Gauss-Newton algorithm.

Remark on Structure of the MINLP (4.4)

1. The reduced-space approach presented here can be generalized to nonlinear PDEs. In this case, however, the solution operator of the PDE (S^{-1} in the linear case) is no longer a linear operator. This observation implies that we can no longer reuse the factors of S and instead must "reinvert" the solution for every iterate in a quasi-Newton process; see Biros and Ghattas (2005a), Biros and Ghattas (2005b), and Akçelik *et al.* (2006) on how the reduced-space methods are applied to nonlinear PDEs. The iterative process consists of a linearization of the optimality conditions, a gradient calculation with adjoint based sensitivities, which consists of a linear system solve with S^T and a linearized objective function as a right hand side.

2. The reduced-space objective function in (4.4) will typically have a dense Hessian matrix, even if S is sparse, and this can cause computational difficulties for MINLP solvers as we increase the mesh size.

3. In general, it is difficult to obtain closed-form expressions for the coefficients of the matrices S and M, making it cumbersome to state these equations in a self-contained model.

The last point motivates us to present an alternative finite-difference discretization that provides closed-form expressions for the coefficients of the discretized PDE to facilitate the reproducibility of our experiments; see Appendix E.1. The finite-difference discretization again results in a MINLP with linear constraints and a convex objective function. As before, we can eliminate the state variables using the discretized PDE and boundary conditions.

In the remainder of this chapter, we present our integrated rounding and trust-region heuristic and our numerical results. We note that the rounding and trust-region schemes are agnostic to the discretization scheme.

4.3 An Integrated Rounding and Trust-Region Heuristic

We show in our numerical results that standard MINLP methods cannot solve the discretized MIPDECOs from the preceding section within a reasonable amount of time, even for coarse discretization levels, because the branch-and-bound tree becomes too large and the subproblems at every node take too much time to solve. Hence, one must consider heuristic techniques. We present a new 'two-phase heuristic' for MIPDECO. In the first phase, we deploy a problem-specific rounding scheme whose solution is passed as an initial guess to the next phase. The second phase is an 'improvement heuristic' that is motivated by trust-region methods for nonlinear optimization; see, for example, Conn *et al.* (2000), as well as local-branching heuristics for MINLP (Fischetti and Lodi (2002); Nannicini *et al.* (2008)). We start our approach by first solving the continuous relaxation, which is then rounded using different heuristics. In Section 4.5.3, we numerically illustrate that starting from various rounded initial points, we arrive at different final solutions due to the different trust-region subproblems obtained at these initial solutions. Other heuristics that have been proposed for MINLPs are large neighborhood search (Danna *et al.* (2005a)) and feasibility pump (Fischetti *et al.* (2005); Bonami *et al.* (2009a)). However, we do not believe that the latter is practical for MIPDECOs because it would require factorizations and rank-one updates of the basis matrices involving the discretized PDE, which may be prohibitive or even impossible for small mesh sizes.

Our heuristic is agnostic to the discretization of the PDE or to the solution of the continuous relaxation. Hence, we assume in the remainder that the control variables, w_i, either represent the values of the control in element i from the finite-element discretization of Section 4.2.2 or represent a lexicographical ordering of the cell-centered controls, W_{kl}, in the finite-difference discretization (see Appendix E.1).

4.3.1 Rounding Schemes for MIPDECO

The improvement heuristic used in our proposed method requires a starting solution. To obtain such an integer feasible solution we present several rounding based schemes in this section. Our results show that the naïve/standard rounding (using a cut-off of 0.5) could fail to identify some sources. On the other hand, the existing NLP based rounding heuristics from the literature (Sigmund and Maute (2013b)) does not produce competitive

solutions in the sense that our trust-region method can improve these solutions objective value by around 24%. Hence, we propose two new rounding schemes to obtain an integer feasible solution, starting from the optimal solution of the continuous relaxation, which can be used as a starting solution for our heuristics. The first scheme takes the objective function into account while rounding, and the second aims to preserve the mass of the sources; that is, it tries to keep $\int_\Omega w\,dx$ invariant. For both proposed heuristics, we let $\widetilde{w} \in [0,1]^{N_d}$ be the solution of the relaxation (4.5) or (FDM).

First we briefly discuss existing NLP based heuristics followed by the proposed rounding schemes: objective-gap-reduction and mass-preserving.

Penalization Based NLP Heuristics. We implemented a penalty based rounding scheme from the literature (Sigmund and Maute (2013b)) in which we relax the integrality restrictions and instead solve a sequence of penalized NLPs for an increasing value of penalty parameter to drive the integrality gap to zero. In particular, we add a penalty term to the objective of (4.5) and (FDM), respectively, resulting in the following (nonconvex) penalized formulation of (4.5) (the approach for (FDM) is similar)

$$\left.\begin{array}{ll} \underset{w}{\text{minimize}} & \frac{1}{2\sigma}\left\|PS^{-1}Mw - b\right\|^2 + \alpha R(w) + \beta \sum_{i=1}^{N^d} \left(w_i\,(1-w_i)\right)^q \\ \text{subject to} & w \in [0,1]^{N^d}, \end{array}\right\} \tag{PBNLP}$$

where q is a positive integer and β is a penalty parameter that we increase until the integrality gap is sufficiently small. In our implementation, we use $q = 1$. We solve the continuous relaxation with $\beta = 0$, set $\beta = 10^{-6}$, and increase β by a factor 2 until the integrality gap, $\max_i \{\min\{w_i, 1-w_i\}\}$, is sufficiently small ($\leq \epsilon := 10^{-4}$), and then round the final w to its nearest integer. We summarize this approach in Algorithm 8.

Algorithm 8: Penalization Based NLP Heuristic for FEM Discretization.

1 Let $w^{(0)}$ be a solution of (PBNLP) with $\beta = 0$;
2 Choose integrality gap $\epsilon > 0$, iteration limit, $K_{\max}$, set $k := 0$, and $\beta_0 := \beta_{\min} > 0$;
3 **while** $k < K_{\max}$ *and* $\max_i \left\{\min\{w_i^{(k)}, 1 - w_i^{(k)}\}\right\} > \epsilon$ **do**
4 Let $w^{(k+1)}$ solve the penalized relaxation (PBNLP) with $\beta = \beta_k$;
5 Set $\beta_{k+1} := 2 \cdot \beta_k$ and $k := k + 1$;
6 **return** *Rounded* $w^{(k)} \in \{0,1\}$;

Objective-Gap-Reduction Rounding. Given $\widetilde{w} \in [0,1]^{N_d}$, the first scheme selects a cut-off value t such that the resulting rounded solution defined as

$$\overline{w}_i(t) = \begin{cases} 1, & \text{if } \widetilde{w}_i \geq t, \\ 0, & \text{otherwise}, \end{cases} \tag{4.6}$$

is as close as possible to the relaxation solution in terms of its objective value. Mathematically, the desired cut-off value, t, is the minimizer of the following optimization problem

$$\underset{0\leq t\leq 1}{\text{minimize}}\, J(w(t)) - J(\widetilde{w}) \quad \Leftrightarrow \quad \underset{0\leq t\leq 1}{\text{minimize}}\, J(w(t)). \tag{4.7}$$

Consequently, we call this scheme *objective gap-reduction rounding*. The optimization problem in (4.7) can be written as a convex MINLP and the upper bound on t can be tightened to $\max_{i=1,\ldots,N_d} \widetilde{w}_i$. Because this problem is hard to solve, we propose a simple iterative algorithm to obtain an acceptable cut-off value for the rounding (see Algorithm 9), because we are interested only in the approximate solution of (4.7). The process starts by iteratively increasing t by a constant step $T \in (0,1)$ from a small initial value until t exceeds $t_{\max} = \max_{i=1,\ldots,N_d} \widetilde{w}_i$ and outputs the best cut-off value t^*. When $t^* = 0.5$, this scheme is the same as naïve rounding.

Algorithm 9: Objective-Gap-Reduction Rounding.

```
Let T ∈ (0, 1);
Set k := 0, t_k := min_{i=1,...,N_d} w̃_i, t* := 0, J* := ∞, and t_max := max_{i=1,...,N_d} w̃_i;
while t_k ≤ t_max do
    Form w(t_k);
    Solve the discretized PDE (4.4a) with w = w(t_k) and evaluate J(w(t_k));
    if J(w(t_k)) < J* then
        Set J* := J(w(c_k)) and t* := t_k;
    Set k := k + 1 and t_k := t_{k−1} + T;
Output: The best cut-off value t* and the rounded solution, w̄ = w(t*);
```

Mass-Preserving Rounding. The scheme rounds the relaxation solution while preserving the mass of the sources in the relaxation solution as much as possible. The mass of the sources in the relaxation solution is given by $\widetilde{S} = \sum_{i=1}^{N_d} \widetilde{w}_i$. Let $\bar{S} = [\widetilde{S}]$ be the nearest integer to $\widetilde{S}$. To compute the rounded solution $\overline{w}$, we first arrange the components of w in decreasing order of $\widetilde{w}_i$ values as

$$1 \geq \widetilde{w}_{i_1} \geq \widetilde{w}_{i_2} \geq \ldots \geq \widetilde{w}_{i_{N_d}} \geq 0.$$

Next, the largest $\bar{S}$ entries are set to 1, and the remaining entries are set to zero. The resulting rounded solution is

$$\overline{w}_{i_k} = \begin{cases} 1, \ k = 1, \ldots, \bar{S}, \\ 0, \ k = \bar{S} + 1, \ldots, N_d. \end{cases} \tag{4.8}$$

It follows that the difference in the mass of the sources between the rounded and the relaxed solutions is less than 1. Unlike the first rounding scheme, this rounding scheme does not require the solution of any additional PDEs once the relaxed problem (4.5) has been solved.

4.3.2 Trust-Region Based Improvement Heuristic

Our trust-region based heuristic for solving (4.4) starts from a binary vector, $w^{(0)} \in \{0,1\}^{N^d}$, and iterates on the binary variables, w. Here $w^{(0)} = \overline{w}$, an integer feasible solution from a rounding scheme in the first phase. At iteration k, we assume that we have solved the discretized PDE in (4.4) with fixed binary vector $w^{(k)} \in \{0,1\}^{N^d}$ and have evaluated the objective function and its adjoint; see, for example, Biegler *et al.* (2003) and Akçelik *et al.* (2006), with respect to the (relaxation of the) binary variables,

$$J^{(k)} := J(u^{(k)}, w^{(k)}) \quad \text{and} \quad J'^{(k)} := \nabla_w J(u^{(k)}, w^{(k)}).$$

We then define the trust-region subproblem that aims to find an improved point, $\widehat{w}$:

$$\left.\begin{array}{ll} \underset{w}{\text{minimize}} & J^{(k)} + J'^{(k)^T}(w - w^{(k)}) \\ \text{subject to} & \left\|w - w^{(k)}\right\|_1 \leq \Delta_k, w \in \{0,1\}^{N^d}, \end{array}\right\} \tag{TRP}$$

where $\Delta_k > 0$ is the trust-region radius and $\Delta_k \in \mathbb{Z}$ is the maximum number of components of w that can flip from 0 to 1 or 1 to 0 during an iteration. It is well known that we can rewrite the ℓ_1 trust-region constraint of (TRP) equivalently as a knapsack constraint

$$\begin{aligned} & \sum_{i:w_i^{(k)}=0} w_i + \sum_{i:w_i^{(k)}=1} (1 - w_i) \leq \Delta_k \\ \Leftrightarrow \quad & \sum_{i:w_i^{(k)}=0} w_i - \sum_{i:w_i^{(k)}=1} w_i \leq \Delta_k - \left|\left\{i : w_i^{(k)} = 1\right\}\right|, \end{aligned}$$

because $w^{(k)} \in \{0,1\}^{N^d}$. This reformulation is the motivation for using the ℓ_1, rather than the ℓ_2 trust-region, because it results in an easier to solve trust-region subproblem.

We also introduce an alternative trust-region subproblem that, in addition to the ℓ_1 trust-region constraint of (TRP), restricts the changes in w to components that are close to current source locations. In particular, we let $\theta > 0$ be a bound on the topological

distance from the current solution, and we define the center of $C(w_i)$ as the coordinates of the center of the element or cell, i, corresponding to w_i. We define the topological θ-neighborhood of the current iterate $w^{(k)}$ as

$$\mathcal{N}_\theta(w^{(k)}) := \left\{ i \ : \ \exists j \text{ with } \left\| C(w_i) - C(w_j^{(k)}) \right\|_2 \leq \theta \text{ and } w_j^{(k)} = 1 \right\}, \tag{4.9}$$

which defines an index set of the finite elements whose centroid is within a distance θ to the centroids of the current estimate of the source, $w_j^{(k)} = 1$. Our alternative trust-region subproblem is then defined as the following problem in which the binary variables that lie outside the neighborhood, $\mathcal{N}_\theta(w^{(k)})$, are fixed at their current values 0

$$\left.\begin{aligned} \underset{w}{\text{minimize}} \quad & J^{(k)} + J'^{(k)^T}(w - w^{(k)}) \\ \text{subject to} \quad & \left\| w - w^{(k)} \right\|_1 \leq \Delta_k, \\ & w_i = 0, \forall i \notin \mathcal{N}_\theta(w^{(k)}), \ w \in \{0,1\}^{N^d}. \end{aligned}\right\} \tag{NTRP}$$

Unlike (TRP), this problem takes the topology of the current iterate into account when defining the trust-region subproblem, because values of w_i that are far from the current sources are fixed at zero. Given either of these trust-region subproblems, we now state our improvement heuristic in Algorithm 10.

In Algorithm 10, we increase the trust-region radius if we observe good agreement (as measured by ρ_k) between the objective function in (4.4) and its linear approximation in (TRP). If the two do not agree, then we reduce the trust-region radius in the hope of getting better agreement in a smaller region. We use the ℓ_1-norm trust-region, because it corresponds to the Hamming distance for binary vectors, and the trust-region radius can be interpreted as limiting the number of binary variables that can change from the current iterate $w^{(k)}$. We also ensure that the trust-region radius is always an integer, and the algorithm stops, once the radius becomes zero. The trust-region radius becomes zero after a finite number of iterations, because the objective is bounded below by zero, there exists a finite number of integer assignments, w, and the trust-region reduction uses the floor operator.

4.3.3 Solving the Trust-Region Subproblems

The trust-region subproblems (TRP) and (NTRP) are linear binary optimization problems with a single constraint. One can easily see that as long as the trust-region radius is non-negative (i.e., as long as $\Delta_k \geq 0$), the problem is feasible. Here we show that this binary optimization problem can be solved to optimality efficiently by observing that it can be reduced to a special knapsack problem for which efficient solution methods exist.

Algorithm 10: Trust-Region Based Improvement Heuristic.

Set the initial trust-region radius, $\Delta_0 \in \mathbb{Z}_+$, and let $w^{(0)} \in \{0,1\}^{N^d}$;
Choose constant $0 < \gamma < 1$, and set $k := 0$;
while $\Delta_k \geq 1$ **do**
    Solve subproblem (TRP) or (NTRP) for $\widehat{w} :=$ argmin(TRP) or (NTRP);
    Solve the PDE for $\widehat{u} := u(\widehat{w})$ and evaluate the objective $J(\widehat{w})$;
    Compute the ratio of actual over predicted reduction:

$$\rho_k := \frac{J(w^{(k)}) - J(\widehat{w})}{-J'^{(k)^T}(\widehat{w} - w^{(k)})}$$

    if $\rho_k > \gamma$ **then**
        Accept the new point: $w^{(k+1)} = \widehat{w}$, solve the adjoint PDE to get $J'^{(k+1)}$;
        if $\|w^{(k+1)} - w^{(k)}\|_1 = \Delta_k$ **then**
            Increase the trust-region radius $\Delta_{k+1} := 2\Delta_k$;
    else if $\gamma \geq \rho_k > 0$ **then**
        Accept the new point $w^{(k+1)} := \widehat{w}$, solve the adjoint PDE to get $J'^{(k+1)}$;
        Leave the trust-region unchanged $\Delta_{k+1} = \Delta_k$;
    else
        Reject the new point, set $w^{(k+1)} := w^{(k)}$, and set $J'^{(k+1)} := J'^{(k)}$;
        Reduce the trust region $\Delta_{k+1} = \lfloor \Delta_k/2 \rfloor$;
    Set $k := k + 1$;

We start by writing the trust-region subproblem as a generic binary linear program of the form

$$\left.\begin{aligned} \underset{w}{\text{minimize}} \quad & g^T w \\ \text{subject to} \quad & a^T w \leq b, \\ & w \in \{0,1\}^p, \end{aligned}\right\} \tag{BLP}$$

where $g \in \mathbb{R}^p$ is the gradient, $a \in \{-1,1\}^p$, and $b \geq 0$ is a positive integer.

To extend the knapsack solution approach to our problem, we distinguish the following cases:

1. $a_i = 1$ and $g_i > 0$ implies that $w_i = 0$ at a solution of (BLP) (because increasing w_i deteriorates both the objective and the constraint satisfaction).

2. $a_i = -1$ and $g_i < 0$ implies that $w_i = 1$ at a solution of (BLP) (because decreasing w_i deteriorates both the objective and the constraint satisfaction).

3. $a_i = -1$ and $g_i \geq 0$: We replace the variable w_i by its "inverse", $\check{w}_i := 1 - w_i$. This change of variable reverses the signs of g_i and a_i, which can be handled by the knapsack approach. We also need to update the right-hand side of the constraint as $\check{b} := b - a_i = b + 1 > b$.

We can now remove the variables w_i that correspond to the first two cases and consider a reduced knapsack problem in standard form with $\check{m} \leq p$ binary variables in the transformed data $\check{g}$, $\check{a}$, $\check{b}$

$$\left.\begin{array}{ll} \underset{\check{w}}{\text{minimize}} & \check{g}^T \check{w} \\ \text{subject to} & \check{a}^T \check{w} \leq \check{b}, \\ & \check{w} \in \{0,1\}^{\check{m}}, \end{array}\right\} \qquad (B\check{L}P)$$

where $\check{a} = (1, \ldots, 1)^T$, and $\check{b} \geq 0$. We now sort the indices in increasing order of coefficients

$$\check{g}_{i_1} \leq \check{g}_{i_2} \leq \check{g}_{i_k} < 0 \leq \check{g}_{i_{k+1}} \leq \cdots \leq \check{g}_{i_{\check{m}}},$$

where ties are broken arbitrarily, and we set $i_k = 0$ if $\check{g}_i \geq 0$ for all indices i. The solution of the reduced knapsack problem ($B\check{L}P$) is obtained by setting $\check{w}_{i_l} = 1$ for all $l = 1, \ldots, \min(\check{b}, i_k)$ and $\check{w}_{i_l} = 0$ for all $l > \min(\check{b}, i_k)$; see, for example, Balas (1975), Horowitz and Sahni (1974), Martello and Toth (1990), Pisinger and Toth (1998), and Martello *et al.* (1999).

4.4 Implementation and Experimental Setup

In this section, we describe our implementation and the generation of the test instances, and we briefly comment on the calibration of the regularization parameter. We also review a popular NLP based rounding heuristic that we use in our comparisons.

4.4.1 Implementation Details

We implemented prototype versions and test instances of the proposed algorithms in Julia (Bezanson *et al.* (2012)), which will enable future algorithmic developments thanks to Julia's rapid prototyping capabilities, and AMPL, which facilitates testing different MINLP solvers and relaxation ideas for our problem. To enable the reproducibility of our results, we provide a Julia module containing the source inversion problem and an implementation of the trust-region method freely at `www.github.com/JuliaInv/ConvDiffMIPDECO`.

The module provides methods to compute the forward problem and matrix-vector products with the adjoint. It also contains several interactive examples that can be modified and extended. The module depends on and extends jInv (Ruthotto *et al.* (2017)), a toolbox for PDE-parameter estimation problem. Our module uses the existing methods in jInv for numerical optimization, PDE solvers, regularization, and visualization in our experiments. All models are solved on a system with two 64-bit Intel(R) Xeon(R) E5-2670 v2, 2.50 GHz CPUs having 10 cores each and sharing 128 GB of RAM. Using the module JuMP (Dunning *et al.* (2017)), our module can also be used to interface with a variety of integer-programming solvers.

In addition we created AMPL code that discretizes the PDE constraint and formulates the MIPDECO, using the finite-difference discretization described below in Appendix E.1. The models and run scripts are freely available at `https://github.com/JuliaInv/ConvDiffMIPDECO/tree/master/examples/ampl`.

We provide the AMPL model as well as scripts that run the penalization based NLP heuristic described in Section 4.3.1 and scripts that allow the user to output images for further processing in MATLAB. We do not directly compare the Julia runs with the AMPL runs in terms of CPU time because this performance measure is strongly dependent on how well the solvers can exploit the structure of the PDE constraint.

4.4.2 Generation of Test Problems and Regularization Parameter

Two-dimensional instance. Using the domain $\Omega = [0, 2] \times [0, 1]$, we construct a 2D source model by evaluating MATLAB peaks function at the cell centers of a grid with 550×256 equally sized cells. Rounding the function with a threshold of 2 results in two sources, one of which we shift right along the x-axis. The true model can be seen in the upper-left subplot of Figure 4.1.

To generate the measurements, we solve the PDE using the finite-element method (FEM) discretization on this mesh with a velocity of $v = (1, 0)^\top$ and a diffusion of $c = 0.01$ and then evaluate the PDE solution at 200 random receiver locations sampled from a uniform distribution on Ω. We visualize the PDE solution and the receiver locations (marked by red dots) in the upper-right subplot of Figure 4.1.

Three-dimensional instance. The data for the 3D instance is generated along the same lines. Here, we choose the domain $\Omega = [0, 2]\times[0, 1]\times[0, 1]$, a mesh size of $128\times64\times64$, and construct a 3D source model by adding three scaled and shifted norm balls. We visualize the true model in the lower-left subplot of Figure 4.1.

To generate the measurements, we solve the PDE using the FEM on this mesh with a velocity of $v = (1, 0, 0)^\top$ and a viscosity of $\sigma = 0.01$, and we then evaluate the PDE solution at 200 randomly spaced boreholes whose first two components are sampled from a uniform distribution on $[0, 2] \times [0, 1]$. In the third dimension, we place one receiver at each mesh cell, which yields overall 12,800 measurements. We visualize the PDE solution using an isocontour plot and the receiver locations (marked by red lines) in the lower-right subplot of Figure 4.1. In Table 4.1 we show the problem sizes of the instances that we solve in our experiments.

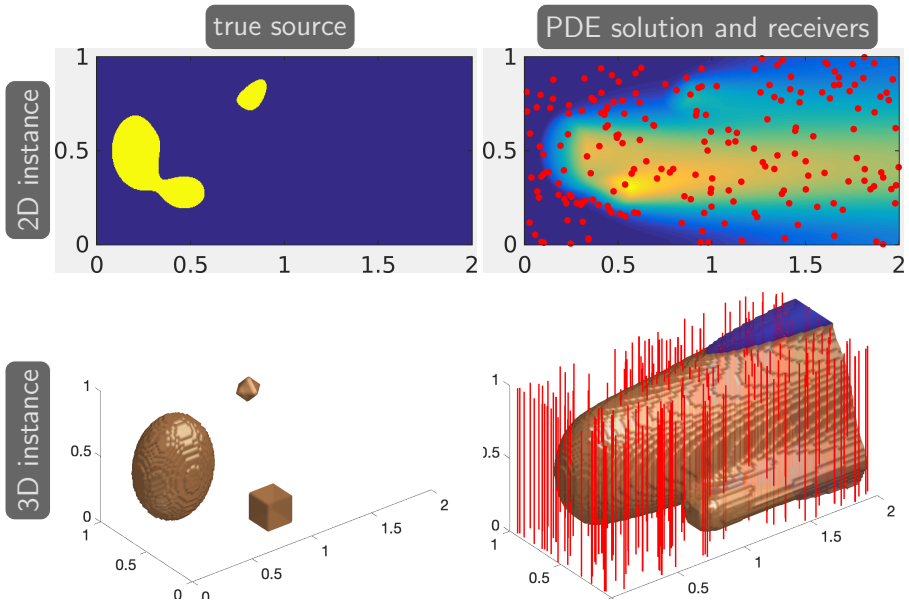


Figure 4.1: Visualization of the ground-truth sources (left column) and the generated test data (right column) for the 2D (top row) and 3D (bottom row) instance. The data is obtained by sampling the PDE solution associated with the source model at the randomly chosen receiver locations (indicated by red dots and lines, respectively).

Because our inverse problem is underdetermined (we have fewer measurements than unknown optimization variables, w), we must add a regularization term, $\mathcal{R}(w)$, in (4.2). This regularization term requires us to choose the regularization parameter, α, in (CFP). To find an effective regularization parameter, we use the continuous relaxation (4.5) and follow the L-curve approach that we describe in more detail in Appendix E.2. Using this process, we select the regularization parameters $\alpha = 8.531 \cdot 10^{-3}$ for the 2D instance and $\alpha = 5.298 \cdot 10^{-3}$ for the 3D instance, respectively.

Method	Mesh size	# States	# Binary control	# Constraints
FDM	16×8	180	128	180
	32×16	612	512	612
	64×32	2244	2048	2244
	128×64	8580	8192	8580
	256×128	33540	32768	33540
FEM	256×128	33153	32768	33153
	$96 \times 48 \times 48$	232897	221184	232897

Table 4.1: Problem size for 2D and 3D MIPDECO instances: For each method and mesh size, we show the number of discrete state, control variables, and constraints. 2D and 3D instances have 200 and 12, 800 number of measurements, respectively

4.5 Numerical Results and Discussion

In this section, we illustrate the performance of the different approaches to the discrete source inversion problem using numerical experiments in two and three dimensions with known ground truth. We show empirically that state-of-the-art MINLP solvers cannot solve even small-scale two-dimensional instances of this problem. Next, we consider naïve rounding (also referred to as standard rounding) and two proposed rounding heuristics applied to the relaxed problem, and we show that they also fail to solve the problem. The latter rounding schemes yield better solution than does naïve rounding. We then show that our trust-region heuristic improves on rounding heuristics to produce good-quality solutions in a reasonable amount of time in both two- and three-dimensional cases.

4.5.1 Performance of MINLP Solvers on 2D Instances

In this section, we explore the effectiveness of state-of-the-art MINLP solvers for tackling the discretized MIPDECO (4.4) for the 2D instance. We use six state-of-the-art MINLP solvers: SCIP (Achterberg (2009)), BONMIN (Bonami and Lee (2007)) using its hybrid (Bonmin-Hyb), branch-and-bound (Bonmin-BnB), and outer-approximation (Bonmin-OA) algorithms, and MINOTAUR (Mahajan *et al.* (2020)) with both its branch-and-bound (Minotaur-Bnb) and LP/NLP based branch-and-bound (Minotaur-QG) algorithms. We use the self-contained finite-difference discretized MIPDECO model (presented in (FDM)) for this comparison, because it allows us to easily explore the effect of increasing the discretization and enables others to easily reproduce our results. Otherwise, we use the same problem setup as described in Section 4.4.

We solve a number of instances of the 2D test problem for mesh sizes between 16×8 to 256×128. Table 4.1 reports the sizes of these instances. In this experiment, we use the regularization parameter, $\alpha = 8.531 \cdot 10^{-3}$, obtained following the L-curve approach. We limit the CPU time for the MINLP solvers to 10 hours. We report the number of nodes processed, runtime, lower and upper bounds, and percentage gap which is a measure of the optimality gap, and is defined as $100 \times \frac{UB-LB}{|UB|}$, where UB and LB are the upper and lower bounds, respectively, from these solvers. If any of these bounds is unknown, we set the percentage gap to infinity. We note that SCIP reports that the optimality gap is infinite if the lower and upper bounds have opposite signs.

We use the intersection-over-union (IoU) score (also known as Jaccard index) to quantify the overlap between the true source and the reconstruction source; see Csurka *et al.* (2013). Let M_{true} and M_{recon} denote the sets for which the true source, w, and the reconstructed source, w_{recon}, are indicator functions, respectively. The IoU score is then defined as the volume of the intersection divided by the volume of the union of these sets:

$$\text{IoU} = \frac{|M_{\text{true}} \cap M_{\text{recon}}|}{|M_{\text{true}} \cup M_{\text{recon}}|} \in [0, 1].$$

Higher values of the IoU score indicate a better overlap. Since the inversion is performed on coarser meshes, we use a next-neighbor interpolation to refine the reconstructed sources.

In Table 4.2, we summarize the performance of the MINLP solvers. We observe that only the two branch-and-bound solvers, Bonmin-Bnb and Minotaur-Bnb, are able to solve the smallest 16×8 instance; Bonmin-BnB also solved 32×16 but took around 454 minutes. All other runs time out after 10 CPU-hours, and in many cases the solvers fail to even produce a feasible source, W, or at least one of the bounds (lower or upper), indicated by ∞ in the last column. Bonmin-OA finds only the trivial feasible point, $W = 0$, on all the instances, indicating that there are no sources. Hence, we excluded the Bonmin-OA results.

Figure 4.2 shows the best solution, W, obtained by the MINLP solvers. The results for the 16×8 case show that the upper bound by SCIP and Bonmin-Hyb are far from optimal; Minotaur-QG found the upper bound (which in this case is also optimal) but could not improve its lower bound and thus finished with a positive optimality gap. As we increase the size of the problem, the MINLP solvers tend to obtain poor reconstructions with speckled areas that would make a source identification difficult. The worst performance is at the finest discretization level, where only Bonmin-Hyb returns some speckled sources and all other solvers fail to identify the sources.

One reason for this poor performance is that all MINLP methods solve a large number of relaxations of the original problem, such as linear programs, nonlinear programs,

Table 4.2: Performance of MINLP solvers for instances of the 2D test problem with mesh sizes ranging from 16×8 to 256×128. The rows represent different solvers and are grouped by mesh sizes. The columns show (left to right) the mesh size, the name of the solver, the number of nodes processed, the run-time in seconds (where TIME-OUT indicates that we reached the time limit of 10 CPU-hours), the lower and upper bounds, and the percentage gap remaining.

	Solver	# nodes	Runtime [sec.]	Bound		Gap [%]
				lower	upper	
16×8	Scip	225691	TIME-OUT	-0.5609	0.1733	∞
	Bonmin-Bnb	266	10.71	0.0530	0.0530	0
	Bonmin-Hyb	2087613	TIME-OUT	0.0413	0.0612	32.40
	Minotaur-Bnb	1043	163.87	0.0530	0.0530	0
	Minotaur-QG	560436	TIME-OUT	0.0416	0.0530	21.51
32×16	Scip	32868	TIME-OUT	-1.6063	–	∞
	Bonmin-Bnb	242126	27222.15	0.0393	0.0393	0
	Bonmin-Hyb	1606956	TIME-OUT	0.0347	0.0634	45.24
	Bonmin-OA	2087613	TIME-OUT	0.0413	0.0612	32.40
	Minotaur-Bnb	58115	TIME-OUT	0.0364	0.0575	36.71
	Minotaur-QG	265309	TIME-OUT	0.0347	0.0470	26.08
64×32	Scip	183	TIME-OUT	-2.0189	1.7104	∞
	Bonmin-Bnb	13569	TIME-OUT	0.0338	0.0369	8.38
	Bonmin-Hyb	293128	TIME-OUT	0.0329	0.0859	61.64
	Bonmin-OA	2087613	TIME-OUT	0.0413	0.0612	32.40
	Minotaur-Bnb	956	TIME-OUT	0.0329	0.1570	79.03
	Minotaur-QG	322969	TIME-OUT	0.0329	0.0449	26.61
128×64	Scip	1	TIME-OUT	–	–	∞
	Bonmin-Bnb	1	TIME-OUT	0.0323	0.0481	32.79
	Bonmin-Hyb	17316	TIME-OUT	0.0293	0.1696	82.69
	Bonmin-OA	2087613	TIME-OUT	0.0413	0.0612	32.40
	Minotaur-Bnb	8	TIME-OUT	0.0322	–	∞
	Minotaur-QG	77295	TIME-OUT	0.0322	0.0696	53.74
256×128	Scip	811	TIME-OUT	–	–	∞
	Bonmin-Bnb	1	TIME-OUT	0.0355	–	∞
	Bonmin-Hyb	17316	TIME-OUT	0.0119	0.1725	93.1
	Bonmin-OA	2087613	TIME-OUT	0.0413	0.0612	32.40
	Minotaur-Bnb	1	TIME-OUT	–	–	∞
	Minotaur-QG	1	TIME-OUT	–	–	∞

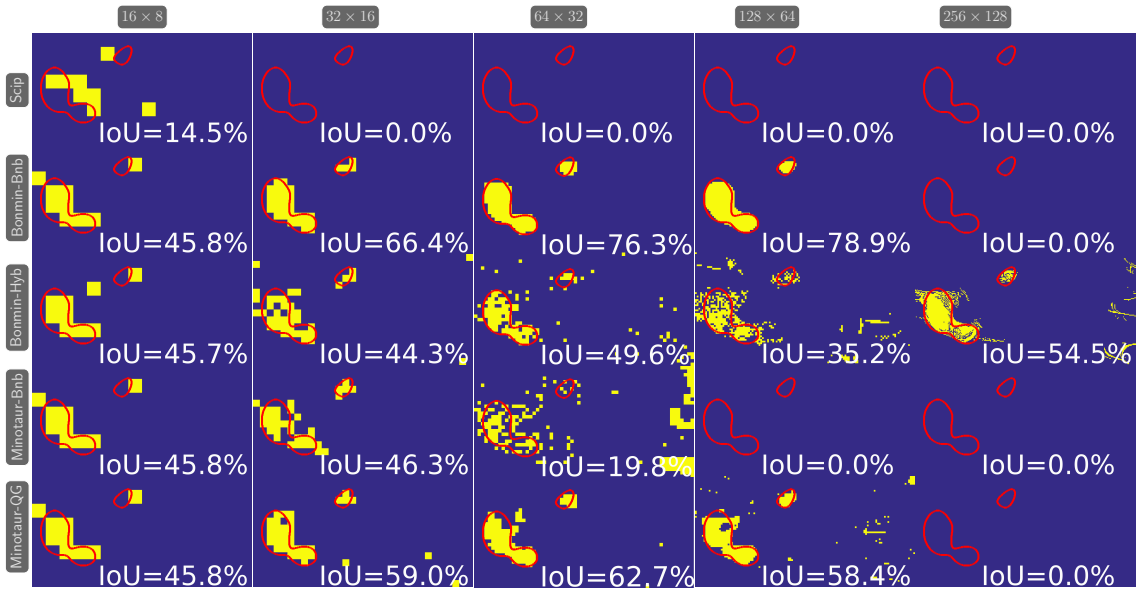


Figure 4.2: Solutions (W) from MINLP solvers SCIP, Bonmin-Bnb, Bonmin-Hyb, Minotaur-Bnb, and Minotaur-QG (row-wise) on mesh sizes of 16×8, 32×16, 64×32, 128×64, 256×128 (column-wise). We indicate the IoU value in the lower-right corner of each image.

and mixed-integer linear programs, depending on the specific method. Moreover, the problem size increases as we refine the computational mesh, making these problems larger and computationally harder to solve. None of the off-the-shelf solvers exploit the special structure that is inherent in the discretized PDEs and, for example, do not take advantage of the fact that the stiffness matrix needs to be factorized only once.

Another factor that prevents the MINLP solvers from solving our problem is the presolve techniques that SCIP and MINOTAUR employ, such as bound tightening, and the derivation of implications, before starting (and intermittently during) the tree-search (Achterberg (2009); Mahajan *et al.* (2020)). SCIP, for example, reformulates the original problem by decomposing the nonlinear objective function into a set of quadratic and nonlinear constraints whose number increases with the size of the instance. For mesh size 128×64, the SCIP preprocessing step took 425.95 seconds and resulted in 8,002 added quadratic constraints, making relaxations harder to solve, especially in view of the fact that our problem can be solved as a bound-constrained NLP by eliminating the PDE states and constraint.

We note that the heuristics used in SCIP and cutting planes used in Bonmin-OA to find better feasible solutions seem to be ineffective for our problems. For example, the best solutions reported by Bonmin-OA for all instances is $W = 0$, which indicates that there are no sources, which - even though feasible - is not a meaningful solution.

We also observe that the optimality gap is high for most solvers, because the lower and upper bounds are quite weak, leading to slow convergence. For mesh-size 256×128, Bonmin (in all algorithms) could not improve its starting lower bound even after the 10 CPU-hours, and MINOTAUR failed to report even a single lower bound because of a restoration failure in IPOPT that assumed local infeasibility. Initially, we allowed only two CPU-hours. Raising this limit to ten hours did not change the quality of the bounds, indicating that these problems are unlikely to be solved within a reasonable time with existing MINLP solvers.

4.5.2 Results for Rounding Approaches

Here, we compare the effectiveness of the different rounding schemes discussed in Section 4.3.1, on the finest mesh (256×128). The naïve, mass-preserving, and gap-reduction rounding schemes start from the continuous relaxation solution of (CFP) given by (4.5). In contrast, the NLP based rounding heuristic solves a sequence of NLPs, taking 11 and 7 iterations for the 2D and 3D case, respectively. In Table 4.3 we report the objective value, the solution time, and the number of PDEs of the solutions obtained from these rounding schemes. For the first two rounding schemes, the number of PDE solves is due to solving the relaxation (4.5). While the computational costs of the naïve and mass-preserving scheme are negligible, the gap reduction rounding requires repeated evaluation of the objective function and thus the PDE solves. Note that the factorizations of the discretized PDEs were computed during the solution of the relaxed problem and reused during the rounding. The additional costs are 16 and 13 PDE solves in the 2D and 3D cases, respectively. The majority of the time required by these three rounding schemes is from solving the relaxation. All the objective values reported henceforth are the objective value as in (4.5).

	2D			**3D**		
Rounding	Obj	Time (s)	# PDE solves	Obj	Time (s)	# PDE solves
Naïve	0.1098	24.29	462	0.0246	587.27	565
Mass-pres	0.0780	24.25	462	0.0262	587.24	565
Gap-red	0.1027	24.76	478	0.0246	600.36	578
NLP based	0.0530	794.10	2750	0.0204	12920.39	1518

Table 4.3: Comparison of objective value, solving time, and number of the PDE solves of different rounding schemes.

In Figures 4.6 and 4.7 the first column shows the solution w from these rounding schemes. The NLP based rounding resulted in a better solution than the rest but took around 33 (22) times more time for 2D (3D) case. We applied the different schemes at every iteration of the NLP based rounding heuristic. In the 2D case, we obtain better solutions with proposed rounding schemes (mass-preserving and gap-reduction) than with a simple rounding scheme, and in some iterations, our rounded solutions are better than any solution of the NLP based rounding heuristic. In the 3D case, the naïve and gap-reduction roundings resulted in the same solution. Figure 4.3 reports iteration statistics of the NLP based rounding heuristic. The top row corresponds to the 2D case and bottom row to the 3D case. In a row, the leftmost figure shows the number of PDE solves and solution time at each iteration; the middle figure reports the optimal objective value (Obj val) and objective value of the integer solution obtained by employing different rounding schemes (naïve, mass-preserving, and gap-reduction) to the solution of NLP based heuristic at each iteration; the rightmost figure shows the IoU values associated with different integer feasible solution reported in the middle figure. These results encourage us to believe that NLP based heuristic with higher integrality tolerance can also give a good-quality integer solution when used in conjunction with the proposed simple rounding schemes. We note that at iteration 10 of the 2D instance the objective value from the NLP based heuristic is more than the rounded solution given by the gap-reduction rounding. This means that the former is not a valid lower bound, because problem (PBNLP) is nonconvex, and we solve it only with a (local) projected Gauss-Newton method.

4.5.3 An Integrated Rounding Heuristic and Trust-Region Approach

We now apply our new trust-region approach to the intermediate solutions from the NLP based rounding approach of Section 4.5.1 for both the 2D and 3D instances. Note that the first iteration in the NLP based heuristic corresponds to the relaxation of (CFP). For each iteration we consider three rounding schemes (naïve, mass-preserving, and objective gap-reduction) and two versions of the trust-region approach (full region and neighborhood of 1 pixel), resulting in six combinations of our algorithm (at each iteration). We refer to $\mathcal{N}_\theta$ in (4.9) with $\theta = \sqrt{\left(\frac{2}{256}\right)^2 + \left(\frac{1}{128}\right)^2}$ and $\theta = \sqrt{\left(\frac{2}{96}\right)^2 + \left(\frac{1}{48}\right)^2 + +\left(\frac{1}{48}\right)^2}$ as neighborhood of 1 pixel in our 2D and 3D cases, respectively. For the 2D instance, Figure 4.4 shows the objective and IoU values from each of these combinations at each iteration; left and right columns are for the trust-region approach on full-space and neighborhood of one pixel, respectively. From our compu-

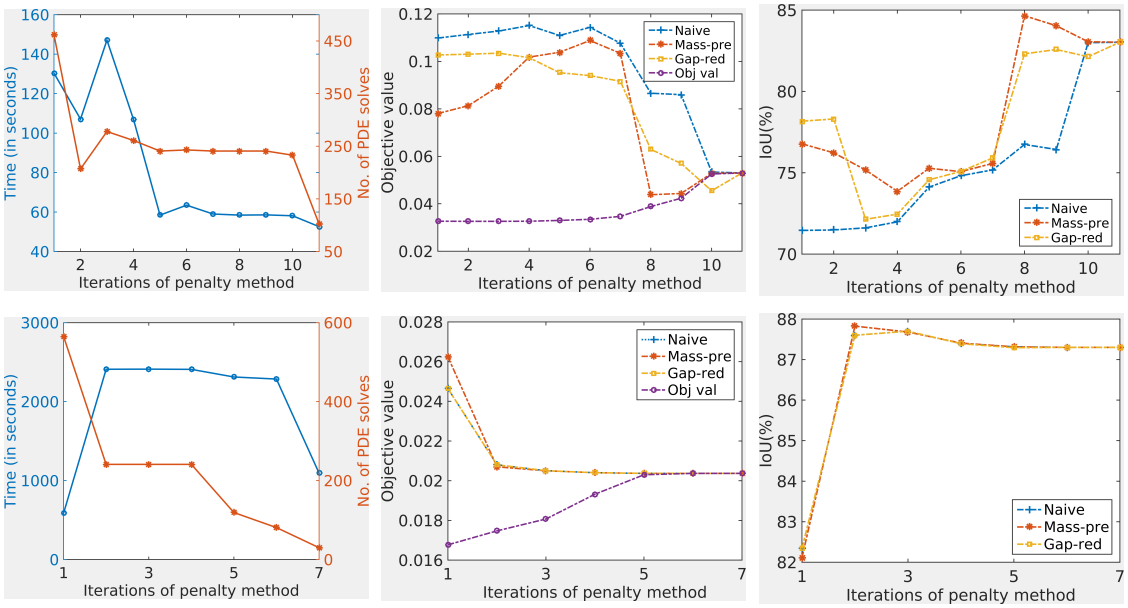


Figure 4.3: Detailed summary of solution time, # of PDE solves, objective value of the original problem, and objective and IoU values of integer feasible solutions obtained by the naïve, mass-preserving, and gap-reduction roundings of the solution at each iteration in the NLP based heuristic. Top row is for 2D instance and bottom row is for 3D instance.

tational results we see that for the 2D instance, for all the iterations (other than iteration 7) the mass-preserving has performed better than the other rounding schemes. Also, the mass-preserving and objective gap-reduction roundings give better solutions in terms of objective than does naïve rounding for the trust-region approach in all the iterations other than the last two. In the last two iterations the integrality gap in small and all three roundings result is the same initial and thus final solutions. Similar results for the 3D case are reported in Figure 4.5.

Figure 4.6 shows the source reconstructions for the first iterate of the penalty method applied to the 2D instance for each of the six combinations of our algorithm (first three rows), and the last iterate considering only mass-preserving rounding (since the integrality gap is small, all the three rounding schemes give same initial solution) in the last row. Each row depicts the initial guess, the reconstruction computed by the full-space trust-region method, and the reconstruction from the neighborhood trust-region method for a given rounding scheme. The superimposed red lines depict the shape of the true source. In each case, the overlap is improved by the trust-region method. Similar results for the 3D case are reported in Figure 4.7.

In the 2D case, we observe that for the first few iterations in the NLP based heuristics naïve rounding identified the larger of the two sources and completely failed to identify the second smaller source. However, the other two roundings, mass-preserving and gap-

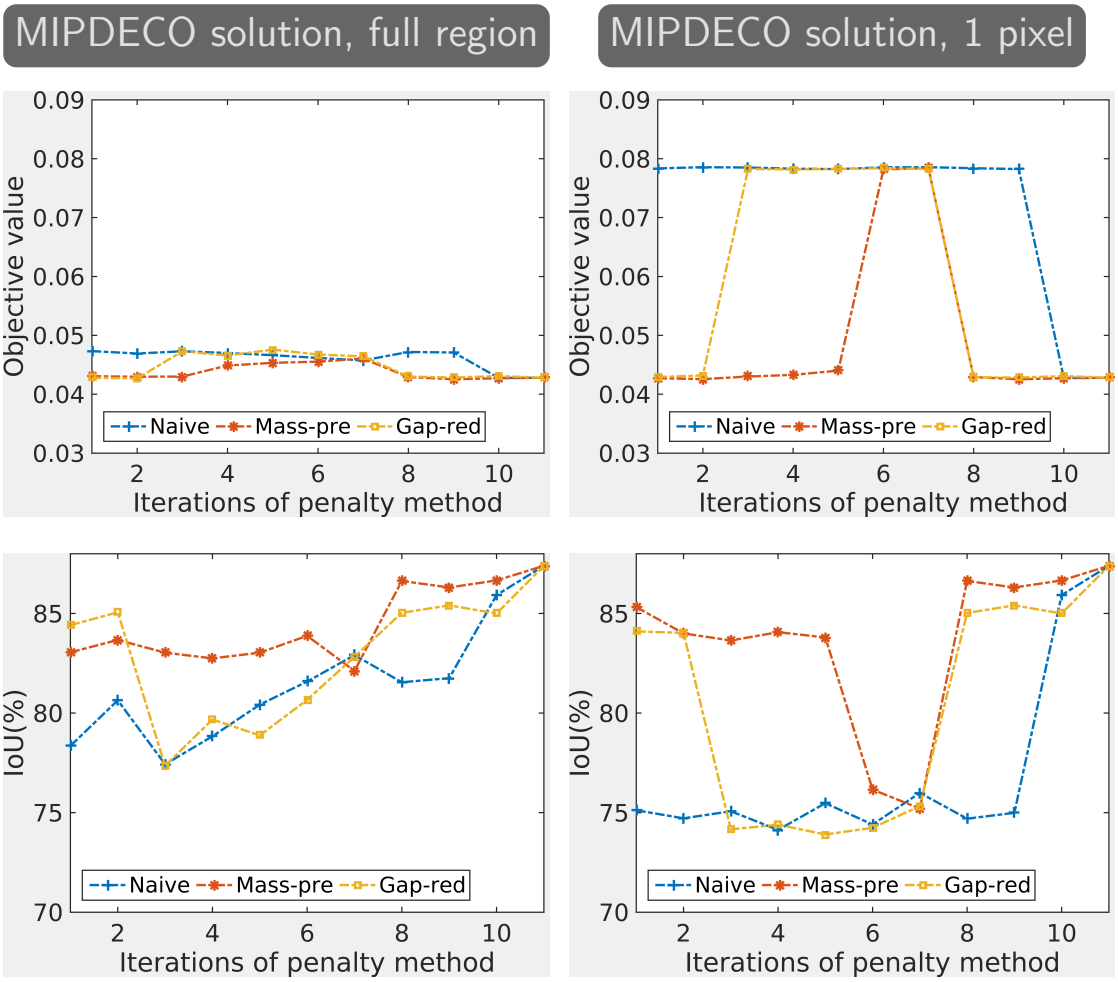


Figure 4.4: Objective and IoU values of trust-region approach on the solutions obtained by the naïve, mass-preserving, and gap-reduction roundings of the solution at each iteration in the penalization based NLP heuristic for 2D instance.

Rounding	2D		3D	
	Full region	1 pixel	Full region	1 pixel
Naïve	0.0473	0.0784	0.0209	0.0207
Mass-preserving	0.0431	0.0427	0.0208	0.0207
Gap-reduction	0.0428	0.0429	0.0209	0.0207
NLP heuristic	0.0428	0.0428	0.0204	0.0204

Table 4.4: Final objective value of rounding heuristics and trust-region approach.

reduction, identified both sources. When only one of the sources is recovered by using a rounding schemes the neighborhood trust-region approach cannot identify the second source, because it can change w_i only near the initial guess. On the other hand, the full-

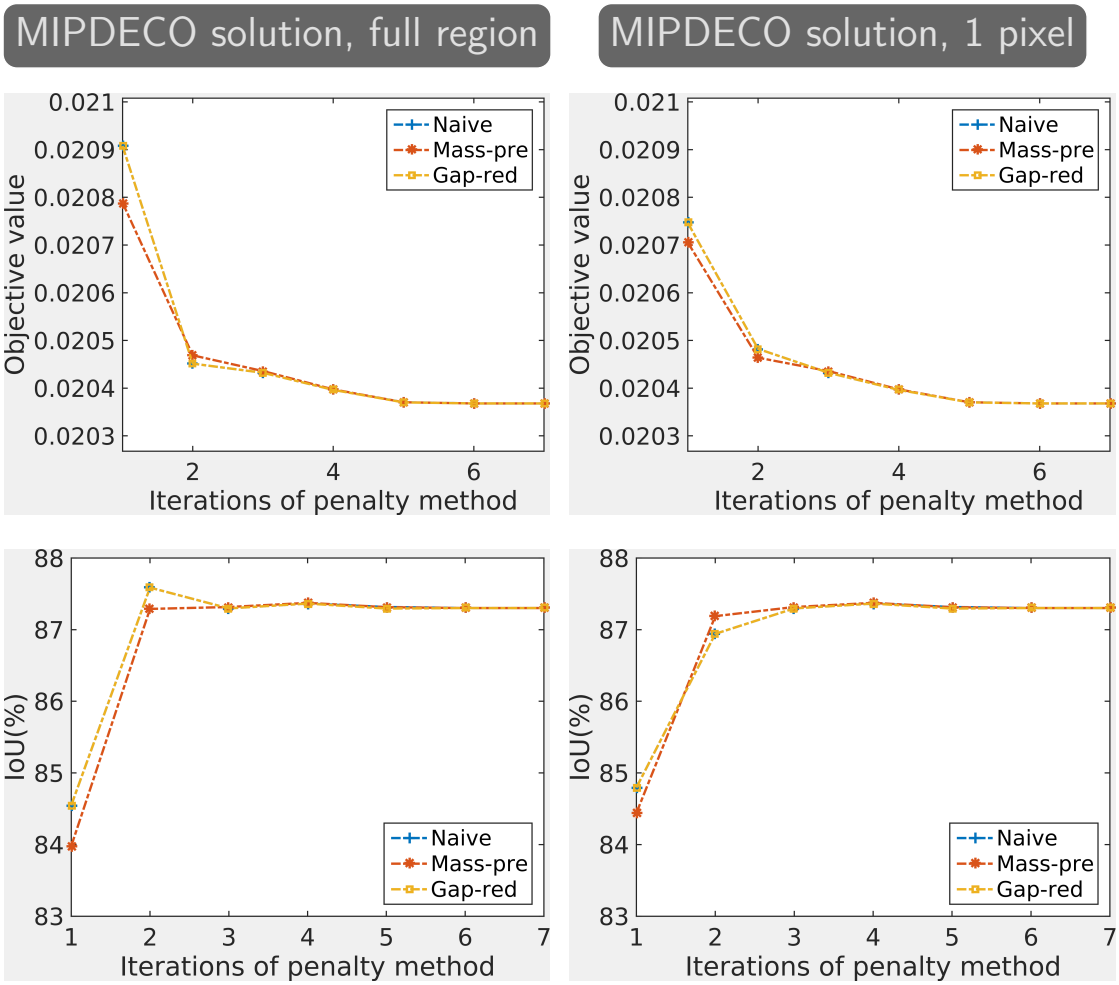


Figure 4.5: Objective and IoU values of trust-region approach on the solutions obtained by the naïve, mass-preserving, and gap-reduction roundings of the solution at each iteration in the penalization based NLP heuristic for 3D instance.

space trust-region algorithm discovers the second smaller source as well, independent of the starting guess.

In 2D (3D) instances, the added runtime of the trust-region approaches is around 7 seconds (between 58 and 129 seconds) when the initial guess is obtained from the first iteration of the NLP based heuristic and less than 3 (between 10 and 74 seconds) seconds when the initial guess is from any other iteration (the largest number of PDE solves for the trust-region scheme was 102 (82); combining this with the PDE solves required for solving the relaxed problem, the total number of PDE solves was 564 (647)). We note that the number of PDE solves is significantly lower than the total number of binary variables, indicating that our solution approach is efficient for solving these large-scale MINLPs.

In Table 4.4 we report the final objective value obtained from our algorithms. In the first three rows, the initial guess is obtained from rounding the relaxation solution. For

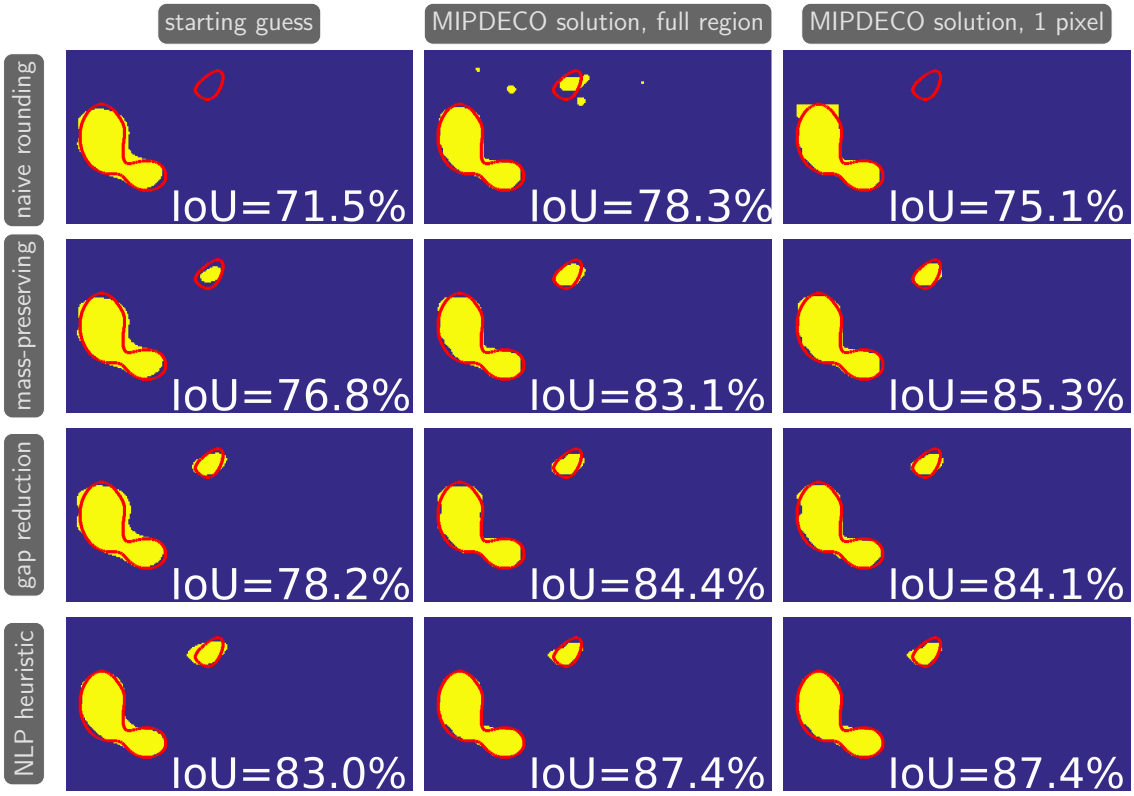


Figure 4.6: 2D results of the full-space and neighborhood variant of the trust-region approach for different rounding heuristics (row-wise). The left column shows the starting guess, obtained by rounding the solution of the relaxed problem at the α value selected from the L-curve. The middle and right columns depict the solutions obtained using the trust region methods. The superimposed red line indicates the location of the true source. The overlap is quantified using the intersection-over-union score (IoU) and printed in the lower-right corner of each image.

the NLP heuristic we used as a starting guess the final iteration solution with the naïve rounding, because the other two rounding schemes also result in the same integer feasible solution (due to the very small integrality gap). Using the proposed trust-region heuristic, We obtain a percentage improvement of 132%, 82.67%, 139.39%, and 23.83% in the objective values of naïve, mass-preserving, gap-reduction, and NLP heuristic rounding solutions, respectively, for 2D case; and of 18.84%, 26.57%, and 18.84% in the objective values of naïve, mass-preserving, and gap-reduction solutions, respectively, for the 3D case.

Figure 4.8 shows the progress in objective value and the change in the trust-region radius for the MIPDECO instances. Here we use the first iteration of the penalty method as an initial guess. We observe that the trust-region algorithm terminates in a modest number of iterations (typically in the range [25, 51]), which implies that we solved at most twice the PDEs to obtain function and adjoint information (we do not need to solve the

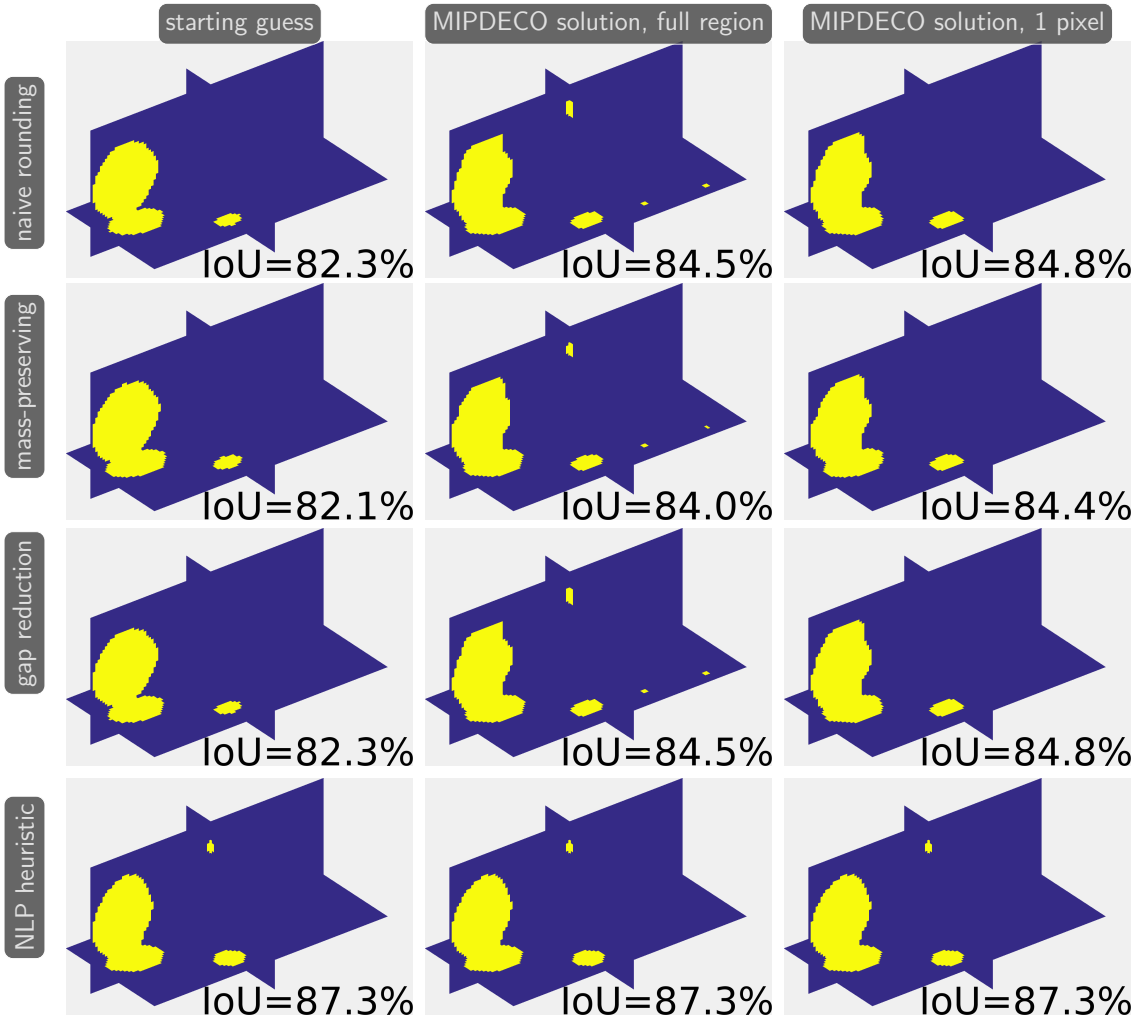


Figure 4.7: 3D results of the full-space and neighborhood variant of the trust-region approach for different rounding heuristics (row-wise). The left column shows the starting guess, obtained by rounding the solution of the relaxed problem at the α value selected from the L-curve. The middle and right columns depict the solutions obtained using the trust region methods. The overlap is quantified using the IoU and printed in the lower-right corner of each image.

adjoint equation on iterations on which we reject the step). These results are encouraging, given that the bulk of the computational effort is the initial factorization of the stiffness matrix, which we do once during the solution of the relaxed discretized MIPDECO and after which we can reuse the factors for fast PDE solves. The reduction in the function value that we obtain is also encouraging, showing that we can significantly improve the objective value in our trust-region iterations.

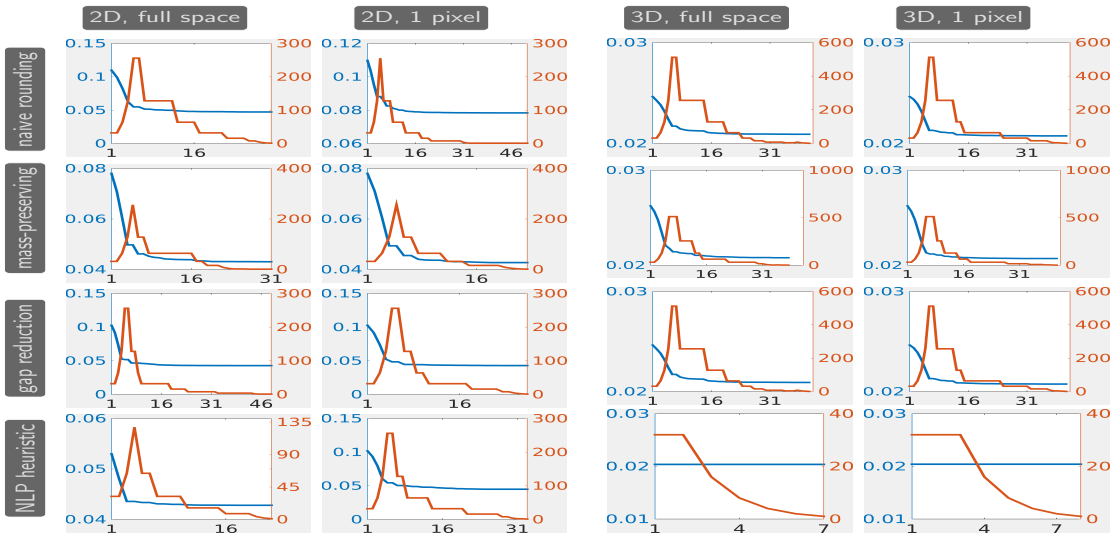


Figure 4.8: Convergence histories of the MIPDECO heuristics. Each plot shows the values of the objective function (blue line, left y-axis) and the trust-region radius (red line, right y-axis) at each iteration (x-axis). The rows correspond to the instances obtained for the naïve, the mass-preserving, and the gap-reduction rounding, respectively. The columns represent the 2D and 3D results of the full-space and reduced space methods.

4.6 Conclusions

We apply several solution approaches to a discrete source inversion problem for the convection-diffusion equation. We discretize the given mixed-integer PDE-constrained optimization (MIPDECO) problem using finite elements and obtain a large-scale convex MINLP. Using numerical examples, we demonstrate that the discretization of this problem can be solved neither by rounding solutions of the relaxed problem nor by state-of-the-art MINLP solvers. We propose a new heuristic for MIPDECO that combines a problem-specific rounding scheme with an improvement heuristic. The method is motivated by trust-region methods for nonlinear optimization and is related to the neighborhood search and local-branching heuristics for MINLP.

We show that our proposed heuristic can solve both 2D and 3D problem instances with more than 65,000 binary variables. In particular, our full-space trust-region approach can add sources even if the initial guess misses an existing source. The algorithm solves at most two PDEs per iteration, and our Julia implementation reuses factorizations of the stiffness matrix for computational efficiency. In most cases, the trust-region approach converges in a modest number of iterations (often around 30).

There are several ways to further improve the efficiency of MINLP solvers, which use IPOPT to solve the continuous relaxations and the nodes in the branch-and-bound

tree. Because IPOPT is a general-purpose framework for solving optimization problems, it does not take advantage of the structure of the discretized PDE. In particular, IPOPT refactors the stiffness matrix on every iteration, although in principle one could rewrite the linear algebra inside IPOPT to take advantage of these factors. On the other hand, the PDECO solver jInv is geared toward PDE-constrained problems and includes a number of choices that reduce the runtime for the specific instance. Since in the problem at hand the PDE-operator does not depend on the optimization variable, our jInv uses a direct method to factorize the stiffness matrix before solving the relaxed problem. While computing the factorization in 3D takes a significant amount of time, subsequent evaluations of the objective function, gradients, and matrix-vector products with the Hessians can be computed quickly.

Chapter 5

Conclusions and Future Work

Although convex MINLPs are theoretically hard to solve, instances of reasonable sizes can be solved in practice and faster by exploiting special mathematical structures (or properties) present in their formulations. Thus, enlarging the scope of improvement in commercial and open-source solvers for convex MINLPs. One such property is convexity, which has been shown extremely helpful in obtaining useful problem reformulations and polyhedral approximations.

In Chapter 2, we enhance the performance of a branch-and-cut based QG method by adding extra valid linear inequalities at appropriate nodes of its branch-and-bound tree. Tight relaxations, especially in the early stages of the branch-and-bound tree, prove quite useful in reducing the overall size of the tree and solution times. Our computational experiments using proposed linearization schemes in the QG algorithm report that, starting with a tight root node relaxation reduces the distance between the root LP solution and the feasible region of the continuous relaxation at the root node by a far greater extent than the reduction achieved in the solution times. Also, exploiting special structures like the univariate structure (US) in nonlinear constraints has a more significant impact than general-purpose routines for generating additional linearizations. Overall, the proposed linearization schemes have shown favorable results in terms of solution times and size of the tree in both serial and parallel versions of the QG algorithm in the open-source solver MINOTAUR.

In Chapter 3, we emphasize that convex MINLPs can be solved faster by exploiting specific mathematical structures present in them. Reformulations that result by exploiting these structures provide tighter relaxations or tighter polyhedral approximations. We discuss problem reformulations using two structures: on-off sets and separable nonlinear functions. Though finding these structures in the problem (P) is challenging in general, we present computationally efficient ways to detect collections of constraints that appear

as small blocks in (P) and are amenable to these reformulations. By automatically exploiting such reformulations in the branch-and-cut framework of QG in MINOTAUR, we could solve many instances faster and solve some instances that could not be solved earlier. Furthermore, we present that reformulations obtained by exploiting nonlinear function separability result in structures that become amenable to perspective reformulations, further enhancing the performance in many instances.

Chapter 4 presents a MIPDECO problem, a relatively new and challenging class of optimization problems. Most of the known approaches for solving these problems lead to optimization problems with a large number of variables and an extensive system of equations (the discretized PDEs) that describe the underlying physics. We apply several solution approaches to a discrete source inversion problem for the convection-diffusion equation. We discretize the given MIPDECO problem using the finite element approach and obtain a large-scale convex MINLP. Using numerical examples, we demonstrate that the discretization of this problem could neither be solved by rounding the solutions of the relaxed problem nor by deploying state-of-the-art MINLP solvers directly. The primary reason for this inability of the solvers is that they are not designed to exploit the discretized PDE structure. We propose a new heuristic for such problems that combines a problem-specific rounding scheme with an improvement heuristic. The method is motivated by the trust-region methods for nonlinear optimization and is related to the neighborhood search and local-branching heuristics for MINLPs. Using our modified trust-region method, we show that three-dimensional instances of MIPDECO can be solved efficiently and in a reasonable amount of time.

Following are some future research directions that involve some natural extension of the presented work to more general MINLPs and the scope of applying other tools like machine learning into such algorithmic frameworks.

- **Structures exploitation in general MINLPs**: Nonconvex MINLPs involve more general nonlinear functions than convex MINLPs. Automatic detection of known structures and finding new structures, especially whose convex hull description is easy to find, are beneficial to advance the area of general MINLPs. With this objective in mind, we seek to first narrow down on special impactful structures in a wide range of mathematical formulations and find efficient ways to automate the identification of such structures in general MINLPs possibly.

- **MILP based bounding in MINLP algorithms**: Solving MILP relaxations (instead of LPs) gives better dual bounds, and their solutions can be used as starting points in the search heuristics and to solve fixed-NLPs to improve the primal bounds.

These observations, along with advancements in state-of-art MILP solvers, have motivated researchers to solve MILPs more frequently within the branch-and-cut framework for solving MINLPs (Kılınç and Sahinidis (2018)). However, when one should solve a more expensive relaxation like an MILP is crucial to ensure effective computational resource usage. BARON (Sahinidis (1996)), another state-of-art solver for MINLPs, decides to solve an MILP at a node based only on the feasibility of integer solutions available from its ancestor nodes. We want to explore strategies that leverage the tightness of an LP relaxation at a node to decide if an MILP should be solved or not. The tightness of an LP relaxation can be estimated using some violation measure of the nonlinear constraints at the LP solution at that node.

- **Learning in the algorithmic framework of MINLPs**: In the last decade, machine learning is being used as a useful tool in combinatorial optimization for many decisions - branching variable selection (Khalil *et al.* (2016)), node selection (He *et al.* (2014)), parameter settings in solvers (Hutter *et al.* (2010)), primal heuristics (Khalil *et al.* (2017)), cutting planes (Tang *et al.* (2019)), whether to solve an MILP or an MIQP etc. - that are adaptive to the progress of the solution method. We are interested in leveraging learning in decisions that can be benefited from data generated during the MINLP solving, namely identifying good portions of the feasible region and infeasible domains, generating and managing valid inequalities, deciding when and which type of relaxations (LP, NLP, or a MILP) to be solved, etc.

Appendix A

Mathematical Preliminaries

A *vector* of dimension n is denoted by $x \in \mathbb{R}^n$, which is a point with n components $(x_1, \ldots, x_n)$. Also, by a vector we mean a column vector and $x^\top$ represents transpose of x. We use $\mathbb{R}^n$ and $\mathbb{Z}^n$ to denote the set of all n-dimensional vectors (or points) of real and integer values, respectively, and $\{0, 1\}^n$ means collection of all n-dimensional vectors of values 0 or 1. Given two n-dimensional vectors x and y, $x^\top y$ is called *inner (or dot) product* of x and y, and is given by $x^\top y = \sum_{i=1}^{n} x_i y_i$. Also, $x^\top y = y^\top x$. If b is a *scalar*, then $b \in \mathbb{R}$. A quantity is *non-negative* means it is greater than or equal to 0 and positive means it is strictly greater than 0. A non-negative vector denoted as $x \geq 0, x \in \mathbb{R}^n$, means every component takes a non-negative value. A nonzero vector suggests at least one component takes a nonzero value. Symbols $\subseteq$, $\subset$, $\forall$, $\exists$ mean subset, proper subset, for all, and there exists, respectively. Let $S \subseteq \mathbb{R}^n$ and $T \subseteq \mathbb{R}^n$ be two sets.

- If S is an empty set, then it is denoted by $\emptyset$.
- $x \in S$ means that x is an element of the set S, $x \notin S$ indicates that x is not an element of S.
- $S^C = \{x \in \mathbb{R}^n | x \notin S\}$ is called complement of set S.
- If the number of elements in a set S is finite, then $|S|$ denotes the cardinality (number of elements) of the set S.
- $S \cup T$ represents their union, $S \cap T$ denotes their intersection, and $T \backslash S$ is the set of elements that are in T but not in S.
- For a $x \in S$ and positive scalar $\delta \in \mathbb{R}$, $N_x(\delta)$ is called *δ-neighborhood* of x, which is defined as the set of elements that are within δ distance (in Euclidean norm, $\|.\|_2$) from x, that is, $N_x(\delta) = \{y \in \mathbb{R}^n | \, \|x - y\|_2 < \delta\}$.

Definition A.0.1. *A set $S \subseteq \mathbb{R}^n$ is called*

(a) open if for every $x \in S$ there exists a δ-neighborhood $N_x(\delta)$ such that $N_x(\delta) \subset S$.

(b) closed if its complement S^C is open.

(c) bounded if there exists a positive scalar r such that for every x, y in S, $\|x - y\|_2 < r$.

(d) compact if and only is it is closed and bounded.

The intersection of any number of closed sets, and the union of any finite collection of closed sets is a closed set.

Definition A.0.2. *Let $a \in \mathbb{R}^n$ be a nonzero vector and b be a scalar,*

(a) a halfspace is a set of the form $\{x \in \mathbb{R}^n | a^\top x \leq b\}$.

(b) a hyperplane is a set of the form $\{x \in \mathbb{R}^n | a^\top x = b\}$.

Vector a in the definition of the hyperplane is orthogonal to the hyperplane.

Every hyperplane can be written as an intersection of two halfspaces, $\{x \in \mathbb{R}^n | a^\top x \leq b\}$ and $\{x \in \mathbb{R}^n | -a^\top x \leq -b\}$. A halfspace is a closed set, thus, a hyperplane is also a closed set.

Definition A.0.3. *A polyhedron is a set of the form $\{x \in \mathbb{R}^n | \, a_i^\top x \leq b_i, i = 1, \ldots, m\}$, where vector $a_i \in \mathbb{R}^n$, b is a scalar, and m is a non-negative integer.*

In the compact form, a polyhedron can be represented as $\{x \in \mathbb{R}^n | Ax \leq b\}$, where A is an $m \times n$ matrix and $b \in \mathbb{R}^m$. Vector $a_i^\top$ forms ith row of matrix A. In addition, a polyhedron can be seen as an intersection of finite number of halfspaces, that implies that a polyhedron is also a closed set.

Definition A.0.4. *Given a finite number of n-dimensional real vectors $x^1, \ldots, x^k$,*

(a) a convex combination of these vector is a vector of the form

$$x = \lambda_1 x^1 + \ldots + \lambda_k x^k,$$

where $\lambda_i \geq 0, \forall i = 1, \ldots, k$ and $\sum_{i=1}^k \lambda_i = 1$.

(b) a convex hull of these vectors is the set of all the convex combinations of them.

Definition A.0.5. *A set $S \subseteq \mathbb{R}^n$ is convex if all the convex combinations of any two points $x, y \in S$ also lie in the set S.*

All the convex combinations between two points constitute the line segment between these points. Thus, a convex set contains the line segment between every pair of points in it. Convex hull of any set S is always a convex set and is denoted by $conv(S)$. Also, it is the smallest convex set that contains the set S. In addition, if S is a convex set, then $conv(S) = S$.

Theorem A.0.6. *Intersection of convex sets is a convex set.*

A halfspace is a convex set and, thus, every polyhedron is also a convex set.
Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real-values function, then

- the *gradient* ∇f and *Hessian* $\nabla^2 f$ of function f are the vector of its first partial derivatives and matrix of its second partial derivatives, respectively. Mathematically,

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \quad \text{and} \quad \nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial {x_2}^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial {x_n}^2} \end{pmatrix}.$$

- a point $x \in \mathbb{R}^n$ is called a *stationary point* if $\nabla f(x) = 0$.

Definition A.0.7. *Let $S \subseteq \mathbb{R}^n$ be a convex set and function $f : S \to \mathbb{R}$.*

(a) f is convex if for every $x, y \in S$ and $\lambda \in [0, 1]$ following holds

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y). \tag{A.1}$$

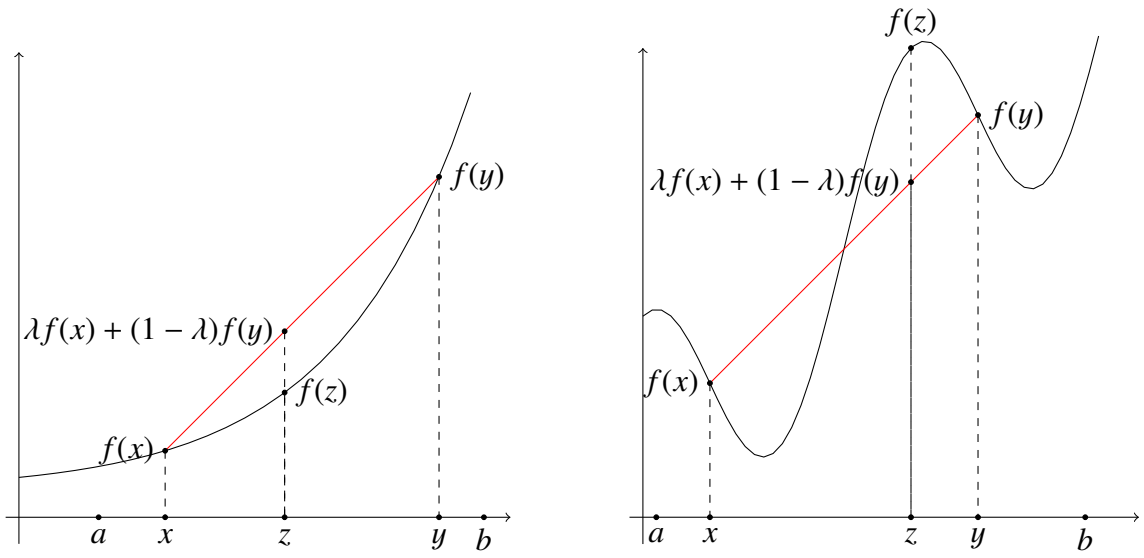
(b) α-sublevel set of f is a set of the form

$$S_\alpha = \{x \in S \,|\, f(x) \leq \alpha\},$$

where $\alpha \in \mathbb{R}$.

Equation (A.1) suggests that for a convex function, the graph of the function between every pair of points in its domain lies no higher than the line segment between these two points. Figure 4.1 shows its graphical interpretation.

If the sign of the inequality in A.1 is reversed, then the function becomes concave. Thus, if f is convex, $-f$ is concave, and vice-versa. All sublevel sets of a convex function are convex sets, but the converse is false. In this thesis, we work majorly with convex and



(a) (b)

Figure A.1: An example of (Left) a convex function and (Right) a nonconvex function on set $S = [a, b]$. Here $z = \lambda x + (1 - \lambda)y, \lambda \in [0, 1]$ is a convex combination of points $x, y \in S$.

smooth functions. We call a function smooth if it is twice-continuously differentiable on its domain.

A $n \times n$ dimensional matrix A is called positive definite if $x^\top Ax > 0$ for all nonzero vector $x \in \mathbb{R}^n$, and positive semidefinite if $x^\top Ax \geq 0$ for all $x \in \mathbb{R}^n$. For a convex function, Hessian at every point in its domain is positive semidefinite.

A useful mathematical tool for studying smooth functions is *Taylor's theorem*. This theorem allows one to derive useful approximations of a smooth function.

Theorem A.0.8. *(Nocedal and Wright (2000)) Suppose that function $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and $p \in \mathbb{R}^n$. Then*

$$f(x + p) = f(x) + p^\top \nabla f(x + tp),$$

for some $t \in (0, 1)$.

Definition A.0.9. *The first-order approximation or linear approximation of a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $y \in \mathbb{R}^n$ is given by*

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

Another useful result for a convex function is that every linear approximation underestimates it. Theory and algorithms of mathematical optimization involving convex functions make extensive use of this result.

Theorem A.0.10. *A differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if for every $x, y \in \mathbb{R}^n$*

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

and, f is strictly convex if and only if

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

An *underestimator of a function* $f : S \to \mathbb{R}$ is another function $\underline{f}(x) : S \to \mathbb{R}$ such that $\underline{f}(x) \leq f(x), \forall x \in S$.

Solving an optimization problem means finding a point where the objective function takes the minimum value among all the allowed values that the objective function can assume. Formally, the notion of minimizer can be understood as follows.

Definition A.0.11. *Given a set $S \subseteq \mathbb{R}^n$ and a function $f : S \to \mathbb{R}$*

(a) a point x^* is a local minimizer of f over S if there exists a δ-neighborhood of x^*, $N_{x^*}(\delta)$, such that $f(x^*) \leq f(x), \forall x \in N_{x^*}(\delta)$.

(b) a point x^* is a global minimizer of f over S if $f(x^*) \leq f(x), \forall x \in S$.

A minimizer (local or global) is called a *strict minimizer* if inequality in Definition A.0.11 holds at strict inequality. For a general optimization problem finding a global minimizer can be difficult. Most algorithms can find only a local minimizer. However, specific properties help in identifying global minimizer. One such property is the convexity of the objective function and the feasible region.

Theorem A.0.12. *(Nocedal and Wright (2000)) When $f : S \to \mathbb{R}$ is convex, any local minimizer x^* is a global minimizer of f. If in addition, f is differentiable, then any stationary point $x^* \in S$ is a global minimizer of f.*

Theorem A.0.12 enables algorithms for convex optimization to seek a point where gradient vanishes.

Appendix B

Description of Test Instances

We have considered 334 convex MINLP instances from the benchmarking library (Bussieck *et al.* (2003)) in our computational experiments in Chapter 2 and Chapter 3. Following is a brief description of the applications and problems associated with these instances. Symbol * following the name of a problem denotes a collection of instances in that class.

- `alan`: A mixed-binary quadratic program (MBQP) for mean-variance portfolio selection problem (Manne (1986)).
- `ball_mk`*: MINLPs representing intersection of a ball $B(\rho, r) \subset \mathbb{Z}^n$ of radius $r = \sqrt{(n-1)}/2$ centered at $\rho = (1/2, \ldots, 1/2)$ with the vertices of the unit hypercube $\{0,1\}^n$ (Hijazi *et al.* (2014)). These examples are constructed to study the worst-case complexity of multi-tree methods presented in Chapter 1. These are infeasible instances in which the outer-approximation method visits all the integer combinations in the feasible region to establish infeasibility in the problem.
- `batch`*: Mixed-binary nonlinear programs (MBNLPs) of batch processing from the chemical processing industry. They model multi-product batch plant design problems with multiple units in parallel and intermediate storage tanks. The objective of these problems is to determine the volume of the equipment, the number of units in parallel, and the volume and location of the intermediate storage tanks. These problems have a single nonlinear constraint, and a nonlinear objective involving an exponential term (Ravemark and Rippin (1998); Vecchietti and Grossmann (1999)).
- `clay`*: Constrained layout problems where non-overlapping rectangular units are to be located within designated areas while minimizing the cost of connecting these units (Sawaya and Grossmann (2008)). These problems are deliberately poorly

modeled to ensure no feasible solution lies near the optimal solution of the continuous relaxation of the problem. The instances with the suffix 'h' are MBNLPs, and 'm' are mixed-binary quadratically constrained problems (MBQCPs).

- `cvxnonsep`*: Fabricated instances to test solvers' capability to apply log and power transformations and exploit nonlinear function separability. They have non-separable functions that are highly nonlinear (in the sense that the Hessian of the Lagrange function is dense). Outer-approximation and cutting-plane algorithms are ineffective on these problems. The reformulated problems are separable and have suffix `'r'`. Problems with `'nsig'` and `'psig'` in their names are amenable to log transformation and `'normcon'` and `'pcon'` to power transformation (Kronqvist *et al.* (2018b)). Problems with suffix `'r'` and `'normcon'` in their names are mixed-integer quadratically constrained problems (MIQCPs) and others are MINLPs.

- `du-opt`*: Mixed-integer quadratic programs (MIQPs) and `enpro` are MBNLPs from batch processing application.

- `ex1223, ex1223b, gams01, ibs2, ravempb, risk2b, st_e14` (MBNLPs), `gbd, netmod`* (MBQPs), `ex1223a` (MBQCQP) are modified textbook examples (Floudas *et al.* (1999)), `nvs`* (IQCQPs), `st_miqp1, st_miqp2, st_miqp3, st_test`* (IQPs), `st_miqp4, st_miqp5` (MIQPs).

- `ex4`: Process synthesis problem (MBQCQP) that aims to determine the optimal structure and operating conditions of a process while satisfying given design specifications (Duran and Grossmann (1986)).

- `fac`* Multi-commodity capacity facility location-allocation problems. `fac3` is a MBQP and others are MBNLPs.

- `flay`* : Farmland layout problems (MBNLPs). These problems determine the optimal length and width of rectangular patches of land with the fixed area while minimizing the set of patches' perimeter (Sawaya and Grossmann (2008)).

- `fo7, fo7_2, fo8, fo9, m3, m6, m7, o7, o7_2`: Facility layout design problem from manufacturing and service organizations (MBNLP). In this facility layout problem, the objective is to find a non-overlapping arrangement of some rectangular departments within a given rectangular facility while minimizing some distance based measures and satisfying size and area requirements (Meller *et al.* (1998)).

- `fo7*`, `fo8*`,`fo9*`, `m7*`, `no7*`, `o7*`, `o8_ar4_1`, `o9_ar4_1`: Block layout design problems (MINLPs). Very similar in spirit to facility layout design problems, these problems strive to find the most efficient arrangement of a given number of departments with unequal area requirements within a facility (Castillo *et al.* (2005)).

- `hybriddynamic_fixed`: Fixed length finite element discretization of the mathematical program with equilibrium constraints for the hybrid dynamic systems (MBQP) (Baumrucker and Biegler (2009)).

- `jit1`: A MINLP from the just-in-time manufacturing systems. The objective of this problem is to minimize the total cost of production, imbalance, and investment while determining the number of machines for each stage of a multi-stage production process (Gunasekaran *et al.* (1993); Gutierrez and Sahinidis (1996)).

- `meanvar*`: Standard mean-variance model from financial operations (MBQP) (Dahl *et al.* (1989)).

- `portfol*`: Portfolio optimization problems. `portfol_buyin` (MBNLP) is a deterministic version of probabilistic mean-variance portfolio selection problems that aim to construct a portfolio with minimal risk, and a prescribed return level (Bonami and Lejeune (2009)). `portfol_card` (MBNLP) finds a portfolio by minimizing a utility function constructed based on the investor's preferences (Ehrgott *et al.* (2009)). `portfol_classical*` (MBQCP) conic quadratic programming of the classical mean-variance problem (Vielma *et al.* (2008)). `portfol_roundlot` (MINLP) is a modification mean-variable models to incorporate requirements to buy assets in large lots (Bonami and Lejeune (2009)).

- `procurement2mot`: A MBNLP of the production planning decisions in a chemical process network. It involves optimal contract selection under uncertainty with suppliers and product selling price optimization (Calfa and Grossmann (2015)).

- `rsyn*`: Retrofit-synthesis problems (MBNLPs). They involve redesigning existing plants to increase throughput, reduce energy consumption, improve yields, and reduce waste generation. Simultaneously, it synthesizes new plants. The objective is to identify design decisions that yield highest economic improvement over the desired time horizon with limited capital investments (Sawaya and Grossmann (2008)).

- `slay*`: Safety Layout problems (MBQPs). These problems consist of placing a set of units with fixed width and length so that the Euclidean distance between their

center points and a prespecified point (called safety point) is minimized (Sawaya and Grossmann (2008)).

- `smallinvDAX`*: Small Investor Mean-Variance Portfolio Optimization problems (MICQPs). They include constraints for small investors in the standard Markovitz Mean-Variance-Optimization model.

- `squfl`*: Separable quadratic uncapacitated facility location problems (MBQPs) as described in (UFL) in Chapter 1 (Günlük *et al.* (2007)).

- `sssd`*: Stochastic service system design problems (MBNLPs). The objective in these problems is to minimize the operating and assignment cost associated with allocating servers (operating at different service levels) to a set of customers (Elhedhli (2006)).

- `stockcycle`: Inventory management problems (MBNLPs) that aim to minimize the total average cycle cost subject to constraints on reordering intervals and the total number of replenishments (Silver and Moon (1999)).

- `syn`*: Synthesis problems (MBNLPs) that involve selecting optimal configuration and parameters for synthesizing a processing system (Duran and Grossmann (1986); Türkay and Grossmann (1996)).

- `tls`*: Cutting stock or trim loss problems (MBNLPs). with the objective of meeting the demand of a set of paper rolls from the raw rolls while minimizing the trim loss and overproduction (Harjunkoski *et al.* (1998)).

- `unitcommit`*: Short-term unit commitment problem in hydro-thermal power generation systems (MBQPs). These problems aim to find a set of hydro and thermal generating units over a specified time horizon to satisfy a forecasted energy demand at the minimum total cost (Zondervan and Grossmann (2009)).

- `watercontamination`*: Discretized inverse problems for finding contamination sources in municipal water networks (MBQPs) (Laird *et al.* (2006)).

Appendix C

Test Sets for Linearization Schemes

Table C.1: Description of instances in the test set TS_l for linearization schemes in Chapter 2. The first column contains the name of the instance. For the instance on the first column, the second column indicates the test set (TS_1 or TS_2) to which it belongs, third column reports whether the objective function is nonlinear (1) or not (0), and the last column contains the number of nonlinear constraints.

Instance	Set	O	C
alan	TS_2	1	0
ball_mk2_10	TS_2	0	1
ball_mk2_30	TS_2	0	1
ball_mk3_10	TS_2	0	1
ball_mk3_20	TS_2	0	1
ball_mk3_30	TS_2	0	1
ball_mk4_05	TS_2	0	1
ball_mk4_10	TS_2	0	1
ball_mk4_15	TS_2	0	1
batch0812	TS_2	1	1
batchdes	TS_2	1	1
batch	TS_2	1	1
batchs101006m	TS_2	1	1
batchs121208m	TS_2	1	1
batchs151208m	TS_2	1	1
batchs201210m	TS_2	1	1
clay0203h	TS_2	0	24
clay0203m	TS_2	0	24
clay0204h	TS_2	0	32
clay0204m	TS_2	0	32
clay0205h	TS_2	0	40
clay0205m	TS_2	0	40
clay0303h	TS_2	0	36
clay0303m	TS_2	0	36
clay0304h	TS_2	0	48
clay0304m	TS_2	0	48
clay0305h	TS_2	0	60
clay0305m	TS_2	0	60

Instance	Set	O	C
cvxnonsep_normcon20	TS_2	0	1
cvxnonsep_normcon20r	TS_1	0	20
cvxnonsep_normcon30	TS_2	0	1
cvxnonsep_normcon30r	TS_1	0	30
cvxnonsep_normcon40	TS_2	0	1
cvxnonsep_normcon40r	TS_1	0	40
cvxnonsep_nsig20	TS_2	0	1
cvxnonsep_nsig20r	TS_1	0	20
cvxnonsep_nsig30	TS_2	0	1
cvxnonsep_nsig30r	TS_1	0	30
cvxnonsep_nsig40	TS_2	0	1
cvxnonsep_nsig40r	TS_1	0	40
cvxnonsep_pcon20	TS_2	0	1
cvxnonsep_pcon20r	TS_2	0	19
cvxnonsep_pcon30	TS_2	0	1
cvxnonsep_pcon30r	TS_2	0	29
cvxnonsep_pcon40	TS_2	0	1
cvxnonsep_pcon40r	TS_2	0	39
cvxnonsep_psig20	TS_2	1	0
cvxnonsep_psig20r	TS_1	0	21
cvxnonsep_psig30	TS_2	1	0
cvxnonsep_psig30r	TS_1	0	31
cvxnonsep_psig40	TS_2	1	0
cvxnonsep_psig40r	TS_1	0	41
du-opt5	TS_2	1	0
du-opt	TS_2	1	0
enpro48pb	TS_2	1	1
enpro56pb	TS_2	1	1

Instance	Set	O	C
ex1223a	TS_1	1	4
ex1223b	TS_2	1	4
ex1223	TS_2	1	4
ex4	TS_2	1	25
fac1	TS_2	1	0
fac2	TS_2	1	0
fac3	TS_2	1	0
flay02h	TS_1	0	2
flay02m	TS_1	0	2
flay03h	TS_1	0	3
flay03m	TS_1	0	3
flay04h	TS_1	0	4
flay04m	TS_1	0	4
flay05h	TS_1	0	5
flay05m	TS_1	0	5
flay06h	TS_1	0	6
flay06m	TS_1	0	6
fo7_2	TS_1	0	14
fo7_ar2_1	TS_1	0	14
fo7_ar25_1	TS_1	0	14
fo7_ar3_1	TS_1	0	14
fo7_ar4_1	TS_1	0	14
fo7_ar5_1	TS_1	0	14
fo7	TS_1	0	14
fo8_ar2_1	TS_1	0	16
fo8_ar25_1	TS_1	0	16
fo8_ar3_1	TS_1	0	16
fo8_ar4_1	TS_1	0	16
fo8_ar5_1	TS_1	0	16
fo8	TS_1	0	16
fo9_ar2_1	TS_1	0	18
fo9_ar25_1	TS_1	0	18
fo9_ar3_1	TS_1	0	18
fo9_ar4_1	TS_1	0	18
fo9_ar5_1	TS_1	0	18
fo9	TS_1	0	18
gams01	TS_2	1	110
gbd	TS_2	1	0
hybriddynamic_fixed	TS_2	1	0
ibs2	TS_2	0	10
jit1	TS_2	1	0
m3	TS_1	0	6
m6	TS_1	0	12
m7_ar2_1	TS_1	0	14
m7_ar25_1	TS_1	0	14
m7_ar3_1	TS_1	0	14
m7_ar4_1	TS_1	0	14
m7_ar5_1	TS_1	0	14

Instance	Set	O	C
m7	TS_1	0	14
meanvarx	TS_2	1	0
meanvarxsc	TS_2	1	0
netmod_dol1	TS_2	1	0
netmod_dol2	TS_2	1	0
netmod_kar1	TS_2	1	0
netmod_kar2	TS_2	1	0
no7_ar2_1	TS_1	0	14
no7_ar25_1	TS_1	0	14
no7_ar3_1	TS_1	0	14
no7_ar4_1	TS_1	0	14
no7_ar5_1	TS_1	0	14
nvs03	TS_1	1	1
nvs10	TS_2	1	2
nvs11	TS_2	1	3
nvs12	TS_2	1	4
nvs15	TS_2	1	0
o7_2	TS_1	0	14
o7_ar2_1	TS_1	0	14
o7_ar25_1	TS_1	0	14
o7_ar3_1	TS_1	0	14
o7_ar4_1	TS_1	0	14
o7_ar5_1	TS_1	0	14
o7	TS_1	0	14
o8_ar4_1	TS_1	0	16
o9_ar4_1	TS_1	0	18
portfol_buyin	TS_2	0	2
portfol_card	TS_2	0	2
portfol_classical050_1	TS_2	0	1
portfol_classical200_2	TS_2	0	1
portfol_roundlot	TS_2	0	2
procurement2mot	TS_1	0	12
ravempb	TS_2	1	1
risk2bpb	TS_2	1	0
rsyn0805h	TS_2	0	3
rsyn0805m02h	TS_2	0	6
rsyn0805m02m	TS_1	0	6
rsyn0805m03h	TS_2	0	9
rsyn0805m03m	TS_1	0	9
rsyn0805m04h	TS_2	0	12
rsyn0805m04m	TS_1	0	12
rsyn0805m	TS_1	0	3
rsyn0810h	TS_2	0	6
rsyn0810m02h	TS_2	0	12
rsyn0810m02m	TS_1	0	12
rsyn0810m03h	TS_2	0	18
rsyn0810m03m	TS_1	0	18
rsyn0810m04h	TS_2	0	24

Instance	Set	O	C
rsyn0810m04m	TS_1	0	24
rsyn0810m	TS_1	0	6
rsyn0815h	TS_2	0	11
rsyn0815m02h	TS_2	0	22
rsyn0815m02m	TS_1	0	22
rsyn0815m03h	TS_2	0	33
rsyn0815m03m	TS_1	0	33
rsyn0815m04h	TS_2	0	44
rsyn0815m04m	TS_1	0	44
rsyn0815m	TS_1	0	11
rsyn0820h	TS_2	0	14
rsyn0820m02h	TS_2	0	28
rsyn0820m02m	TS_1	0	28
rsyn0820m03h	TS_2	0	42
rsyn0820m03m	TS_1	0	42
rsyn0820m04h	TS_2	0	56
rsyn0820m04m	TS_1	0	56
rsyn0820m	TS_1	0	14
rsyn0830h	TS_2	0	20
rsyn0830m02h	TS_2	0	40
rsyn0830m02m	TS_1	0	40
rsyn0830m03h	TS_2	0	60
rsyn0830m03m	TS_1	0	60
rsyn0830m04h	TS_2	0	80
rsyn0830m04m	TS_1	0	80
rsyn0830m	TS_1	0	20
rsyn0840h	TS_2	0	28
rsyn0840m02h	TS_2	0	56
rsyn0840m02m	TS_1	0	56
rsyn0840m03h	TS_2	0	84
rsyn0840m03m	TS_1	0	84
rsyn0840m04h	TS_2	0	112
rsyn0840m04m	TS_1	0	112
rsyn0840m	TS_1	0	28
slay04h	TS_2	1	0
slay04m	TS_2	1	0
slay05h	TS_2	1	0
slay05m	TS_2	1	0
slay06h	TS_2	1	0
slay06m	TS_2	1	0
slay07h	TS_2	1	0
slay07m	TS_2	1	0
slay08h	TS_2	1	0
slay08m	TS_2	1	0
slay09h	TS_2	1	0
slay09m	TS_2	1	0
slay10h	TS_2	1	0
slay10m	TS_2	1	0

Instance	Set	O	C
smallinvDAXr1b010-011	TS_2	0	1
smallinvDAXr1b020-022	TS_2	0	1
smallinvDAXr1b050-055	TS_2	0	1
smallinvDAXr1b100-110	TS_2	0	1
smallinvDAXr1b150-165	TS_2	0	1
smallinvDAXr1b200-220	TS_2	0	1
smallinvDAXr2b010-011	TS_2	0	1
smallinvDAXr2b020-022	TS_2	0	1
smallinvDAXr2b050-055	TS_2	0	1
smallinvDAXr2b100-110	TS_2	0	1
smallinvDAXr2b150-165	TS_2	0	1
smallinvDAXr2b200-220	TS_2	0	1
smallinvDAXr3b010-011	TS_2	0	1
smallinvDAXr3b020-022	TS_2	0	1
smallinvDAXr3b050-055	TS_2	0	1
smallinvDAXr3b100-110	TS_2	0	1
smallinvDAXr3b150-165	TS_2	0	1
smallinvDAXr3b200-220	TS_2	0	1
smallinvDAXr4b010-011	TS_2	0	1
smallinvDAXr4b020-022	TS_2	0	1
smallinvDAXr4b050-055	TS_2	0	1
smallinvDAXr4b100-110	TS_2	0	1
smallinvDAXr4b150-165	TS_2	0	1
smallinvDAXr4b200-220	TS_2	0	1
smallinvDAXr5b010-011	TS_2	0	1
smallinvDAXr5b020-022	TS_2	0	1
smallinvDAXr5b050-055	TS_2	0	1
smallinvDAXr5b100-110	TS_2	0	1
smallinvDAXr5b150-165	TS_2	0	1
smallinvDAXr5b200-220	TS_2	0	1
squfl010-025	TS_2	1	0
squfl010-040	TS_2	1	0
squfl010-080	TS_2	1	0
squfl015-060	TS_2	1	0
squfl015-080	TS_2	1	0
squfl020-040	TS_2	1	0
squfl020-050	TS_2	1	0
squfl020-150	TS_2	1	0
squfl025-025	TS_2	1	0
squfl025-030	TS_2	1	0
squfl025-040	TS_2	1	0
squfl030-100	TS_2	1	0
squfl030-150	TS_2	1	0
squfl040-080	TS_2	1	0
sssd08-04	TS_1	0	12
sssd12-05	TS_1	0	15
sssd15-04	TS_1	0	12
sssd15-06	TS_1	0	18

Instance	Set	O	C
sssd15-08	TS_1	0	24
sssd16-07	TS_1	0	21
sssd18-06	TS_1	0	18
sssd18-08	TS_1	0	24
sssd20-04	TS_1	0	12
sssd20-08	TS_1	0	24
sssd22-08	TS_1	0	24
sssd25-04	TS_1	0	12
sssd25-08	TS_1	0	24
st_e14	TS_2	1	4
st_miqp2	TS_2	1	0
st_miqp3	TS_2	1	0
st_miqp4	TS_2	1	0
st_miqp5	TS_2	1	0
stockcycle	TS_2	1	0
st_test3	TS_2	1	0
st_test4	TS_2	1	0
st_test8	TS_2	1	0
st_testgr1	TS_2	1	0
st_testgr3	TS_2	1	0
st_testph4	TS_2	1	0
syn05h	TS_2	0	3
syn05m02h	TS_2	0	6
syn05m02m	TS_1	0	6
syn05m03h	TS_2	0	9
syn05m03m	TS_1	0	9
syn05m04h	TS_2	0	12
syn05m04m	TS_1	0	12
syn05m	TS_1	0	3
syn10h	TS_2	0	6
syn10m02h	TS_2	0	12
syn10m02m	TS_1	0	12
syn10m03h	TS_2	0	18
syn10m03m	TS_1	0	18
syn10m04h	TS_2	0	24
syn10m04m	TS_1	0	24
syn10m	TS_1	0	6
syn15h	TS_2	0	11
syn15m02h	TS_2	0	22
syn15m02m	TS_1	0	22
syn15m03h	TS_2	0	33
syn15m03m	TS_1	0	33
syn15m04h	TS_2	0	44

Instance	Set	O	C
syn15m04m	TS_1	0	44
syn15m	TS_1	0	11
syn20h	TS_2	0	14
syn20m02h	TS_2	0	28
syn20m02m	TS_1	0	28
syn20m03h	TS_2	0	42
syn20m03m	TS_1	0	42
syn20m04h	TS_2	0	56
syn20m04m	TS_1	0	56
syn20m	TS_1	0	14
syn30h	TS_2	0	20
syn30m02h	TS_2	0	40
syn30m02m	TS_1	0	40
syn30m03h	TS_2	0	60
syn30m03m	TS_1	0	60
syn30m04h	TS_2	0	80
syn30m04m	TS_1	0	80
syn30m	TS_1	0	20
syn40h	TS_2	0	28
syn40m02h	TS_2	0	56
syn40m02m	TS_1	0	56
syn40m03h	TS_2	0	84
syn40m03m	TS_1	0	84
syn40m04h	TS_2	0	112
syn40m04m	TS_1	0	112
syn40m	TS_1	0	28
synthes1	TS_2	1	2
synthes2	TS_1	1	3
synthes3	TS_1	1	4
tls12	TS_2	0	12
tls2	TS_2	0	2
tls4	TS_2	0	4
tls5	TS_2	0	5
tls6	TS_2	0	6
tls7	TS_2	0	7
unitcommit1	TS_2	1	0
unitcommit_50_20_2_mod_8	TS_2	1	0
unitcommit_200_100_1_mod_8	TS_2	1	0
unitcommit_200_100_2_mod_8	TS_2	1	0
watercontamination0202	TS_2	1	0
watercontamination0202r	TS_2	1	0
watercontamination0303	TS_2	1	0
watercontamination0303r	TS_2	1	0

Appendix D

Test Sets for Reformulations Techniques

Table D.1: Description of instances with collections (C_1), (C_2), and (C_3) in Chapter 3. First column shows instance name and the entries $(bv, tv, fv, b0, b1, b01, v0, v1, v01)$ in the second column are: bv denotes the number of binary variables, tv indicates the total number of variables, fv reports the number of binary variables that are fixed as part of structure identification, $b0$ and $b1$ represent the number of binary variables z and $1-z$, respectively, controlling at least one other variable, $b01$ denotes number of binary variables z such that both z and $1-z$ control another variable, $v0$ and $v1$ report the number of variables controlled by a binary variable z and $1-z$ respectively, $v01$ is the number of variables controlled by a binary variable z and also $1-z$.

Instance	$(bv, tv, fv, b0, b1, b01, v0, v1, v01)$
alan	(4, 4, 0, 4, 0, 0, 4, 0, 0)
batch0812	(60, 60, 28, 0, 32, 0, 0, 70, 0)
batchdes	(9, 9, 1, 0, 8, 0, 0, 12, 0)
batch	(24, 24, 2, 0, 22, 0, 0, 30, 0)
batchs101006m	(120, 120, 0, 0, 120, 0, 0, 140, 0)
batchs121208m	(191, 191, 0, 0, 191, 0, 0, 215, 0)
batchs151208m	(188, 188, 0, 0, 188, 0, 0, 212, 0)
batchs201210m	(225, 225, 0, 0, 225, 0, 0, 249, 0)
clay0203h	(18, 18, 0, 0, 0, 18, 60, 12, 6)
clay0203m	(18, 18, 0, 0, 12, 6, 0, 12, 6)
clay0204h	(32, 32, 0, 0, 0, 32, 112, 24, 8)
clay0204m	(32, 32, 0, 0, 24, 8, 0, 24, 8)
clay0205h	(50, 50, 0, 0, 0, 50, 180, 40, 10)
clay0205m	(50, 50, 0, 0, 40, 10, 0, 40, 10)
clay0303h	(21, 21, 0, 0, 0, 21, 66, 21, 0)
clay0303m	(21, 21, 0, 0, 21, 0, 0, 21, 0)
clay0304h	(36, 36, 0, 0, 0, 36, 120, 36, 0)
clay0304m	(36, 36, 0, 0, 36, 0, 0, 36, 0)
clay0305h	(55, 55, 0, 0, 0, 55, 190, 55, 0)
clay0305m	(55, 55, 0, 0, 55, 0, 0, 55, 0)
color_lab2_4x0	(28920, 28920, 0, 0, 28680, 240, 28680, 240, 0)
color_lab6b_4x20	(27730, 27730, 0, 0, 27495, 235, 27495, 235, 0)
enpro48pb	(92, 92, 0, 0, 92, 0, 0, 108, 0)
enpro56pb	(73, 73, 0, 0, 73, 0, 0, 85, 0)
fac1	(6, 6, 0, 2, 0, 4, 16, 0, 4)
fac2	(12, 12, 0, 3, 0, 9, 54, 0, 9)
fac3	(12, 12, 0, 3, 0, 9, 54, 0, 9)
flay02h	(4, 4, 0, 0, 0, 4, 32, 4, 0)
flay02m	(4, 4, 0, 0, 4, 0, 0, 4, 0)

Instance	$(bv, tv, fv, b0, b1, b01, v0, v1, v01)$
flay03h	(12, 12, 0, 0, 0, 12, 96, 12, 0)
flay03m	(12, 12, 0, 0, 12, 0, 0, 12, 0)
flay04h	(24, 24, 0, 0, 0, 24, 192, 24, 0)
flay04m	(24, 24, 0, 0, 24, 0, 0, 24, 0)
flay05h	(40, 40, 0, 0, 0, 40, 320, 40, 0)
flay05m	(40, 40, 0, 0, 40, 0, 0, 40, 0)
flay06h	(60, 60, 0, 0, 0, 60, 480, 60, 0)
flay06m	(60, 60, 0, 0, 60, 0, 0, 60, 0)
gams01	(110, 110, 0, 0, 100, 0, 0, 100, 0)
hybriddynamic_fixed	(1, 1, 0, 0, 0, 1, 8, 0, 2)
ibs2	(1500, 1500, 0, 0, 0, 1500, 1500, 1500, 0)
meanvarx	(12, 12, 0, 2, 0, 10, 12, 10, 0)
meanvarxsc	(22, 22, 0, 12, 0, 10, 12, 10, 0)
netmod_dol1	(462, 462, 0, 0, 0, 462, 1524, 462, 0)
netmod_dol2	(455, 455, 0, 0, 79, 367, 973, 440, 6)
netmod_kar1	(136, 136, 0, 0, 15, 121, 255, 136, 0)
netmod_kar2	(136, 136, 0, 0, 15, 121, 255, 136, 0)
pedigree_ex1058	(49386, 49386, 0, 112, 48387, 865, 48387, 112, 865)
pedigree_ex485_2	(7136, 7136, 0, 110, 6710, 294, 6710, 110, 294)
pedigree_ex485	(7136, 7136, 0, 110, 6710, 294, 6710, 110, 294)
pedigree_sim400	(11226, 11226, 0, 51, 11076, 99, 11076, 51, 99)
pedigree_sp_top4_250	(11694, 11694, 0, 243, 10981, 414, 10981, 243, 414)
pedigree_sp_top4_300	(5969, 5969, 0, 160, 5496, 244, 5496, 160, 244)
pedigree_sp_top4_350tr	(3100, 3100, 0, 105, 2838, 145, 2838, 105, 145)
pedigree_sp_top5_200	(32120, 32120, 0, 336, 30862, 871, 30862, 336, 871)
pedigree_sp_top5_250	(17028, 17028, 0, 243, 16193, 536, 16193, 243, 536)
portfol_buyin	(8, 8, 0, 8, 0, 0, 8, 0, 0)
portfol_card	(8, 8, 0, 8, 0, 0, 8, 0, 0)
portfol_classical050_1	(50, 50, 0, 50, 0, 0, 50, 0, 0)

Instance	$(bv, tv, fv, b0, b1, b01, v0, v1, v01)$
portfol_classical200_2	(200, 200, 0, 200, 0, 0, 200, 0, 0)
procurement2mot	(60, 60, 0, 19, 3, 38, 77, 18, 23)
ravempb	(53, 53, 0, 0, 53, 0, 0, 65, 0)
risk2bpb	(12, 12, 0, 0, 12, 0, 0, 183, 0)
rsyn0805h	(37, 37, 0, 3, 0, 34, 84, 32, 26)
rsyn0805m02h	(148, 148, 0, 3, 0, 145, 171, 37, 166)
rsyn0805m02m	(148, 148, 0, 3, 64, 81, 19, 53, 118)
rsyn0805m03h	(222, 222, 0, 3, 0, 219, 255, 42, 264)
rsyn0805m03m	(222, 222, 0, 3, 96, 123, 27, 66, 192)
rsyn0805m04h	(296, 296, 0, 3, 0, 293, 339, 47, 362)
rsyn0805m04m	(296, 296, 0, 3, 128, 165, 35, 79, 266)
rsyn0805m	(37, 37, 0, 3, 32, 2, 8, 32, 2)
rsyn0810h	(41, 41, 0, 3, 0, 38, 95, 34, 26)
rsyn0810m02h	(166, 166, 0, 3, 0, 163, 187, 47, 182)
rsyn0810m02m	(166, 166, 0, 3, 64, 99, 35, 63, 134)
rsyn0810m03h	(249, 249, 0, 3, 0, 246, 278, 57, 289)
rsyn0810m03m	(249, 249, 0, 3, 96, 150, 50, 81, 217)
rsyn0810m04h	(332, 332, 0, 3, 0, 329, 369, 67, 396)
rsyn0810m04m	(332, 332, 0, 3, 128, 201, 65, 99, 300)
rsyn0810m	(41, 41, 0, 3, 32, 6, 19, 34, 2)
rsyn0815h	(44, 44, 0, 3, 0, 41, 105, 35, 27)
rsyn0815m02h	(182, 182, 0, 3, 0, 179, 204, 57, 197)
rsyn0815m02m	(182, 182, 0, 3, 64, 115, 52, 73, 149)
rsyn0815m03h	(273, 273, 0, 3, 0, 270, 303, 72, 312)
rsyn0815m03m	(273, 273, 0, 3, 96, 174, 75, 96, 240)
rsyn0815m04h	(364, 364, 0, 3, 0, 361, 402, 87, 427)
rsyn0815m04m	(364, 364, 0, 3, 128, 233, 98, 119, 331)
rsyn0815m	(44, 44, 0, 3, 32, 9, 29, 35, 3)
rsyn0820h	(49, 49, 0, 3, 0, 46, 116, 35, 29)
rsyn0820m02h	(202, 202, 0, 3, 0, 199, 223, 67, 214)
rsyn0820m02m	(202, 202, 0, 3, 64, 135, 71, 83, 166)
rsyn0820m03h	(303, 303, 0, 3, 0, 300, 330, 87, 339)
rsyn0820m03m	(303, 303, 0, 3, 96, 204, 102, 111, 267)
rsyn0820m04h	(404, 404, 0, 3, 0, 401, 437, 107, 464)
rsyn0820m04m	(404, 404, 0, 3, 128, 273, 133, 139, 368)
rsyn0820m	(49, 49, 0, 3, 32, 14, 40, 35, 5)
rsyn0830h	(58, 58, 0, 6, 0, 52, 136, 37, 30)
rsyn0830m02h	(240, 240, 0, 6, 0, 234, 259, 90, 243)
rsyn0830m02m	(240, 240, 0, 6, 64, 170, 107, 106, 195)
rsyn0830m03h	(360, 360, 0, 6, 0, 354, 381, 120, 387)
rsyn0830m03m	(360, 360, 0, 6, 96, 258, 153, 144, 315)
rsyn0830m04h	(480, 480, 0, 6, 0, 474, 503, 150, 531)
rsyn0830m04m	(480, 480, 0, 6, 128, 346, 199, 182, 435)
rsyn0830m	(58, 58, 0, 6, 32, 20, 60, 37, 6)
rsyn0840h	(66, 66, 0, 6, 0, 60, 157, 38, 33)
rsyn0840m02h	(276, 276, 0, 6, 0, 270, 295, 110, 275)
rsyn0840m02m	(276, 276, 0, 6, 64, 206, 143, 126, 227)
rsyn0840m03h	(414, 414, 0, 6, 0, 408, 433, 150, 437)
rsyn0840m03m	(414, 414, 0, 6, 96, 312, 205, 174, 365)
rsyn0840m04h	(552, 552, 0, 6, 0, 546, 571, 190, 599)
rsyn0840m04m	(552, 552, 0, 6, 128, 418, 267, 222, 503)
rsyn0840m	(66, 66, 0, 6, 32, 28, 81, 38, 9)

Instance	$(bv, tv, fv, b0, b1, b01, v0, v1, v01)$
slay04h	(24, 24, 0, 0, 0, 24, 96, 24, 0)
slay04m	(24, 24, 0, 0, 24, 0, 0, 24, 0)
slay05h	(40, 40, 0, 0, 0, 40, 160, 40, 0)
slay05m	(40, 40, 0, 0, 40, 0, 0, 40, 0)
slay06h	(60, 60, 0, 0, 0, 60, 240, 60, 0)
slay06m	(60, 60, 0, 0, 60, 0, 0, 60, 0)
slay07h	(84, 84, 0, 0, 0, 84, 336, 84, 0)
slay07m	(84, 84, 0, 0, 84, 0, 0, 84, 0)
slay08h	(112, 112, 0, 0, 0, 112, 448, 112, 0)
slay08m	(112, 112, 0, 0, 112, 0, 0, 112, 0)
slay09h	(144, 144, 0, 0, 0, 144, 576, 144, 0)
slay09m	(144, 144, 0, 0, 144, 0, 0, 144, 0)
slay10h	(180, 180, 0, 0, 0, 180, 720, 180, 0)
slay10m	(180, 180, 0, 0, 180, 0, 0, 180, 0)
squfl010-025	(10, 10, 0, 10, 0, 0, 250, 0, 0)
squfl010-040	(10, 10, 0, 10, 0, 0, 400, 0, 0)
squfl010-080	(10, 10, 0, 10, 0, 0, 800, 0, 0)
squfl015-060	(15, 15, 0, 15, 0, 0, 900, 0, 0)
squfl015-080	(15, 15, 0, 15, 0, 0, 1200, 0, 0)
squfl020-040	(20, 20, 0, 20, 0, 0, 800, 0, 0)
squfl020-050	(20, 20, 0, 20, 0, 0, 1000, 0, 0)
squfl020-150	(20, 20, 0, 20, 0, 0, 3000, 0, 0)
squfl025-025	(25, 25, 0, 25, 0, 0, 625, 0, 0)
squfl025-030	(25, 25, 0, 25, 0, 0, 750, 0, 0)
squfl025-040	(25, 25, 0, 25, 0, 0, 1000, 0, 0)
squfl030-100	(30, 30, 0, 30, 0, 0, 3000, 0, 0)
squfl030-150	(30, 30, 0, 30, 0, 0, 4500, 0, 0)
squfl040-080	(40, 40, 0, 40, 0, 0, 3200, 0, 0)
sssd08-04	(44, 44, 0, 0, 32, 12, 12, 44, 0)
sssd12-05	(75, 75, 0, 0, 60, 15, 15, 75, 0)
sssd15-04	(72, 72, 0, 0, 60, 12, 12, 72, 0)
sssd15-06	(108, 108, 0, 0, 90, 18, 18, 108, 0)
sssd15-08	(144, 144, 0, 0, 120, 24, 24, 144, 0)
sssd16-07	(133, 133, 0, 0, 112, 21, 21, 133, 0)
sssd18-06	(126, 126, 0, 0, 108, 18, 18, 126, 0)
sssd18-08	(168, 168, 0, 0, 144, 24, 24, 168, 0)
sssd20-04	(92, 92, 0, 0, 80, 12, 12, 92, 0)
sssd20-08	(184, 184, 0, 0, 160, 24, 24, 184, 0)
sssd22-08	(200, 200, 0, 0, 176, 24, 24, 200, 0)
sssd25-04	(112, 112, 0, 0, 100, 12, 12, 112, 0)
sssd25-08	(224, 224, 0, 0, 200, 24, 24, 224, 0)
st_miqp2	(2, 2, 0, 2, 0, 0, 2, 0, 0)
st_miqp4	(2, 2, 0, 2, 0, 0, 2, 0, 0)
stockcycle	(432, 432, 0, 0, 432, 0, 0, 480, 0)
st_test3	(5, 5, 0, 3, 0, 0, 3, 0, 0)
syn05h	(5, 5, 0, 3, 0, 2, 8, 0, 2)
syn05m02h	(20, 20, 0, 3, 0, 17, 19, 13, 14)
syn05m02m	(20, 20, 0, 3, 0, 17, 19, 13, 14)
syn05m03h	(30, 30, 0, 3, 0, 27, 27, 18, 24)
syn05m03m	(30, 30, 0, 3, 0, 27, 27, 18, 24)
syn05m04h	(40, 40, 0, 3, 0, 37, 35, 23, 34)
syn05m04m	(40, 40, 0, 3, 0, 37, 35, 23, 34)

Instance	$(bv, tv, fv, b0, b1, b01, v0, v1, v01)$
syn05m	(5, 5, 0, 3, 0, 2, 8, 0, 2)
syn10h	(9, 9, 0, 3, 0, 6, 19, 2, 2)
syn10m02h	(38, 38, 0, 3, 0, 35, 35, 23, 30)
syn10m02m	(38, 38, 0, 3, 0, 35, 35, 23, 30)
syn10m03h	(57, 57, 0, 3, 0, 54, 50, 33, 49)
syn10m03m	(57, 57, 0, 3, 0, 54, 50, 33, 49)
syn10m04h	(76, 76, 0, 3, 0, 73, 65, 43, 68)
syn10m04m	(76, 76, 0, 3, 0, 73, 65, 43, 68)
syn10m	(9, 9, 0, 3, 0, 6, 19, 2, 2)
syn15h	(12, 12, 0, 3, 0, 9, 29, 3, 3)
syn15m02h	(54, 54, 0, 3, 0, 51, 52, 33, 45)
syn15m02m	(54, 54, 0, 3, 0, 51, 52, 33, 45)
syn15m03h	(81, 81, 0, 3, 0, 78, 75, 48, 72)
syn15m03m	(81, 81, 0, 3, 0, 78, 75, 48, 72)
syn15m04h	(108, 108, 0, 3, 0, 105, 98, 63, 99)
syn15m04m	(108, 108, 0, 3, 0, 105, 98, 63, 99)
syn15m	(12, 12, 0, 3, 0, 9, 29, 3, 3)
syn20h	(17, 17, 0, 3, 0, 14, 40, 3, 5)
syn20m02h	(74, 74, 0, 3, 0, 71, 71, 43, 62)
syn20m02m	(74, 74, 0, 3, 0, 71, 71, 43, 62)
syn20m03h	(111, 111, 0, 3, 0, 108, 102, 63, 99)
syn20m03m	(111, 111, 0, 3, 0, 108, 102, 63, 99)
syn20m04h	(148, 148, 0, 3, 0, 145, 133, 83, 136)
syn20m04m	(148, 148, 0, 3, 0, 145, 133, 83, 136)
syn20m	(17, 17, 0, 3, 0, 14, 40, 3, 5)
syn30h	(26, 26, 0, 6, 0, 20, 60, 5, 6)
syn30m02h	(112, 112, 0, 6, 0, 106, 107, 66, 91)
syn30m02m	(112, 112, 0, 6, 0, 106, 107, 66, 91)
syn30m03h	(168, 168, 0, 6, 0, 162, 153, 96, 147)

Instance	$(bv, tv, fv, b0, b1, b01, v0, v1, v01)$
syn30m03m	(168, 168, 0, 6, 0, 162, 153, 96, 147)
syn30m04h	(224, 224, 0, 6, 0, 218, 199, 126, 203)
syn30m04m	(224, 224, 0, 6, 0, 218, 199, 126, 203)
syn30m	(26, 26, 0, 6, 0, 20, 60, 5, 6)
syn40h	(34, 34, 0, 6, 0, 28, 81, 6, 9)
syn40m02h	(148, 148, 0, 6, 0, 142, 143, 86, 123)
syn40m02m	(148, 148, 0, 6, 0, 142, 143, 86, 123)
syn40m03h	(222, 222, 0, 6, 0, 216, 205, 126, 197)
syn40m03m	(222, 222, 0, 6, 0, 216, 205, 126, 197)
syn40m04h	(296, 296, 0, 6, 0, 290, 267, 166, 271)
syn40m04m	(296, 296, 0, 6, 0, 290, 267, 166, 271)
syn40m	(34, 34, 0, 6, 0, 28, 81, 6, 9)
synthes1	(3, 3, 0, 0, 1, 1, 1, 2, 0)
synthes2	(5, 5, 0, 1, 1, 3, 3, 2, 2)
synthes3	(8, 8, 0, 3, 1, 4, 8, 3, 2)
tls12	(489, 489, 0, 12, 465, 12, 0, 504, 129)
tls2	(31, 31, 0, 2, 29, 0, 0, 18, 17)
tls4	(85, 85, 0, 4, 81, 0, 0, 76, 25)
tls5	(131, 131, 0, 5, 126, 0, 0, 125, 31)
tls6	(165, 165, 0, 6, 159, 0, 0, 156, 45)
tls7	(278, 278, 0, 7, 271, 0, 0, 266, 61)
unitcommit1	(427, 427, 0, 9, 235, 179, 310, 196, 83)
unitcommit_200_100_1_mod_8	(4380, 4380, 0, 3843, 0, 537, 13245, 398, 190)
unitcommit_200_100_2_mod_8	(4400, 4400, 0, 3969, 0, 431, 13148, 530, 230)
unitcommit_50_20_2_mod_8	(1093, 1093, 0, 991, 0, 102, 3259, 132, 58)
watercontamination0202	(7, 7, 0, 7, 0, 0, 521, 0, 0)
watercontamination0202r	(7, 7, 0, 7, 0, 0, 188, 0, 0)
watercontamination0303	(14, 14, 0, 14, 0, 0, 1046, 0, 0)
watercontamination0303r	(14, 14, 0, 14, 0, 0, 370, 0, 0)

Table D.2: Description of test set T_{pr} of 104 instances with structures, (PS_1) and (PS_2), amenable to perspective reformulation in the Chapter 3. The entry in the first column is the instance name and for each instance the entries in the second column are as follows: ts denotes total number of nonlinear constraints, pc shows the number of PR amenable constraints, $s1$ and $s2$ report number of constraints (out of pc) of type (S_1) and (S_2), respectively, and the last entry ub denotes the number of unique variables associated with PR amenable constraints.

Instance	(tc, pc, $s1$, $s2$)
clay0203h	(24, 24, 24, 0)
clay0204h	(32, 32, 32, 0)
clay0205h	(40, 40, 40, 0)
clay0303h	(36, 36, 36, 0)
clay0304h	(48, 48, 48, 0)
clay0305h	(60, 60, 60, 0)
rsyn0805h	(3, 3, 3, 0)
rsyn0805m02h	(6, 6, 6, 0)
rsyn0805m02m	(6, 6, 6, 0)
rsyn0805m03h	(9, 9, 9, 0)
rsyn0805m03m	(9, 9, 9, 0)
rsyn0805m04h	(12, 12, 12, 0)
rsyn0805m04m	(12, 12, 12, 0)
rsyn0805m	(3, 3, 3, 0)
rsyn0810h	(6, 6, 6, 0)
rsyn0810m02h	(12, 12, 12, 0)
rsyn0810m02m	(12, 12, 12, 0)
rsyn0810m03h	(18, 18, 18, 0)
rsyn0810m03m	(18, 18, 18, 0)
rsyn0810m04h	(24, 24, 24, 0)
rsyn0810m04m	(24, 24, 24, 0)
rsyn0810m	(6, 6, 6, 0)
rsyn0815h	(11, 11, 11, 0)
rsyn0815m02h	(22, 22, 22, 0)
rsyn0815m02m	(22, 22, 22, 0)
rsyn0815m03h	(33, 33, 33, 0)
rsyn0815m03m	(33, 33, 33, 0)
rsyn0815m04h	(44, 44, 44, 0)
rsyn0815m04m	(44, 44, 44, 0)
rsyn0815m	(11, 11, 11, 0)
rsyn0820h	(14, 14, 14, 0)
rsyn0820m02h	(28, 28, 28, 0)
rsyn0820m02m	(28, 28, 28, 0)
rsyn0820m03h	(42, 42, 42, 0)
rsyn0820m03m	(42, 42, 42, 0)

Instance	(tc, pc, $s1$, $s2$)
rsyn0820m04h	(56, 56, 56, 0)
rsyn0820m04m	(56, 56, 56, 0)
rsyn0820m	(14, 14, 14, 0)
rsyn0830h	(20, 20, 20, 0)
rsyn0830m02h	(40, 40, 40, 0)
rsyn0830m02m	(40, 40, 40, 0)
rsyn0830m03h	(60, 60, 60, 0)
rsyn0830m03m	(60, 60, 60, 0)
rsyn0830m04h	(80, 80, 80, 0)
rsyn0830m04m	(80, 80, 80, 0)
rsyn0830m	(20, 20, 20, 0)
rsyn0840h	(28, 28, 28, 0)
rsyn0840m02h	(56, 56, 56, 0)
rsyn0840m02m	(56, 56, 56, 0)
rsyn0840m03h	(84, 84, 84, 0)
rsyn0840m03m	(84, 84, 84, 0)
rsyn0840m04h	(112, 112, 112, 0)
rsyn0840m04m	(112, 112, 112, 0)
rsyn0840m	(28, 28, 28, 0)
syn05h	(3, 3, 3, 0)
syn05m02h	(6, 6, 6, 0)
syn05m02m	(6, 6, 6, 0)
syn05m03h	(9, 9, 9, 0)
syn05m03m	(9, 9, 9, 0)
syn05m04h	(12, 12, 12, 0)
syn05m04m	(12, 12, 12, 0)
syn05m	(3, 3, 3, 0)
syn10h	(6, 6, 6, 0)
syn10m02h	(12, 12, 12, 0)
syn10m02m	(12, 12, 12, 0)
syn10m03h	(18, 18, 18, 0)
syn10m03m	(18, 18, 18, 0)
syn10m04h	(24, 24, 24, 0)
syn10m04m	(24, 24, 24, 0)
syn10m	(6, 6, 6, 0)

Instance	(tc, pc, $s1$, $s2$)
syn15h	(11, 11, 11, 0)
syn15m02h	(22, 22, 22, 0)
syn15m02m	(22, 22, 22, 0)
syn15m03h	(33, 33, 33, 0)
syn15m03m	(33, 33, 33, 0)
syn15m04h	(44, 44, 44, 0)
syn15m04m	(44, 44, 44, 0)
syn15m	(11, 11, 11, 0)
syn20h	(14, 14, 14, 0)
syn20m02h	(28, 28, 28, 0)
syn20m02m	(28, 28, 28, 0)
syn20m03h	(42, 42, 42, 0)
syn20m03m	(42, 42, 42, 0)
syn20m04h	(56, 56, 56, 0)
syn20m04m	(56, 56, 56, 0)
syn20m	(14, 14, 14, 0)
syn30h	(20, 20, 20, 0)
syn30m02h	(40, 40, 40, 0)
syn30m02m	(40, 40, 40, 0)
syn30m03h	(60, 60, 60, 0)
syn30m03m	(60, 60, 60, 0)
syn30m04h	(80, 80, 80, 0)
syn30m04m	(80, 80, 80, 0)
syn30m	(20, 20, 20, 0)
syn40h	(28, 28, 28, 0)
syn40m02h	(56, 56, 56, 0)
syn40m02m	(56, 56, 56, 0)
syn40m03h	(84, 84, 84, 0)
syn40m03m	(84, 84, 84, 0)
syn40m04h	(112, 112, 112, 0)
syn40m04m	(112, 112, 112, 0)
syn40m	(28, 28, 28, 0)
synthes2	(3, 1, 1, 0)
synthes3	(4, 2, 1, 1)

Table D.3: Description of the instances in the test set TS_{sep} for separability based reformulation in the Chapter 3. First column shows the instance name and the entries (nc, sp, os, us, rs) in the second column are: nc and sc number of nonlinear constraints and number of separable nonlinear constraints, respectively, os indicates whether objective function is separabe (1) or not (0), us is the number of unique separable parts considering all separable constraints and objective function, rs is the number of separable parts that are repeated. 26 instances also belonging to the test set TS_{ps} (that became amenable to perspective reformulation after the reformulation based on separability of nonlinear constraints and objective) are highlighted in bold.

Instance	(nc, sp, os, us, rs)
ball_mk2_10	(1, 1, 0, 10, 0)
ball_mk2_30	(1, 1, 0, 30, 0)
ball_mk3_10	(1, 1, 0, 10, 0)
ball_mk3_20	(1, 1, 0, 20, 0)
ball_mk3_30	(1, 1, 0, 30, 0)
ball_mk4_05	(1, 1, 0, 5, 0)
ball_mk4_10	(1, 1, 0, 10, 0)
ball_mk4_15	(1, 1, 0, 15, 0)
batch0812	(2, 2, 0, 20, 0)
batchdes	(2, 2, 0, 5, 0)
batch	(2, 2, 0, 11, 0)
batchs101006m	(2, 2, 0, 29, 0)
batchs121208m	(2, 2, 0, 35, 0)
batchs151208m	(2, 2, 0, 38, 0)
batchs201210m	(2, 2, 0, 43, 0)
clay0203m	(24, 24, 0, 24, 24)
clay0204m	(32, 32, 0, 32, 32)
clay0205m	(40, 40, 0, 40, 40)
clay0303m	(36, 36, 0, 36, 36)
clay0304m	(48, 48, 0, 48, 48)
clay0305m	(60, 60, 0, 60, 60)
enpro48pb	(2, 2, 0, 13, 0)
enpro56pb	(2, 2, 0, 12, 0)
ex1223a	(5, 2, 0, 6, 0)
ex1223b	(5, 5, 0, 12, 5)
ex1223	(5, 5, 0, 12, 5)
ex4	(26, 26, 0, 125, 2)
fac1	(1, 0, 1, 2, 0)
fac2	(1, 0, 1, 3, 0)
fac3	(1, 0, 1, 3, 0)
gams01	(111, 0, 1, 10, 0)
hybriddynamic_fixed	(1, 0, 1, 11, 0)
immun	(1, 0, 1, 6, 0)
netmod_dol1	(1, 0, 1, 6, 0)
netmod_dol2	(1, 0, 1, 6, 0)
netmod_kar1	(1, 0, 1, 4, 0)
netmod_kar2	(1, 0, 1, 4, 0)
nvs03	(2, 0, 1, 2, 0)
nvs10	(3, 0, 1, 2, 0)
pedigree_ex1058	(1, 1, 0, 28, 0)
pedigree_ex485_2	(1, 1, 0, 28, 0)
pedigree_ex485	(1, 1, 0, 28, 0)
pedigree_sp_top4_250	(1, 1, 0, 58, 0)
pedigree_sp_top4_300	(1, 1, 0, 74, 0)
pedigree_sp_top4_350tr	(1, 1, 0, 17, 0)
pedigree_sp_top5_200	(1, 1, 0, 54, 0)
pedigree_sp_top5_250	(1, 1, 0, 58, 0)
portfol_classical050_1	(1, 1, 0, 50, 0)
portfol_classical200_2	(1, 1, 0, 200, 0)
risk2bpb	(1, 0, 1, 3, 0)
slay04h	(1, 0, 1, 8, 0)
slay04m	(1, 0, 1, 8, 0)
slay05h	(1, 0, 1, 10, 0)
slay05m	(1, 0, 1, 10, 0)
slay06h	(1, 0, 1, 12, 0)
slay06m	(1, 0, 1, 12, 0)
slay07h	(1, 0, 1, 14, 0)
slay07m	(1, 0, 1, 14, 0)
slay08h	(1, 0, 1, 16, 0)
slay08m	(1, 0, 1, 16, 0)
slay09h	(1, 0, 1, 18, 0)
slay09m	(1, 0, 1, 18, 0)
slay10h	(1, 0, 1, 20, 0)
slay10m	(1, 0, 1, 20, 0)
squfl010-025	(1, 0, 1, 250, 0)
squfl010-040	(1, 0, 1, 400, 0)
squfl010-080	(1, 0, 1, 800, 0)
squfl015-060	(1, 0, 1, 900, 0)
squfl015-080	(1, 0, 1, 1200, 0)
squfl020-040	(1, 0, 1, 800, 0)
squfl020-050	(1, 0, 1, 1000, 0)
squfl020-150	(1, 0, 1, 3000, 0)
squfl025-025	(1, 0, 1, 625, 0)
squfl025-030	(1, 0, 1, 750, 0)
squfl025-040	(1, 0, 1, 1000, 0)
squfl030-100	(1, 0, 1, 3000, 0)
squfl030-150	(1, 0, 1, 4500, 0)
squfl040-080	(1, 0, 1, 3200, 0)
st_e14	(5, 5, 0, 12, 5)
st_miqp1	(1, 0, 1, 5, 0)
st_miqp2	(1, 0, 1, 2, 0)
st_miqp4	(1, 0, 1, 3, 0)
st_miqp5	(1, 0, 1, 2, 0)
stockcycle	(1, 0, 1, 48, 0)
st_test1	(1, 0, 1, 4, 0)
st_test2	(1, 0, 1, 5, 0)
st_test3	(1, 0, 1, 5, 0)
st_test4	(1, 0, 1, 2, 0)
st_test5	(1, 0, 1, 7, 0)
st_test6	(1, 0, 1, 10, 0)
st_test8	(1, 0, 1, 24, 0)
st_testgr1	(1, 0, 1, 10, 0)
st_testgr3	(1, 0, 1, 20, 0)
st_testph4	(1, 0, 1, 3, 0)
synthes2	(4, 0, 1, 3, 0)
synthes3	(5, 2, 0, 6, 1)
tls12	(12, 12, 0, 144, 0)
tls2	(2, 2, 0, 4, 0)
tls4	(4, 4, 0, 16, 0)
tls5	(5, 5, 0, 25, 0)
tls6	(6, 6, 0, 36, 0)
tls7	(7, 7, 0, 49, 0)
unitcommit1	(1, 0, 1, 240, 0)
unitcommit_200_100_1_mod_8	(1, 0, 1, 4662, 0)
unitcommit_200_100_2_mod_8	(1, 0, 1, 4639, 0)
unitcommit_50_20_2_mod_8	(1, 0, 1, 1152, 0)
watercontamination0202	(1, 0, 1, 4017, 0)
watercontamination0303	(1, 0, 1, 4521, 0)

Appendix E

Discretization of Source Inversion Problem

E.1 Finite-Difference Discretization of Source Inversion Problem

We discretize both the source, w, and the state, u, in the cell-centered points of our $N_x \times N_y$ computational mesh as

$$W_{kl} \simeq w\left(kL_x - L_x/2, lL_y - L_y/2\right), \quad U_{ij} \simeq u\left(iL_x - L_x/2, jL_y - L_y/2\right),$$

where $L_x = 2/N_x$ and $L_y = 1/N_y$ are the discretization steps in the x and y direction, respectively, and $k = 1, \ldots, N_x, l = 1, \ldots, N_y$, $i = 0, \ldots, N_x + 1$, $j = 0, \ldots, N_y$. Here, the variables $U_{i,0}, U_{N_x+1,j}, U_{i,N_y+1}$ approximate the PDE solution at ghost points placed along Γ_N, and the variables $U_{0,j}$ are the ghost points near Γ_D.

Let us further define $V \in \mathbb{R}^m$ to obtain the bilinear interpolation from the cell-centered points of the mesh closest to the receiver locations $r^1, r^2, \ldots, r^m$. Mathematically, for each receiver location $r^k = (r_x^k, r_y^k), k = 1, \ldots, m$, we define variable V_k as

$$V_k = \frac{1}{L_x L_y} \begin{pmatrix} iL_x + \dfrac{L_x}{2} - r_x^k \\ r_x^k - iL_x + \dfrac{L_x}{2} \end{pmatrix}^T \begin{pmatrix} U_{i,j} & U_{i,j+1} \\ U_{i+1,j} & U_{i+1,j+1} \end{pmatrix} \begin{pmatrix} jL_y + \dfrac{L_y}{2} - r_y^k \\ r_y^k - jL_y + \dfrac{L_y}{2} \end{pmatrix}, \tag{E.1}$$

where, $i = \lfloor \frac{r_x^k}{L_x} + 0.5 \rfloor$ and $j = \lfloor \frac{r_y^k}{L_y} + 0.5 \rfloor$, leading to the mixed-integer quadratic program of the form

$$\left.\begin{aligned}
\underset{U,\,W}{\text{minimize}} \quad & \frac{1}{2\sigma}\Big(\textstyle\sum_{k=1}^{m}\big(V_k - b_k\big)^2\Big) \\
& + \alpha L_x L_y \textstyle\sum_{k=2}^{N_x}\sum_{l=2}^{N_y}\sqrt{\Big(\frac{W_{kl}-W_{(k-1)l}}{L_x}\Big)^2 + \Big(\frac{W_{kl}-W_{k(l-1)}}{L_y}\Big)^2 + \kappa}, \\
\text{subject to} \quad & c\frac{4U_{ij} - U_{(i-1)j} - U_{(i+1)j} - U_{i(j-1)} - U_{i(j+1)}}{{L_x}^2 {L_y}^2}, \\
& + \frac{U_{ij} - U_{(i-1)j}}{L_x} = W_{ij},\ i = 1,\dots,N_x,\ j = 1,\dots,N_y, \\
& U_{N_x+1,j} = U_{N_x,j},\ U_{i,0} = U_{i,1},\ U_{i,N_y+1} = U_{i,N_y}, U_{0,j} = -U_{1,j}, \\
& i = 0,\dots,N_x,\ j = 0,\dots,N_y, \\
& W \in \{0,1\}^{N_x \times N_y}, U \in \mathbb{R}^{(N_x+2)\times(N_y+2)}.
\end{aligned}\right\} \tag{FDM}$$

Here, the fifth row explicitly encodes the Neumann and Dirichlet boundary conditions. As before, we can again eliminate the state variables U using the discretized PDE and boundary conditions and the state variables V using (E.1), resulting in a problem that has similar structure to (4.4). As before, the objective function is second-order cone representable.

E.2 Selection of Regularization Parameter for Relaxed Problem

To find an effective regularization parameter, we consider the continuous relaxation (4.5) and follow the L-curve procedure; see Hansen (1998) for details. In the inversions we use coarser meshes with 256×128 and $96 \times 48 \times 48$ cells for the 2D instance and 3D instance, respectively. We consider the datasets generated in the preceding section and perturb the generated data with 10% iid Gaussian white noise.

To compute the L-curve, we solve 30 instances of the continuous relaxation for different values of α that are logarithmically spaced between 1 and 10^{-6}. To accelerate the computation, we initialize the optimization with the solution from the previous α. For each value of α, we use up to 20 Gauss-Newton iterations and approximately compute the search direction using 5 iterations of projected preconditioned CG that use the Hessian of the regularization function as a preconditioner. For each experiment, we store the reconstructed source, the predicted data, the value of the misfit function, and the values of the regularization function (without the factor α). The L-curve shown in Figure E.2 show the value of the regularizer and the value of the misfit of these optimal solutions.

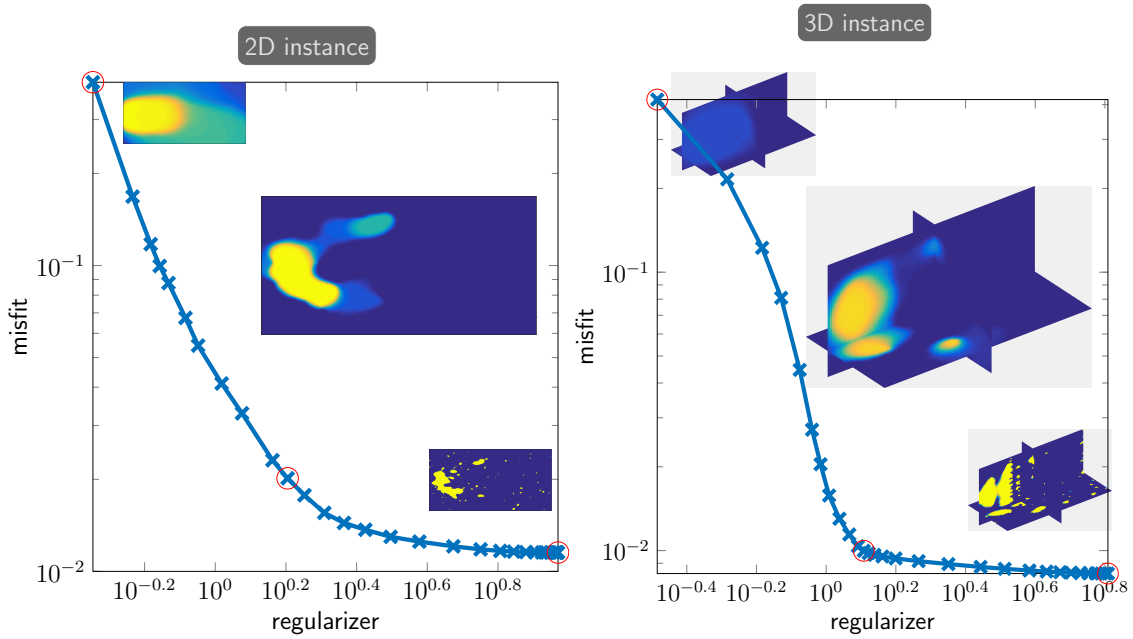


Figure E.1: L-curve plots for the relaxed optimization problem (4.5) for the two-dimensional instance (left) and three-dimensional instance (right). In both cases, we solve the relaxed problem for 30 α values logarithmically spaced between 1 and 10^{-6}. We plot the value of the regularizer and misfit at the computed solution in a loglog plot. To highlight the impact of α on the smoothness of the reconstructed images, we provide snapshots of the reconstructed source at the extremal values and one value that provides a good trade-off (values are marked with a circle).

As is common, the axes are scaled logarithmically; and to provide additional insight, we have added visualizations of the reconstructed sources for the largest and smallest value of α (resulting in overly smoothed and very noisy reconstructions, respectively) as well as solutions that provide a good trade-off. Using this process we select the regularization parameters $\alpha = 8.531 \cdot 10^{-3}$ for the two-dimensional instance and $\alpha = 5.298 \cdot 10^{-3}$ for the three-dimensional instance, respectively. Computing the L-curves took about 4 and 48 minutes and involved 32,580 and 30,892 PDE solves in 2D and 3D, respectively. The large number of PDE solves underscores the importance of computing a factorization (or a good preconditioner in large-scale problems) apriori.

Bibliography

Abhishek, K., Leyffer, S., and Linderoth, J. T., 2006, *FilMINT: An Outer-Approximation-Based Solver for Nonlinear Mixed Integer Programs*, Preprint ANL/MCS-P1374-0906 (Mathematics and Computer Science Division, Argonne National Laboratory).

Abhishek, K., 2008, *Topics in mixed integer nonlinear programming*, Ph.D. thesis (Lehigh University).

Achterberg, T., 2007, *Constraint integer programming*, Ph.D. thesis (Technical University Berlin).

Achterberg, T., 2009, "SCIP: Solving constraint integer programs," *Mathematical Programming Computation* **1**, 1–41.

Achterberg, T., Bixby, R. E., Gu, Z., Rothberg, E., and Weninger, D., 2020, "Presolve reductions in mixed integer programming," *INFORMS Journal on Computing* **32**, 473–506.

Achterberg, T., Koch, T., and Martin, A., 2005, "Branching rules revisited," *Operations Research Letters* **33**, 42–54.

Achterberg, T., and Wunderling, R., 2013, "Mixed integer programming: Analyzing 12 years of progress," in *Facets of combinatorial optimization* (Springer). pp. 449–481.

Agnetis, A., Grande, E., and Pacifici, A., 2012, "Demand allocation with latency cost functions," *Mathematical programming* **132**, 277–294.

Ahmadi-Javid, A., and Hoseinpour, P., 2019, "Service system design for managing interruption risks: A backup-service risk-mitigation strategy," *European Journal of Operational Research* **274**, 417–431.

Akcelik, V., Biros, G., Draganescu, A., Ghattas, O., Hill, J., and van Bloemen Waanders, B., 2005, "Dynamic data-driven inversion for terascale simulations: Real-time identification of airborne contaminants," in *Proceedings of SC2005, Seattle, WA*.

Akçelik, V., Biros, G., Ghattas, O., Hill, J., Keyes, D., and van Bloemen Waanders, B., 2006, "Parallel algorithms for PDE-constrained optimization," in *Parallel Processing for Scientific Computing* (SIAM). pp. 291–322.

Ascher, U. M., and Haber, E., 2001, "Grid refinement and scaling for distributed parameter estimation problems," *Inverse Problems* **17**, 571–590.

Balas, E., 1975, "Facets of the knapsack polytope," *Mathematical Programming* **8**, 146–164.

Balas, E., 1971, "Intersection cuts-a new type of cutting planes for integer programming," *Operations Research* **19**, 19–39.

Balas, E., 1985, "Disjunctive programming and a hierarchy of relaxations for discrete optimization problems," *SIAM Journal on Algebraic Discrete Methods* **6**, 466–486.

Balas, E., 1998, "Disjunctive programming: Properties of the convex hull of feasible points," *Discrete Applied Mathematics* **89**, 3–44.

Balas, E., Ceria, S., and Cornuéjols, G., 1993, "A lift-and-project cutting plane algorithm for mixed 0–1 programs," *Mathematical programming* **58**, 295–324.

Bartlett, R., Heinkenschloss, M., Ridzal, D., and van Bloemen Waanders, B., 2006, "Domain decomposition methods for advection dominated linear-quadratic elliptic optimal control problems," *Computer Methods in Applied Mechanics and Engineering* **195**, 6428–6447.

Baumrucker, B., and Biegler, L., 2009, "MPEC strategies for optimization of a class of hybrid dynamic systems," *Journal of Process Control* **19**, 1248–1256.

Belotti, P., 2009, *COUENNE: a user's manual*, Tech. Rep. (Lehigh University).

Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., and Mahajan, A., 2013, "Mixed integer nonlinear programming," *Acta Numerica* **22**, 1–131.

Bendsøe, M., and Sigmund, O., 2004, *Topological Optimization Theory* (Springer, Berlin).

Berenguel, J. L., Casado, L. G., García, I., Hendrix, E. M., and Messine, F., 2013, "On interval branch-and-bound for additively separable functions with common variables," *Journal of Global Optimization* **56**, 1101–1121.

Berthold, T., 2012 April, *RENS - The optimal rounding*, ZIB-Report 12-17 (Zuse Institut Berlin).

Berthold, T., and Gleixner, A. M., 2012 February, *Undercover: a primal MINLP heuristic exploring a largest sub-MIP*, ZIB-Report 12-07 (Zuse Institut Berlin).

Berthold, T., 2006, *Primal heuristics for mixed integer programs*, Diploma thesis (Technical University Berlin).

Berthold, T., 2014, *Heuristic algorithms in global MINLP solvers*, Ph.D. thesis (Technical University Berlin).

Berthold, T., and Csizmadia, Z., 2020, "The confined primal integral: a measure to benchmark heuristic MINLP solvers against global MINLP solvers," *Mathematical Programming*, 1–15.

Bertsimas, D., and Tsitsiklis, J. N., 1997, *Introduction to Linear Optimization* (Athena Scientific, Belmont, MA).

Bezanson, J., Karpinski, S., Shah, V. B., and Edelman, A., 2012, "Julia: A fast dynamic language for technical computing," *arXiv preprint arXiv:1209.5145.*

Biegler, L., Ghattas, O., Heinkenschloss, M., and van Bloemen Waanders, B., 2001, *Large-Scale PDE-Constrained Optimization, Lecture Notes in Computational Sience and Engineering*, Vol. 30 (Springer-Verlag).

Biegler, L., Ghattas, O., Heinkenschloss, M., Keyes, D., and van Bloemen Waanders, B., 2007, *Real-Time PDE-Constrained Optimization* (SIAM).

Biegler, L. T., Ghattas, O., Heinkenschloss, M., and van Bloemen Waanders, B., 2003, *Large-Scale PDE-Constrained Optimization* (Springer).

Biros, G., and Ghattas, O., 2005a, "Parallel Lagrange-Newton-Krylov-Schur Methods for PDE-Constrained Optimization. Part I: The Krylov-Schur Solver," *SIAM Journal on Scientific Computing* **27.**

Biros, G., and Ghattas, O., 2005b, "Parallel Lagrange-Newton-Krylov-Schur Methods for PDE-Constrained Optimization, Part II: The Lagrange-Newton Solver, and its Application to Optimal Control of Steady Viscous Flows.." *SIAM Journal on Scientific Computing* **27.**

Bisschop, J. J., and Entriken, R., 1993, *AIMMS: The modeling system* (Paragon Decision Technology BV).

Bonami, P., Biegler, L., Conn, A., Cornuéjols, G., Grossmann, I., Laird, C., Lee, J., Lodi, A., Margot, F., Sawaya, N., and Wächter, A., 2008a, "An algorithmic framework for convex mixed integer nonlinear programs," *Discrete Optimization* **5**, 186–204.

Bonami, P., Cornuéjols, G., Lodi, A., and Margot, F., 2009a, "A feasibility pump for mixed integer nonlinear programs," *Mathematical Programming* **119**, 331–352.

Bonami, P., and Gonçalves, J. P. M., 2012, "Heuristics for convex mixed integer nonlinear programs," *Computational Optimization and Applications* **51**, 729–747.

Bonami, P., Biegler, L. T., Conn, A. R., Cornuéjols, G., Grossmann, I. E., Laird, C. D., Lee, J., Lodi, A., Margot, F., Sawaya, N., *et al.*, 2008b, "An algorithmic framework for convex mixed integer nonlinear programs," *Discrete Optimization* **5**, 186–204.

Bonami, P., Cornuéjols, G., Lodi, A., and Margot, F., 2009b, "A feasibility pump for mixed integer nonlinear programs," *Mathematical Programming* **119**, 331–352.

Bonami, P., and Lee, J., 2007, "BONMIN user's manual," *Numer Math* **4**, 1–32.

Bonami, P., Lee, J., Leyffer, S., and Wächter, A., 2011, "More branch-and-bound experiments in convex nonlinear integer programming," *Preprint ANL/MCS-P1949-0911, Argonne National Laboratory, Mathematics and Computer Science Division.*

Bonami, P., and Lejeune, M. A., 2009, "An exact solution approach for portfolio optimization problems under stochastic and integer constraints," *Operations research* **57**, 650–670.

Borzi, A., 2007, "High-order discretization and multigrid solution of elliptic nonlinear constrained optimal control problems," *J. Comp. Applied Math* **200**, 67–85.

Borzì, A., and Schulz, V., 2009, "Multigrid methods for PDE optimization," *SIAM Review* **51**, 361–395.

Brooke, A., Kendrick, D., Meeraus, A., and Raman, R., 1992, *GAMS, A User's Guide*, GAMS Development Corporation.

Bürger, A., Zeile, C., Hahn, M., Altmann-Dieses, A., Sager, S., and Diehl, M., 2020, "pycombina: An open-source tool for solving combinatorial approximation problems arising in mixed-integer optimal control," in *IFAC World Congress 2020.*

Bussieck, M. R., Drud, A. S., and Meeraus, A., 2003, "MINLPLib - a collection of test models for mixed-integer nonlinear programming," *INFORMS Journal on Computing* **15**, 114–119.

Bussieck, M. R., and Pruessner, A., 2003, "Mixed-integer nonlinear programming," *SIAG/OPT Views-and-News* **14**, 19–22.

Calfa, B. A., and Grossmann, I. E., 2015, "Optimal procurement contract selection with price optimization under uncertainty for process networks," *Computers & Chemical Engineering* **82**, 330–343.

Cao, W., and Lim, G. J., 2010, "Optimization models for cancer treatment planning," *Wiley Encyclopedia of Operations Research and Management Science.*

Caprara, A., and Fischetti, M., 1997, "Branch-and-cut algorithms," *Annotated bibliographies in combinatorial optimization*, 45–64.

Castillo, I., Westerlund, J., Emet, S., and Westerlund, T., 2005, "Optimization of block layout design problems with unequal areas: A comparison of MILP and MINLP optimization methods," *Computers & Chemical Engineering* **30**, 54–69.

CBC, 2004, "CBC user's guide," https://coin-or.github.io/Cbc/.

Ceria, S., and Soares, J., 1999, "Convex programming for disjunctive convex optimization," *Mathematical Programming* **86**, 595–614.

Chan, T. F., and Shen, J., 2005, *Image Processing and Analysis* (Society for Industrial and Applied Mathematics).

CLP, 2004, "CLP user's guide," https://coin-or.github.io/Clp/.

Committee, T., 2010, *Advanced Fueld Pellet Materials and Fuel Rod Design for Water Cooled Reactors*, Tech. Rep. (International Atomic Energy Agency).

Conn, A. R., Gould, N. I., and Toint, P. L., 2000, *Trust Region Methods*, Vol. 1 (SIAM, Philadelphia).

CONOPT, 2007, *CONOPT User's Manual*, ARKI Consulting and Development, Denmark.

Costa, M. F. P., Rocha, A. M. A., Francisco, R. B., and Fernandes, E. M., 2016, "Firefly penalty-based algorithm for bound constrained mixed-integer nonlinear programming," *Optimization* **65**, 1085–1104.

CPLEX, 2017, *IBM Ilog CPLEX V12.8: User's Manual for CPLEX*, IBM Corp.

Csurka, G., Larlus, D., Perronnin, F., and Meylan, F., 2013, "What is a good evaluation measure for semantic segmentation?." in *BMVC*, Vol. 27 (Citeseer). p. 2013.

Dahl, H., Meeraus, A., and Zenios, S. A., 1989, *Some financial optimization models: I. risk management* (Fishman-Davidson Center for the Study of the Service Sector, Wharton School).

Dakin, R. J., 1965, "A tree search algorithm for mixed programming problems," *Computer Journal* **8**, 250–255.

Danna, E., Rothberg, E., and LePape, C., 2005a, "Exploring relaxation induced neighborhoods to improve MIP solutions," *Mathematical Programming* **102**, 71–90.

Danna, E., Rothberg, E., and Le Pape, C., 2005b, "Exploring relaxation induced neighborhoods to improve mip solutions," *Mathematical Programming* **102**, 71–90.

Dantzig, G. B., 1998, *Linear programming and extensions*, Vol. 48 (Princeton university press).

Dolan, E., and Moré, J., 2002, "Benchmarking optimization software with performance profiles," *Mathematical Programming* **91**, 201–213.

Donde, V., Lopez, V., Lesieutre, B., Pinar, A., Yang, C., and Meza, J., 2005, "Identification of severe multiple contingencies in electric power networks," in *Proceedings of the 37th Annual North American Power Symposium, 2005.* (IEEE). pp. 59–66.

Donovan, G. H., and Rideout, D. B., 2003, "An integer programming model to optimize resource allocation for wildfire containment," *Forest Science* **49**, 331–335.

Duives, J., Lodi, A., and Malaguti, E., 2013, "Test-assignment: a quadratic coloring problem," *Journal of heuristics* **19**, 549–564.

Dunning, I., Huchette, J., and Lubin, M., 2017, "Jump: A modeling language for mathematical optimization," *SIAM Review* **59**, 295–320.

Duran, M. A., and Grossmann, I. E., 1986, "An outer-approximation algorithm for a class of mixed-integer nonlinear programs," *Mathematical programming* **36**, 307–339.

Ehrgott, M., Waters, C., Kasimbeyli, R., and Ustun, O., 2009, "Multiobjective programming and multiattribute utility functions in portfolio optimization," *INFOR: Information Systems and Operational Research* **47**, 31–42.

Elhedhli, S., 2006, "Service system design with immobile servers, stochastic demand, and congestion," *Manufacturing & Service Operations Management* **8**, 92–97.

FICO-Xpress, 2009, *FICO Xpress Optimization Suite: Xpress-BCL Reference manual*, Fair Isaac Corporation.

Fipki, S., and Celi, A., 2008, "The use of multilateral well designs for improved recovery in heavy oil reservoirs," in *IADV/SPE Conference and Exhibition* (SPE, Orlanda, Florida).

Fischetti, M., Glover, F., and Lodi, A., 2005, "The feasibility pump," *Mathematical Programming* **104**, 91–104.

Fischetti, M., and Lodi, A., 2002, "Local branching," *Mathematical Programming* **98**, 23–47.

Fischetti, M., and Lodi, A., 2008, "Repairing mip infeasibility through local branching," *Computers & operations research* **35**, 1436–1445.

Fletcher, R., and Leyffer, S., 1998, "User manual for filterSQP," university of Dundee Numerical Analysis Report NA-181.

Fletcher, R., and Leyffer, S., 1994, "Solving mixed integer nonlinear programs by outer approximation," *Mathematical programming* **66**, 327–349.

Floudas, C. A., 1995, *Nonlinear and mixed-integer optimization: fundamentals and applications* (Oxford University Press).

Floudas, C. A., and Pardalos, P. M., 2013, *State of the art in global optimization: computational methods and applications*, Vol. 7 (Springer Science & Business Media).

Floudas, C. A., Pardalos, P. M., Adjiman, C. S., Esposito, W. R., Gümüs, Z. H., Harding, S. T., Klepeis, J. L., Meyer, C. A., and Schweiger, C. A., 1999, *Handbook of test problems in local and global optimization* (Kluwer Academic Publishers).

Forsgren, A., Gill, P. E., and Wong, E., 2015, "Active-set methods for convex quadratic programming," *arXiv preprint ArXiv:1503.08349.*

Forsgren, A., Gill, P. E., and Wright, M. H., 2002, "Interior methods for nonlinear optimization," *SIAM review* **44**, 525–597.

Fourer, R., Gay, D. M., and Kernighan, B. W., 1993, *AMPL: A Modeling Language for Mathematical Programming* (The Scientific Press).

Frangioni, A., and Gentile, C., 2006, "Perspective cuts for a class of convex 0–1 mixed integer programs," *Mathematical Programming* **106**, 225–236.

Frangioni, A., and Gentile, C., 2009, "A computational comparison of reformulations of the perspective relaxation: SOCP vs. cutting planes," *Operations Research Letters* **37**, 206–210.

Frangioni, A., Gentile, C., and Lacalandra, F., 2008, "Solving unit commitment problems with general ramp constraints," *International Journal of Electrical Power & Energy Systems* **30**, 316–326.

Friedlander, M. P., and Leyffer, S., 2008, "Global and finite termination of a two-phase augmented lagrangian filter method for general quadratic programs," *SIAM Journal on Scientific Computing* **30**, 1706–1729.

Fügenschuh, A., Geißler, B., Martin, A., and Morsi, A., 2009, "The transport PDE and mixed-integer linear programming," in *Dagstuhl Seminar Proceedings* (Schloss Dagstuhl-Leibniz-Zentrum für Informatik).

Furman, K., 2009, "An exact MINLP formulation for nonlinear disjuctive programs based on the convex hull," in *Presentation at 20th International Symposium on Mathematical Programming*.

Furman, K. C., Sawaya, N. W., and Grossmann, I. E., 2020, "A computationally useful algebraic representation of nonlinear disjunctive convex sets using the perspective function," *Computational Optimization and Applications*, 1–26.

Garmatter, D., Porcelli, M., Rinaldi, F., and Stoll, M., 2019, "Improved penalty algorithm for mixed integer PDE constrained optimization (MIPDECO) problems," *arXiv preprint arXiv:1907.06462*.

Geoffrion, A. M., 1972, "Generalized Benders Decomposition," *Journal of optimization theory and applications* **10**, 237–260.

Gerdts, M., and Sager, S., 2012, "Mixed-integer DAE optimal control problems: Necessary conditions and bounds," in *Control and Optimization with Differential-Algebraic Constraints*, edited by Biegler, L., Campbell, S., and Mehrmann, V. (SIAM). pp. 189–212.

Gill, P. E., and Wong, E., 2012, "Sequential quadratic programming methods," in *Mixed integer nonlinear programming* (Springer). pp. 147–224.

Gill, P. E., and Wong, E., 2015, "Methods for convex and general quadratic programming," *Mathematical programming computation* **7**, 71–112.

Golub, G. H., Heath, M., and Wahba, G., 1979, "Generalized cross-validation as a method for choosing a good ridge parameter," *Technometrics* **21**, 215–223.

Gomory, R. E., 1960, *An Algorithm for the Mixed Integer Problem*, Tech. Rep. RM-2597 (The RAND Corporation).

Gomory, R. E., 1963, "An algorithm for integer solutions to linear programs," *Recent advances in mathematical programming* **64**, 14.

Griewank, A., and Toint, P., 1982, "On the unconstrained optimization of partially separable functions," in *Nonlinear Optimization 1981* (Academic press). pp. 301–312.

Grossmann, I., and Lee, S., 2003, "Generalized convex disjunctive programming: Nonlinear convex hull relaxation," *Computational Optimization and Applications*, 83–100.

Grossmann, I. E., Viswanathan, J., Vecchietti, A., Raman, R., Kalvelagen, E., *et al.*, 2002, "Gams/dicopt: A discrete continuous optimization package," *GAMS Corporation Inc* **37**, 55.

Gu, Z., Nemhauser, G. L., and Savelsbergh, M. W., 1999, "Lifted flow cover inequalities for mixed 0-1 integer programs," *Mathematical Programming* **85**, 439–467.

Gunasekaran, A., Goyal, S., Martikainen, T., and Yli-Olli, P., 1993, "Equipment selection problems in just-in-time manufacturing systems," *Journal of the Operational Research Society* **44**, 345–353.

Günlük, O., Lee, J., and Weismantel, R., 2007, "MINLP strengthening for separable convex quadratic transportation-cost UFL," *IBM Res. Report*, 1–16.

Günlük, O., and Linderoth, J., 2008, "Perspective relaxation of mixed integer nonlinear programs with indicator variables," in *International Conference on Integer Programming and Combinatorial Optimization* (Springer). pp. 1–16.

Günlük, O., and Linderoth, J., 2010, "Perspective reformulations of mixed integer nonlinear programs with indicator variables," *Mathematical programming* **124**, 183–205.

Guo, J.-y., Lu, W.-x., Yang, Q.-c., and Miao, T.-s., 2019, "The application of 0–1 mixed integer nonlinear programming optimization model based on a surrogate model to identify the groundwater pollution source," *Journal of contaminant hydrology* **220**, 18–25.

Gupta, O. K., and Ravindran, A., 1985, "Branch and bound experiments in convex nonlinear integer programming," *Management science* **31**, 1533–1546.

Gurobi, 2012, *Gurobi Optimizer Reference Manual, Version 5.0*, Gurobi Optimization, Inc.

Gutierrez, R. A., and Sahinidis, N., 1996, "A branch-and-bound approach for machine selection in just-in-time manufacturing systems," *International journal of production research* **34**, 797–818.

Haber, E., and Ascher, U. M., 2001, "Preconditioned all-at-once methods for large, sparse parameter estimation problems," *Inverse Problems* **17**, 1847–1864.

Haber, E., and Oldenburg, D., 2000, "A GCV based method for nonlinear ill-posed problems," *Computational Geosciences* **4**, 41–63.

Hahn, M., Sager, S., and Leyffer, S., 2020, "Binary optimal control by trust-region steepest descent," submitted for publication.

Hahn, M., Leyffer, S., and Zavala, V. M., 2017, *Mixed-Integer PDE-Constrained Optimal Control of Gas Networks*, Tech. Rep. Preprint ANL/MCS-P7095-0817 (Mathematics and Computer Science Division, Argonne National Laboratory).

Hansen, P. C., 1998, *Rank-deficient and discrete ill-posed problems*, SIAM Monographs on Mathematical Modeling and Computation (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA). ISBN 0-89871-403-6

Hante, F. M., 2017, "Relaxation methods for hyperbolic PDE mixed-integer optimal control problems," *Optimal Control Applications and Methods* **38**, 1103–1110.

Hante, F. M., and Sager, S., 2013, "Relaxation methods for mixed-integer optimal control of partial differential equations," *Computational Optimization and Applications* **55**, 197–225.

Harjunkoski, I., Westerlund, T., Pörn, R., and Skrifvars, H., 1998, "Different transformations for solving non-convex trim-loss problems by MINLP," *European Journal of Operational Research* **105**, 594–603.

Hart, W. E., Watson, J.-P., and Woodruff, D. L., 2011, "Pyomo: modeling and solving mathematical programs in Python," *Mathematical Programming Computations* **3**, 219–260.

Hazra, S. B., and Schulz, V., 2006, "Simultaneous pseudo-timestepping for aerodynamic shape optimization problems with state constraints," *SIAM J. Sci. Comput.* **28**, 1078–1099.

He, H., Daume III, H., and Eisner, J. M., 2014, "Learning to search in branch and bound algorithms," in *Advances in neural information processing systems*, pp. 3293–3301.

Heinkenschloss, M., and Ridzal, D., 2008, "Lecture notes in computational science and engineering," Chap. Integration of Sequential Quadratic Programming and Domain Decomposition Methods for Nonlinear Optimal Control Problems (Springer-Verlag).

Hijazi, H., Bonami, P., and Ouorou, A., 2014, "An outer-inner approximation for separable mixed-integer nonlinear programs," *INFORMS Journal on Computing* **26**, 31–44.

Hintermuller, M., and Vicente, L. N., 2005, "Space mapping for optimal control of partial differential equations," *SIAM J. Opt.* **15**, 1002–1025.

Hinze, M., Pinnau, R., Ulbrich, M., and Ulbrich, S., 2009, *Optimization with PDE Constraints* (Springer).

Horowitz, E., and Sahni, S., 1974, "Computing partitions with applications to the knapsack problem," *Journal of ACM* **21**, 277–292.

Hutter, F., Hoos, H. H., and Leyton-Brown, K., 2010, "Automated configuration of mixed integer programming solvers," in *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming* (Springer). pp. 186–202.

Ibaraki, T., 1976, "Theoretical comparisons of search strategies in branch-and-bound algorithms," *International Journal of Computer & Information Sciences* **5**, 315–344.

IPOPT, 2015, "IPOPT user's manual," https://projects.coin-or.org/Ipopt/browser/stable/3.11/Ipopt.

Iri, M., 1984, "Simultaneous computation of functions, partial derivatives and estimates of rounding errors-complexity and practicality," *Japan Journal of Industrial and Applied Mathematics* **1**, 223–252.

Iribarren, O. A., Montagna, J. M., Vecchietti, A. R., Andrews, B., Asenjo, J. A., and Pinto, J. M., 2004, "Optimal process synthesis for the production of multiple recombinant proteins," *Biotechnology progress* **20**, 1032–1043.

Jablonskỳ, J., *et al.*, 2015, "Benchmarks for current linear and mixed integer optimization solvers," *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis* **63**, 1923–1928.

Jeroslow, R. G., 1973, "There cannot be any algorithm for integer programming with quadratic constraints," *Operations Research* **21**, 221–224.

Jobst, N. J., Horniman, M. D., Lucas, C. A., Mitra, G., *et al.*, 2001, "Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints," *Quantitative finance* **1**, 489–501.

Jung, M., 2013, *Relaxations and Approximations for Mixed-Integer Optimal Control*, Ph.D. thesis (University Heidelberg).

Jung, M., Reinelt, G., and Sager, S., 2015, "The Lagrangian Relaxation for the Combinatorial Integral Approximation Problem," *Optimization Methods and Software* **30**, 54–80.

Junior, H. V., and Lins, M. P. E., 2005, "An improved initial basis for the simplex algorithm," *Computers & Operations Research* **32**, 1983–1993.

Kannan, R., and Monma, C., 1978a, "On the computational complexity of integer programming problems," in *Optimization and Operations Research,*, Lecture Notes in Economics and Mathematical Systems, Vol. 157, edited by Henn, R., Korte, B., and Oettli, W. (Springer). pp. 161–172.

Kannan, R., and Monma, C. L., 1978b, "On the computational complexity of integer programming problems," in *Optimization and Operations Research* (Springer). pp. 161–172.

Karmarkar, N., 1984, "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pp. 302–311.

Kaur, S., Kumbhar, G., and Sharma, J., 2014, "A MINLP technique for optimal placement of multiple dg units in distribution systems," *International Journal of Electrical Power & Energy Systems* **63**, 609–617.

Kendrick, D., 1982, "Caution and probing in a macroeconomic model," *Journal of Economic Dynamics and Control* **4**, 149–170.

Khalil, E. B., Dilkina, B., Nemhauser, G. L., Ahmed, S., and Shao, Y., 2017, "Learning to run heuristics in tree search.." in *IJCAI*, pp. 659–666.

Khalil, E. B., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B., 2016, "Learning to branch in mixed integer programming," in *Thirtieth AAAI Conference on Artificial Intelligence.*

Kılınç, M. R., and Sahinidis, N. V., 2018, "Exploiting integrality in the global optimization of mixed-integer nonlinear programming problems with BARON," *Optimization Methods and Software* **33**, 540–562.

Kilinç, M. R., 2011, *Disjunctive cutting planes and algorithms for convex mixed integer nonlinear programming*, Ph.D. thesis (University of Wisconsin-Madison).

Klee, V., Minty, G. J., and Shisha, O., 1972, "Inequalities, iii," *How Good is the Simplex Algorithm (1972)*, 159–175.

KNITRO, 2012 dec., *KNITRO Documentation*, Ziena Optimization.

Koberstein, A., 2005, "The dual simplex method, techniques for a fast and stable implementation," *Unpublished doctoral thesis, Universität Paderborn, Paderborn, Germany.*

Kostina, E., 2002, "The long step rule in the bounded-variable dual simplex method: numerical experiments," *Mathematical Methods of Operations Research* **55**, 413–429.

Kronqvist, J., Bernal, D. E., Lundell, A., and Grossmann, I. E., 2018a, "A review and comparison of solvers for convex MINLP," *Optimization and Engineering*, 1–59.

Kronqvist, J., Lundell, A., and Westerlund, T., 2016, "The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming," *Journal of Global Optimization* **64**, 249–272.

Kronqvist, J., Lundell, A., and Westerlund, T., 2018b, "Reformulations for utilizing separability when solving convex MINLP problems," *Journal of Global Optimization* **71**, 571–592.

Laird, C. D., Biegler, L. T., van Bloemen Waanders, B., and Bartlett, R. A., 2005, "Time dependent contaminant source determination for municipal water networks using large scale optimization," *ASCE J. Water Res. Mgt. Plan.*, 125–134.

Laird, C. D., Biegler, L. T., and van Bloemen Waanders, B. G., 2006, "Mixed-integer approach for obtaining unique solutions in source inversion of water networks," *Journal of Water Resources Planning and Management* **132**, 242–251.

Land, A. H., and Doig, A. G., 2010, "An automatic method for solving discrete programming problems," in *50 Years of Integer Programming 1958-2008* (Springer). pp. 105–132.

Lasdon, L. S., Fox, R. L., and Ratner, M. W., 1974, "Nonlinear optimization using the generalized reduced gradient method," *Revue française d'automatique, informatique, recherche opérationnelle. Recherche opérationnelle* **8**, 73–103.

Lee, M., Ma, N., Yu, G., and Dai, H., 2020, "Accelerating generalized benders decomposition for wireless resource allocation," *arXiv preprint arXiv:2003.01294*.

Legg, M., Davidson, R. A., and Nozick, L. K., 2013, "Optimization-based regional hurricane mitigation planning," *Journal of infrastructure systems* **19**, 1–11.

Lemke, C. E., 1954, "The dual method of solving the linear programming problem," *Naval Research Logistics Quarterly* **1**, 36–47.

Leyffer, S., and Mahajan, A., 2010, "Software for nonlinearly constrained optimization," *Wiley Encyclopedia of Operations Research and Management Science*.

Leyffer, S., Munson, T., Wild, S., van Bloemen Waanders, B., and Ridzal, D., 2013 August, "Mixed-integer PDE-constrained optimization," Position Paper #15 submitted in response to the ExaMath13 Call for Position Papers, https://collab.mcs.anl.gov/download/attachments/7569466/examath13_submission_15.pdf.

Linderoth, J. T., and Savelsbergh, M. W., 1999, "A computational study of search strategies for mixed integer programming," *INFORMS Journal on Computing* **11**, 173–187.

Lodi, A., 2010, "Mixed integer programming computation," in *50 Years of Integer Programming 1958-2008* (Springer). pp. 619–645.

Lubin, M., Yamangil, E., Bent, R., and Vielma, J. P., 2016, "Extended formulations in mixed-integer convex programming," in *International Conference on Integer Programming and Combinatorial Optimization* (Springer). pp. 102–113.

Lucidi, S., and Rinaldi, F., 2013, "An exact penalty global optimization approach for mixed-integer programming problems," *Optimization Letters* **7**, 297–307.

Lundell, A., Kronqvist, J., and Westerlund, T., 2018, "The supporting hyperplane optimization toolkit," *http://www.github.com/coin-or/shot*.

Lundell, A., and Westerlund, T., 2018, "Solving global optimization problems using reformulations and signomial transformations," *Computers & Chemical Engineering* **116**, 122–134.

Mahajan, A., 2010, "Presolving mixed–integer linear programs," *Wiley Encyclopedia of Operations Research and Management Science.*

Mahajan, A., Leyffer, S., Linderoth, J., Luedtke, J., and Munson, T., 2020, "Minotaur: A mixed-integer nonlinear optimization toolkit," *Mathematical Programming Computation*, 1–38.

Manne, A. S., 1986, "GAMS/MINOS: three examples," *Draft (Stanford, CA: Stanford University, Department of Operations Research, December).*

Manns, P., and Kirches, C., 2018 July, "Multi-dimensional sum-up rounding for elliptic control systems," *DFG SPP 1962 Preprint.*

Manns, P., and Kirches, C., 2019, "Multi-dimensional sum-up rounding using Hilbert curve iterates," *PAMM* **19**, e201900065.

Manns, P., and Kirches, C., 2020, "Improved regularity assumptions for partial outer convexification of mixed-integer PDE-constrained optimization problems," *ESAIM: Control, Optimisation and Calculus of Variations* **26**, 32.

Maros, I., 2003, "A generalized dual phase-2 simplex algorithm," *European Journal of Operational Research* **149**, 1–16.

Martello, S., and Toth, P., 1990, *Knapsack Problems: Algorithms and Computer Implementations* (John Wiley & Sons Ltd, Chichester, West Sussex, England).

Martello, S., Pisinger, D., and Toth, P., 1999, "Dynamic programming and strong bounds for the 0-1 knapsack problem," *Management Science* **45**, 414–424.

Meller, R. D., Narayanan, V., and Vance, P. H., 1998, "Optimal facility layout design," *Operations Research Letters* **23**, 117–127.

Mikolajková, M., Saxén, H., and Pettersson, F., 2018, "Linearization of an MINLP model and its application to gas distribution optimization," *Energy* **146**, 156–168.

Misener, R., and Floudas, C. A., 2014, "ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations," *Journal of Global Optimization* **59**, 503–526.

Mitchell, J. E., 2010, "Branch and cut," *Wiley encyclopedia of operations research and management science.*

Mittelmann, H., 2019, "Decision tree for optimization software," http://plato.asu.edu/bench.html.

Mittelmann, H. D., 2017, "Latest benchmarks of optimization software," in *INFORMS Annual Meeting. Houston, TX.*

MOSEK, 2004, "Mosek ApS," www.mosek.com.

Mullin, T., and Belotti, P., 2016, "Using branch-and-bound algorithms to optimize selection of a fixed-size breeding population under a relatedness constraint," *Tree genetics & genomes* **12**, 4.

Murty, K. G., and Kabadi, S. N., 1985, *Some NP-complete problems in quadratic and nonlinear programming*, Tech. Rep.

Nannicini, G., Belotti, P., and Liberti, L., 2008, "A local branching heuristic for MINLPs," *arXiv preprint arXiv:0812.2188.*

Nesterov, Y., and Nemirovskii, A., 1994, *Interior-point polynomial algorithms in convex programming* (SIAM).

Nocedal, J., and Wright, S., 2000, *Numerical Optimization* (Springer-Verlag, New York).

Nowak, I., Breitfeld, N., Hendrix, E. M., and Njacheun-Njanzoua, G., 2018, "Decomposition-based inner-and outer-refinement algorithms for global optimization," *Journal of Global Optimization* **72**, 305–321.

Ozdogan, U., 2004, *Optimization of Well Placement Under Time-Dependent Uncertainty*, Master's thesis (Stanford University).

Pisinger, D., and Toth, P., 1998, "Knapsack problems," in *Handbook of Combinatorial Optimization*, edited by Du, D. Z. and Pardalos, P. (Kluwer). pp. 1–89.

Pourakbari-Kasmaei, M., Lehtonen, M., Fotuhi-Firuzabad, M., Marzband, M., and Mantovani, J. R. S., 2019, "Optimal power flow problem considering multiple-fuel options and disjoint operating zones: A solver-friendly MINLP model," *International Journal of Electrical Power & Energy Systems* **113**, 45–55.

Quesada, I., and Grossmann, I. E., 1992, "An LP/NLP based branch and bound algorithm for convex MINLP optimization problems," *Computers & chemical engineering* **16**, 937–947.

Ralphs, T., and Ladányi, L., 2000, "SYMPHONY : A parallel framework for branch, cut, and price," available from ftp://ftp.branchandcut.org/pub/reference/symphony.ps.

Ravemark, D. E., and Rippin, D. W., 1998, "Optimal design of a multi-product batch plant," *Computers & Chemical Engineering* **22**, 177–183.

Rockafellar, R., 1970, *Convex Analysis* (Princeton University Press, Princeton, NJ).

Rudin, L. I., Osher, S., and Fatemi, E., 1992 Nov., "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena* **60**, 259–268.

Ruthotto, L., Treister, E., and Haber, E., 2017, "jInv–a flexible julia package for pde parameter estimation," *SIAM Journal on Scientific Computing* **39**, S702–S722.

Ryoo, H. S., and Sahinidis, N. V., 1996, "A branch-and-reduce approach to global optimization," *Journal of Global Optimization* **8**, 107–139.

Sager, S., 2006, *Numerical methods for mixed–integer optimal control problems*, Ph.D. thesis (Universität Heidelberg).

Sager, S., 2009, "Reformulations and Algorithms for the Optimization of Switching Decisions in Nonlinear Optimal Control," *Journal of Process Control* **19**, 1238–1247.

Sager, S., Bock, H., and Diehl, M., 2012, "The Integer Approximation Error in Mixed-Integer Optimal Control," *Mathematical Programming A* **133**, 1–23.

Sager, S., Jung, M., and Kirches, C., 2011, "Combinatorial integral approximation," *Mathematical Methods of Operations Research* **73**, 363–380.

Sahinidis, N. V., 1996, "BARON: A general purpose global optimization software package," *Journal of global optimization* **8**, 201–205.

Savelsbergh, M. W., 1994, "Preprocessing and probing techniques for mixed integer programming problems," *ORSA Journal on Computing* **6**, 445–454.

Sawaya, N., and Grossmann, I. E., 2008, *Reformulations, relaxations and cutting planes for linear generalized disjunctive programming* (Citeseer).

Scherzer, O and Grasmair, M and Grossauer, H and Haltmeier, M and Lenzen, F, 2013, *Variational Methods in Imaging* (Springer, Berlin).

Schlueter, M., Erb, S. O., Gerdts, M., Kemble, S., and Rückmann, J.-J., 2013, "MIDACO on MINLP space applications," *Advances in Space Research* **51**, 1116–1131.

Sharma, M., Hahn, M., Leyffer, S., Ruthotto, L., and van Bloemen Waanders, B., 2020a, "Inversion of convection–diffusion equation with discrete sources," *Optimization and Engineering*, 1–39.

Sharma, M., Palkar, P., and Mahajan, A., 2020b, "Linearization and parallelization schemes for convex mixed-integer nonlinear optimization," *Optimization Online* **7793.**

Sharma, S., 2013, *Mixed-integer nonlinear programming heuristics applied to a shale gas production optimization problem*, Master's thesis (Institutt for teknisk kybernetikk).

Shectman, J. P., and Sahinidis, N. V., 1998, "A finite algorithm for global minimization of separable concave programs," *Journal of Global Optimization* **12**, 1–36.

Sigmund, O., and Maute, K., 2013a, "Topological optimization approaches," *Structural Multidisciplinary Optimization* **48**, 1031–1055.

Sigmund, O., and Maute, K., 2013b, "Topology optimization approaches: A comparative review," *Structural and Multidisciplinary Optimization* **48**, 1031–1055.

Silver, E., and Moon, I., 1999, "A fast heuristic for minimising total average cycle stock subject to practical constraints," *Journal of the Operational Research Society* **50**, 789–796.

Simon, R., 2008, *Multigrid Solver for Saddle Point Problems in PDE-Constrained Optimization*, Ph.D. thesis (Johannes Kepler Universitat Linz).

SNOPT, 2008, *SNOPT User's Manual*, University of California, San Diego, La Jolla, CA 92093-0112, USA.

de Souza, M., and Ritt, M., 2018, "An automatically designed recombination heuristic for the test-assignment problem," in *2018 IEEE Congress on Evolutionary Computation (CEC)* (IEEE). pp. 1–8.

Su, L., Tang, L., and Grossmann, I. E., 2015, "Computational strategies for improved MINLP algorithms," *Computers & Chemical Engineering* **75**, 40–48.

Tang, Y., Agrawal, S., and Faenza, Y., 2019, "Reinforcement learning for integer programming: Learning to cut," *arXiv preprint arXiv:1906.04859*

Tawarmalani, M., and Sahinidis, N. V., 2005, "A polyhedral branch-and-cut approach to global optimization," *Mathematical Programming* **103**, 225–249.

Türkay, M., and Grossmann, I. E., 1996, "Logic-based MINLP algorithms for the optimal synthesis of process networks," *Computers & Chemical Engineering* **20**, 959–978.

Vavasis, S. A., 1991, *Nonlinear optimization: complexity issues* (Oxford University Press, Inc.).

Vecchietti, A., and Grossmann, I. E., 1999, "Logmip: a disjunctive 0–1 non-linear optimizer for process system models," *Computers & chemical engineering* **23**, 555–565.

Vielma, J. P., Ahmed, S., and Nemhauser, G. L., 2008, "A lifted linear programming branch-and-bound algorithm for mixed-integer conic quadratic programs," *INFORMS Journal on Computing* **20**, 438–450.

Vogel, C. R., 1999, "Sparse matrix computations arising in distributed parameter identification," *SIAM J. Matrix Anal. Appl.*, 1027–1037.

Vogel, C. R., 2002 Jan., *Computational Methods for Inverse Problems* (Society for Industrial and Applied Mathematics). ISBN 978-0-89871-550-7

Wächter, A., Biegler, L., Lang, Y., and Raghunathan, A., 2002, "IPOPT: An interior point algorithm for large-scale nonlinear optimization," http://www.coin-or.org.

Westerlund, T., and Lundqvist, K., 2001, *Alpha-ECP, Version 5.01: An Interactive MINLP-Solver Based on the Extended Cutting Plane Method*, Technical Report Report 01-178-A (Process Design Laboratory at Åbo University).

Westerlund, T., and Pettersson, F., 1995, "An extended cutting plane method for solving convex MINLP problems," *Computers & Chemical Engineering* **19**, 131–136.

Witzig, J., Berthold, T., and Heinz, S., 2017, "Experiments with conflict analysis in mixed integer programming," in *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (Springer). pp. 211–220.

Wright, S. J., Nowak, R. D., and Figueiredo, M. A., 2009, "Sparse reconstruction by separable approximation," *IEEE Transactions on Signal Processing* **57**, 2479–2493.

Yamamura, K., and Kiyoi, M., 1992, "Algorithms for detecting separability of nonlinear mappings using computational graphs," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)* **75**, 93–105.

You, F., and Leyffer, S., 2010, "Oil spill response planning with MINLP," *SIAG/OPT Views-and-News* **21**, 1–8.

You, F., and Leyffer, S., 2011, "Mixed-integer dynamic optimization for oil-spill response planning with integration of a dynamic oil weathering model," *AIChe Journal.*

Zamora, J. M., and Grossmann, I. E., 1998, "A global MINLP optimization algorithm for the synthesis of heat exchanger networks with no stream splits," *Computers & Chemical Engineering* **22**, 367–384.

Zondervan, E., and Grossmann, I. E., 2009, "A deterministic security constrained unit commitment model," http://www.minlp.org.

List of Publications

- Sharma M., Hahn M., Leyffer S., Ruthotto L., and Van Bloemen Waanders, B., 2020, "Inversion of convection-diffusion equation with discrete sources," *Optimization and Engineering*, 1-39.

- Sharma M., Palkar P., and Mahajan A., "Linearization and parallelization schemes for convex mixed-integer nonlinear optimization," *Under review, Optimization Online,* http://www.optimization-online.org/DB_FILE/2020/05/7793.pdf

- Sharma M., Mahajan A., "Automatic reformulation techniques using problem structures for convex mixed-integer nonlinear programs," *To be communicated.*

- Inversion of convection-diffusion equation with discrete sources. In *6th International Conference on Continuous Optimization (ICCOPT) 2019, Berlin, Germany*, August 03-08, 2019. (Invited Talk)

- Practical enhancements for the LP/NLP algorithm for convex MINLPs. *3rd International Conference and Summer School on Numerical Computations: Theory and Algorithms (NUMTA) 2019, Calabria, Italy*, June 15-21, 2019.

- Inversion of convection-diffusion PDE with discrete source: Mixed integer PDE constrained optimization. *23rd International Symposium on Mathematical Programming (ISMP), Bordeaux, France*, July 01-06, 2018. (Invited Talk)

- Automatic reformulation techniques for mixed-integer nonlinear programs. *Mixed Integer Programming (MIP) Workshop 2018, Greenville, SC, USA*, June 18-21, 2018.

- MINOTAUR: Mixed integer nonlinear optimization toolkit - algorithms, underestimators, relaxations. *OPTSUM 2017, Mumbai, India*, September 14, 2017.

- Exploiting structures in nonlinear constraints for reformulation of convex mixed-integer nonlinear programs. *ORSI-2016, New Delhi, India*, December 12-14, 2016.

- Automatic reformulation of convex mixed-integer nonlinear programs. *SPJIMR-POMS India Chapter Conference on Big Data Analytics for Optimising Supply Chains, Mumbai*, July 29-30, 2016.

- MILP formulations for a ship routing and scheduling problem in port operations of a refinery. *XIX Annual International Conference of the Society of Operations Management*, December 11-13, 2015.

Acknowledgements

Firstly, I would like to thank Prof. Ashutosh Mahajan for supervising this thesis, providing valuable suggestions, and timely encouragement. I also thank my research progress committee members for their time and advice, and my special thanks to Prof. N. Hemachandra for some insightful discussions.

I am grateful to Dr. Sven Leyffer, Prof. Lars Ruthotto, and Dr. Bart van Bloemen Waanders for their guidance and support during our collaborative work and always.

I also wish to thank faculty, staff, fellow students at Industrial Engineering and Operations Research for their help and great company.

Finally, I thank my husband, family, and friends for always being there.

Meenarli Sharma
IIT Bombay
July 8, 2021