Quantified Linear Programming

Quantifizierte Lineare Programmierung

Vom Fachbereich Maschinenbau
an der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

genehmigte

DISSERTATION

vorgelegt von

Dipl.-Inf. Jan Wolf

aus Minden

Berichterstatter:	Prof. Dr.-Ing. Peter F. Pelz
1. Mitberichterstatter:	Prof. Dr. rer. nat. Alexander Martin
2. Mitberichterstatter:	Prof. Dr. rer. nat. Ulf Lorenz
Tag der Einreichung:	01.07.2014
Tag der mündlichen Prüfung:	18.11.2014

Darmstadt 2014

D 17

Vorwort des Herausgebers

Kontext
Ingenieure haben die Aufgabe Systeme (Maschinen, Apperate, Anlagen) so zu gestalten, dass eine Funktion nachhaltig und zuverlässig erfüllt wird. Die Funktion wird durch Verben wie fördern, tragen, ... beschrieben. Die Nachhaltigkeit wird durch Adverbien und Zusätze wie „mit wenig Material und geringer Unsicherheit tragen“ gefordert. Methodisch setzt sich der von der DFG geförderte Sonderforschungsbereich SFB805 „Beherrschung von Unsicherheit in lasttragenden Systemen des Maschinenbaus“ genau mit dieser Kernaufgabe des Ingenieurwesens auseinander. Die vorliegende Dissertation von Herrn Dipl.-Inf. Jan Wolf ist dem erweiterten SFB-Kreis zuzuordnen.

Um die Aufgabe „Funktion nachhaltig und zuverlässig erfüllen“zu lösen, kombiniert der Ingenieur entweder bekannte Teilfunktionen zu einer Funktionsstruktur und letztlich zu einer Komponentenstruktur oder aber er sieht Bedarf für neue Komponenten, die die Aufgabe in besserer oder günstigerer, d.h. nachhaltigerer, Weise erfüllen. Ersteres, d.h. die Kombination bestehender Komponenten, ist ein *planerisches Tun*. Letzteres, d.h. das Erschaffen neuer Komponenten ist ein *kreatives Tun*.

„Computer sind nicht kreativ“, so Gunter Dueck, was man an dem langweiligen Fussballspiel von Robotern sieht: den Robotern fehlen Emotionen, die für Imagination Voraussetzung sind. Dem gegenüber können Algorithmen und Computer uns sehr gut bei planerischen Aufgaben unterstützen. Dies gelingt auch in komplexen Situationen, die schwer einzuschätzen sind, was eindrucksvoll durch die Erfolge der Schachcomputer „Deep Blue“ gegen den Schachspieler Garri Kasparow im Jahr 1996, sowie „Hydra“ gegen Michael Adams, belegt ist. Heute ist allgemein anerkannt: Computer sind die besseren Schachspieler. Weder ermüden sie, noch kopieren sie menschliche Spielweisen.

Daher stellt sich zwangsläufig die Frage: Sind Computer die besseren Planungsingenieure? Die Antwort muss heute bereits lauten: Ja! Der Mensch bewertet die Nachhaltigkeit (Gewicht, Energieverbrauch, ...) und Unsicherheit dem Strukturierungsprozess nachgelagert. Nur sehr selten wird die Struktur wieder in Frage gestellt. Die heute im Ingenieurwesen durchgeführten Optimierungen zielen daher zumeist allein auf eine Komponentenoptimierung und eine Optimierung der Betriebsstrategie (z.B. in der Regelungstechnik). Dem liegt die falsche Hypothese zugrunde, dass ein System aus guten Komponenten ebenfalls gut ist. Ziel muss es sein bereits im Strukturierungsprozess, d.h. bei der diskreten Entscheidung für oder gegen eine Komponente aus dem Komponentenbaukasten bzw. Spielfeld, nicht nur die Funktion sondern auch zeitgleich Nachhaltigkeit und Unsicherheit zu bewerten. Letztere können Zielfunktionale oder Restriktionen sein. Die Funktion wird immer

Nebenbedingung sein, die zwangslufig erfüllt sein muss, selbst wenn eine Lasthistorie unsicherheitsbehaftet ist.

Die wissenschaftliche Arbeit von Herrn Wolf setzt sich methodisch und technologisch mit algorithmischen (quantifizierten) Entscheidungen unter Unsicherheit auseinander, die gerade im technischen Kontext bei der Systemsynthese nach meiner Einschätzung enorm an Bedeutung gewinnen werden.

Forschungsfrage
Herr Wolf hat als Kern seiner Forschung quantifizierte lineare Programme (QLPs), die erst seit ca. 10 Jahren im Forschungsfokus stehen. Der Begriff QLP wurde im Jahr 2003 erstmals genannt. Herr Wolf stellt die Frage nach den theoretischen Eigenschaften von QLPs und nach Algorithmen, um QLPs in der Praxis zu lösen.

Methoden und Ergebnisse
Da QLPs vergleichsweise wenig erforscht sind, führt Herr Wolf sehr genau in das Thema ein. Überaus wertvoll sind die Anwendungsbeispiele in Kapitel 2, die die Mächtigkeit der Methode gerade im Kontext der Forschungsfragen des erwähnten Sonderforschungbereiches zeigt. Bei der im Beispiel behandelten Strukturierung einer Druckerhöhungsanlage müssen Entscheidungen getroffen werden, so dass die Lebenszeitkosten, hier Investitionskosten plus Betriebskosten, minimal sind und die Funktion bei beliebigen Lasten zwischen den Nennlasten noch erfüllt ist.

Herr Wolf zeigt, dass QLPs auch spieltheoretisch als Zwei-Peronen-Spiele zwischen einem Entscheidungen treffenden Existenzspieler und einem Allspieler interpretiert werden können. Durch den Allspieler wird die Unsicherheit abgebildet, da seine Entscheidungen nicht vorhersagbar sind. Der Existenzspieler ist der „algorithmische Planungsingenieur“. Im Beispiel der Druckerhöhungsanlage wird durch den Allspieler die Unsicherheit in der Druckanforderung abgebildet. Durch die Interpretation als Zwei-Personen-Spiel können Entscheidungsbäume in Kombination mit Polytopen genutzt werden, die die Methoden sehr anschaulich machen (vgl. Fig. 2.1 und 2.2.).

Bei dem technischen Beispiel wird auch deutlich, wie physikalisch-technische Zusammenhänge wie Massen- und Energiebilanz, sowie phänomenologische Kennfelder für alle möglichen Systeme (nicht nur eine einzelne Struktur) in einem (quantifizierten) gemischtganzzahligen linearen Programm abgebildet werden können. Damit liefert Herr Wolf Pionierarbeit für das sicherlich wachsende Gebiet des Technical Operations Research (TOR). Herr Wolf liefert einen wichtigen Beitrag für die Grundlagen von QLPs. Er geht aber darüber hinaus, indem er den Solver `QlpOpt` für QLPs entwickelt, indem seine Grundlagenerkenntisse angewendet werden.

Peter Pelz
Darmstadt, den 18.11.2014

Danksagung

An dieser Stelle möchte ich die Gelegenheit nutzen, um allen Personen meinen Dank zum Ausdruck zu bringen, die zum gelingen dieser Arbeit beigetragen haben.

Ganz besonders danke ich Herrn Prof. Ulf Lorenz, der diese Arbeit betreut hat und mir stets mit seinem Fachwissen zur Seite stand. Auch sonst war er ein wichtiger Diskussionspartner und ist nicht zuletzt auch durch private Gespräche zu einem wertvollen und freundschaftlichen Wegbegleiter während der letzten Jahre geworden.

Großen Dank schulde ich auch Herrn Prof. Peter Pelz, der mir ermöglichte diese Arbeit am Institut für Fluidsystemchnik zu schreiben. Seine Ratschläge sowie die gemeinsamen Diskussionen waren mir eine große Hilfe. Bei Herrn Prof. Alexander Martin bedanke ich mich für die Begutachtung dieser Arbeit und die wissenschaftliche Zusammenarbeit in den letzten Jahren.

Ein großer Dank geht auch an meine Kollegen für viele wertvolle Anregungen und konstruktive Kritik, besonders aber für das freundschaftliche Arbeitsklima.

Und nicht zuletzt danke ich meiner Frau, die mir stets Mut zugesprochen und mich in meiner Arbeit bestärkt hat. Hätte sie mir nicht den Rücken freigehalten, wäre meine Arbeit in dieser Form nicht möglich gewesen.

Jan Wolf
Darmstadt, den 18.11.2014

Zusammenfassung

Bei klassischen Optimierungsproblemen wird häufig davon ausgegangen, dass die Eingabedaten für ein gegebenes Problem zum Planungszeitpunkt bekannt sind. In der Praxis stösst man jedoch häufig auf Situationen, bei denen ein Teil dieser Daten mit Unsicherheiten behaftet ist und klassische Modelle nicht mehr geeignet sind, diese Probleme adäquat abzubilden. Eine Möglichkeit um Unsicherheitsaspekte in herkömmlichen Modellen zu integrieren, ist mittels der expliziten Quantifizierung von Entscheidungsvariablen.

Diesem Ansatz folgend beschäftigt sich die vorliegende Arbeit mit der Quantifizierung von (ganzzahligen) linearen Programmen, einer Erweiterung der klassischen (ganzzahligen) linearen Programmierung, bei der Entscheidungsvariablen nicht mehr wie im herkömmlichen Fall implizit *existenzquantifiziert* sind, sondern alternativ auch *allquantifiziert* sein können. Dies führt zu einer deutlichen Erhöhung der Ausdrucksstärke und ermöglicht es, Optimierungsprobleme unter Unsicherheit zu formulieren. In diesem Kontext wird momentan hauptsächlich die *Stochastische Optimierung* und die *Robuste Optimierung* angewandt, die Möglichkeiten der Quantifizierung von Entscheidungsvariablen sind weitestgehend unerforscht. Die vorliegende Arbeit ist in zwei Teile unterteilt und verfolgt das Ziel, die quantifizierte (ganzzahlige) lineare Programmierung als eigenständiges Modellierungsparadigma zu untersuchen. Das Hauptaugenmerk liegt dabei auf dem kontinuierlichen Fall.

Im ersten Teil wird dazu eine detaillierte Einführung in die grundlegenden Konzepte gegeben und auf die Unterschiede und Gemeinsamkeiten in Bezug auf andere Ansätze im Bereich der Optimierung unter Unsicherheit eingegangen. Es wird der Zusammenhang zu Zwei-Personen-Spielen hergestellt und anhand von Beispielen gezeigt, wie unterschiedliche reale Probleme modelliert werden können. Im Anschluss folgt eine ausführliche Untersuchung der theoretischen Eigenschaften von QLPs, welcher eine umfangreiche geometrische Betrachtung zu Grunde liegt. Des weiteren wird eine Komplexitätsuntersuchung von QLPs, basierend auf einer polyedrischen Betrachtung, präsentiert. Ziel dieser Bemühungen ist es, die Methoden und Erkenntnisse der linearen Optimierung anzuwenden und so zu erweitern, dass sie zur Lösung von QLPs genutzt werden können. Als Ergebnis wird im zweiten Teil ein Lösungsalgorithmus präsentiert, der die spezielle Eigenschaft von QLPs ausnutzt, dass diese auf der einen Seite konvexe mehrstufige Entscheidungsprobleme sind und auf der anderen Seite als Zwei-Personen-Spiele interpretiert werden können. Der Algorithmus ist eine Kombination aus klassischen LP-Lösungstechniken, angepasst auf die spezielle Struktur von QLPs, sowie dem Alpha-Beta Algorithmus aus der Spielbaumsuche, der z.B. beim Computerschach eingesetzt wird. Die im Rahmen dieser Arbeit vorgestellten Algorithmen wurden in dem QLP und QIP Framework `QlpOpt` implementiert und in einer detaillierten experimentellen Studie untersucht.

To my family

Abstract

In many real world problems, dealing with uncertainty is a significant challenge for mathematical programming and related areas, and typically, standard (mixed-integer) linear programming formulations are not well suited to model such problems. However, a powerful way to express uncertainty or adversarial situations is through the use of quantified variables. To pursue this approach, this thesis is concerned with *quantified (integer) linear programming*, a generalization of traditional *(integer) linear programming*. Whereas in traditional *linear programs* (LPs) and *integer programs* (IPs) all variables are implicitly existentially quantified, they can be either existentially or universally quantified in *quantified (continuous) linear programs* (QLPs) and *quantified integer programs* (QIPs).

Explicit quantification of variables yields a considerable increase in expressive power, suitable to compactly represent many problems from the field of optimization under uncertainty, a topic that has engaged much attention in the mathematical programming community and beyond in the last decade. In the context of optimization under uncertainty *robust optimization* and *stochastic programming* are currently the most prominent modeling paradigms. However, the abilities of linear programming extensions by variable quantification are relatively unexplored. The present thesis is subdivided into two parts and its aim is to study quantified linear programming as an independent mathematical programming paradigm. However, we mainly focus on the continuous case.

In the first part we give a detailed introduction into the concept of quantified (integer) linear programming and highlight the strong similarities to two-person zero-sum games and related work in the context of optimization under uncertainty. We furthermore present some illustrating examples to demonstrate the modeling power and broad applicability of this methodology. After a survey of the basic results of current research in this field both theoretical and applied, we also provide a detailed study of the algorithmic properties of QLPs by a comprehensive geometric analysis. As a result we present a complete and thorough complexity analysis by using a polyhedral approach and further analytical insights.

The second part of this thesis deals with the development and the implementation of an efficient algorithm to solve QLPs. We first review existing solution approaches and then develop an algorithm that exploits the problem's hybrid nature of being a convex multistage decision problem on the one hand, and being a two-person zero-sum game on the other hand. Based on the results of the theoretical analysis we propose an algorithm that combines linear programming techniques with solution techniques from game-tree search. This thesis is supplemented by the software `QlpOpt`, which is a quantified linear and integer programming solver and framework that features different solution techniques and a variety of methods to work with QLPs and QIPs.

Contents

1. Introduction

> Planning under uncertainty.
> This, I feel, is the real field we should all be working on.
>
> *(G.B. Dantzig)*

1.1. Motivation

Optimization is what we do all of the time. We strive for better products, faster cars, more efficient networks, and cheaper solutions. In engineering we wish to produce the best possible result with the available resources, owed by the ever-increasing demand on engineers to design and produce products both economically and efficiently. In the 1940s, linear programming arose as a mathematical planning and optimization model, integer linear programming and mixed-integer linear programming emerged in the late 1950s and the early 1960s. They rapidly found daily and widespread application in a large variety of domains like production planning, scheduling, location and transportation, or resource allocation in financial systems. Research in both continuous and discrete optimization has seen tremendous growth. As a result of intensive research in this field and due to the progress in the power and availability of computing, it is today possible to solve very large mixed-integer linear programs (MIPs) of practical size with up to millions of variables and constraints. Modern software for MIP optimization is highly sophisticated, with several commercial codes being actively developed and marketed.

Today, the application of optimization techniques also plays an increasingly important role in engineering sciences and many problems can be solved efficiently using modern (mixed-integer) linear programming techniques. Using these methods it is possible to optimize the design of a system of single components, which are composed in order to accomplish a certain functionality. However, the knowledge transfer of existing operations research techniques as e.g. successfully used in planning and logistics, to technical applications is elaborate and under-explored up to date but efforts to adapt these techniques to engineering sciences were made recently [223, 149, 157]. Making them interdisciplinary manageable in the context of technical system optimization is one of the main emphases of the *Technical Operations Research* (TOR) approach [168, 167]. Usually technical systems composed of single components are designed based on the practical experience of an engineer and afterwards verified and potentially improved via simulation and iteration. The TOR approach instead first describes the desired system function in a formal way and then uses mathematical optimization techniques to develop and structure a best suited system.

TOR can e.g. be used to design an optimal topology for a hydrostatic power transmission system as described in [86]. However, the TOR methodology is not necessarily restricted

to the synthesis of technical systems, but can also be applied in the context of production planning and scheduling where mixed-integer linear programming is successfully used in [172] for example. Another possible case of application is in the context of problems that are currently tackled by non-linear optimization techniques, like e.g. forming optimization or structural optimization. Non-linear optimization techniques suffer from the fact that they might get stuck at a local optimum and one possibility to ensure that a global optimum is found is a transformation of the non-liner optimization problem to a mixed-integer linear program as e.g. described in [101, 102] and applied to optimize a task in gas transportation [170]. Optimization problems of the latter form are also very similar to mathematical optimization of water networks [154]. Recently topics that deal with energy efficiency and environmental impact assessment become increasingly import when a new technical system is designed [168], but also in the context of traffic and transport, like e.g. fleet assignment.

An interesting branch of research in the field of TOR is the incorporation of uncertainty into models and solution techniques that were already successfully applied in the deterministic case. Uncertainty in engineering processes is also the central topic in the DFG founded Collaborative Research Centre (SFB) 805 *"Sonderforschungsbereich 805 - Beherrschung von Unsicherheit in lasttragenden Systemen des Maschinenbaus"* [1][2] at TU Darmstadt, which deals with the control of uncertainty in mechanical systems.

As a modeling tool, traditional (mixed-integer) linear programming is intended for deterministic problems. Thus, a common assumption is that all the input data used in the model formulation are deterministic and exactly known when the decisions have to be made. However, uncertainties are present in many real world optimization problems across all industries, and when a real-world problem is formulated as a mathematical optimization model, more often than not, the problem data are not certainly known, but are subject to uncertainty. This is due to several reasons, e.g., in supply chain optimization, the actual demand for products, financial returns, actual material requirements and other resources are not precisely known when critical decisions need to be made. In the case of a hydrostatic power transmission system for example, the potentially occurring load cases might be unknown in advance. In engineering and science, data is furthermore subjected to measurement errors, which also constitute sources of data uncertainty in the optimization model [201]. As a result decisive design parameters are often increased by $30\% - 50\%$ to hedge against these inaccuracies or other unexpected environmental influences [18]. Other indirect methods to cope with uncertainty is the addition of safety stock, the overestimation of demanded quantities, and oversized buffers for example.

Thus, there is a need for planning and deciding under uncertainty, and as a consequence, this field has experienced rapid development in the mathematical programming community from the very beginning in 1950s [15, 72, 16, 65]. The incorporation of uncertainty often pushes the complexity of traditional problems, which are in **P** or **NP**, to **PSPACE**, and also many optimization problems under uncertainty are **PSPACE**-complete [164]. Consequently, traditional (mixed-integer) linear programming models are not suitable to

[1] `http://www.sfb805.tu-darmstadt.de/sfb805`
[2] `http://gepris.dfg.de/gepris/projekt/96786330`

model situations in which decisions have to be made in the presence of uncertainty since they do not have enough expressive power. As a result, optimization under uncertainty has been actively studied over the last several decades by the mathematical programming community and beyond, and a wide variety of approaches to model the lack of information have been developed. The most prominent modeling paradigms and presently the subject of intense research in this field, are robust optimization [24, 42, 18, 36, 215], where uncertain variables are known to lie within some bounds, and stochastic programming [128, 174, 47, 189], where uncertain variables follow a known distribution. Strongly related in the area of decision theory is the field of Optimal Control [35, 141].

Another important way of modeling uncertainty, which has already been successfully applied in the context of propositional logic and constraint based reasoning in order to model decision making under uncertainty and adversarial situations, is through the use of quantified variables (i.e., $\exists$ and $\forall$). Quantified variables offer an intriguing increase in modeling power over traditional formulations, however, in the context of linear programming, this approach is relatively unexplored. Consequently, this thesis deals with quantified linear programming, a term coined by Subramani [212] to describe a generalization of traditional linear programming that allows the explicit quantification of variables. Whereas in traditional linear programs (LPs) all decision variables are implicitly existentially quantified, they can be either existentially or universally quantified in quantified linear programs (QLPs). Adding universally quantified variables yields a considerable increase in expressive power, and enables to combine traditional linear programming formulations with the description of uncertainty or the presence of an adversary. However, the extra expressiveness comes with an increase in complexity, and in contrast to traditional integer programs (IPs), which are known to be "only" $\mathbf{NP}$-hard in general, quantified integer programs (QIPs) are $\mathbf{PSPACE}$-complete. Many different problems can be efficiently encoded as $\mathbf{PSPACE}$-complete problems and there has been a great deal of recent interest and progress in solving such instances efficiently. The exact complexity class of QLPs is unknown general, however, they are at least $\mathbf{coNP}$-hard.

The aim of our research is to explore the abilities of LP-techniques when being applied to $\mathbf{PSPACE}$-complete problems, similar as when being applied to $\mathbf{NP}$-hard combinatorial optimization problems in the 1990s. Despite the intrinsically high complexity the goal of developing practically useful QLP and QIP solvers still seems to be feasible given sufficient conceptual and technical advances. Furthermore, many problems have a much more compact encoding when quantifiers are available, so a quantified solver can still be useful even if it can only deal with much smaller instances than a traditional solver.

In this thesis we mainly concentrate on QLPs since apart from being a highly interesting problem class itself, they are relaxations of QIPs to the continuous case, and at the same time they have structural properties of traditional LPs. Systems of linear inequalities define polyhedra and it is natural to optimize a linear function over them, which is the topic of linear programming. However, the polyhedral approach is also a powerful tool for solving mixed-integer linear programs (MIPs), and although integer programming is $\mathbf{NP}$-hard in general, polyhedral techniques are very successful in theory and practice. The solution of IPs and MIPs typically involves repeated solution of the problem's linear programming re-

laxation [13], e.g., in a branch-and-bound based solution procedure, the bounding usually comes from the LP relaxation, and the tighter the relaxation, the stronger is the bound. We think that QLPs allow the fast computation of bounds of a QIP by its QLP-relaxation in a similar manner, resulting in a considerable speedup of the solution process for QIPs. Consequently, any algorithmic advance in the solution of QLPs has a strong practical impact. Our aim is to develop a methodology for solving **PSPACE**-complete optimization problems with extended LP-techniques, and their combination with techniques from other fields successfully applied in the context of **PSPACE**-complete problems, as e.g. two-person games, constraint satisfaction or propositional logic. This thesis presents some new insights and techniques that make progress towards this goal.

1.2. Structure of the Thesis

The remaining part of this thesis consists of six chapters and is organized as follows:

- **Chapter 2: Quantified Linear and Integer Programming.** We give a detailed introduction into the concepts of quantified linear and integer programming and formally describe the continuous and the integer case. We present illustrating examples and reveal the strong connection to two-person game playing. Furthermore, we show up links to other related fields of research in the context of optimization under uncertainty and contrast the differences to QLPs and QIPs. At the end of this chapter we show several illustrating examples how to model real-world problems with the help of the quantified linear and integer programming approach.

- **Chapter 3: Polyhedral and Algorithmic Properties.** We provide a detailed study of the algorithmic properties of QLPs by a comprehensive geometric analysis. Therefore, we make intensive use of the problem's hybrid ambiguity, of being a convex multi-stage decision problem on the one hand, and being a two-person zero-sum game on the other hand. We furthermore present certain kinds of QLP relaxations.

- **Chapter 3: Algorithms and Complexity.** We review existing solution approaches for the continuous and the integer case, which are based on quantifier elimination techniques, the construction of an equivalent LP formulation, and a game-tree search algorithm respectively. As a main contribution of this chapter we accomplish a complete complexity analysis for the continuous case and review existing results for the integer case.

- **Chapter 4: The Alpha-Beta Nested Benders Decomposition Algorithm.** We propose a new algorithm to solve QLP optimization problems using a combination of linear programming techniques and techniques from game-tree search. As a result, we present an extension of the Nested Benders Decomposition algorithm by the $\alpha\beta$-heuristic and move-ordering, two techniques that are successfully used in game-tree search to solve minimax-trees. We furthermore develop a new sort of cut based on the computation of QLP-relaxations similar to the bounding mechanism used in modern MIP solvers.

- **Chapter 5: Implementation and Computational Results.** We present implementation details of our algorithm and the quantified linear and integer programming framework and solver `QlpOpt`. We evaluate the performance of our algorithmic implementation in a detailed computational study, where we compare it with other existing solution techniques on a large test set of QLP instances.

- **Chapter 6: Conclusion and Outlook.** We summarize the findings of the prevailing chapters and offer an outlook for future considerations in the field of quantified linear programming.

1.3. Own Contribution to Knowledge

The contribution of this thesis is twofold and important from both theoretical and practical perspectives:

- We give a broad introduction in the topic of quantified (integer) linear programming from the viewpoint of mathematical optimization. We propose an extension of the decision problem by the addition of a linear objective function, which tries to minimize the objective function with respect to the maximum possible loss that can result from the adversarial or uncertain behavior represented by the universally quantified variables.

- We present a comprehensive geometric analysis for the continuous case and present for the first time a complete and thorough complexity analysis by using a polyhedral approach. We show for the continuous case that the solution space of a QLP forms a polytope in $\mathbb{Q}^n$ and that its vertices can be encoded with polynomially many bits if the number of quantifier changes is constant. We show that in this case the QLP decision problem is in the complexity class $\mathbf{PSPACE}$.

- We develop an algorithm to solve QLP optimization problems, which exploits the problem's hybrid nature of being a convex multi-stage decision problem on the one hand, and being a two-person zero-sum game on the the hand. It combines solution techniques from traditional linear programming and techniques successfully applied in game-tree search. Many of the techniques are gathered from the literature, but their combination as well as a lot of algorithmic subtleties and small improvements are new developments.

- We present a definitive evaluation of our algorithm as a practical solution method for QLPs. Prior to this work the only known approaches to the solution of QLPs involved sequential variable elimination techniques or the solution of an exponentially large LP reformulation. We compare the strengths and weaknesses of these approaches with our algorithm in a detailed computational study.

- This thesis is supplemented by the software `QlpOpt`, which is a quantified linear programming solver and framework that features different solution techniques and a

variety of methods to work with QLPs, QIPs, and QMIPs. The sources can be made available upon request for research purposes by the author.

The results of this thesis are based on the following set of publications. The work is done in close cooperation with the respective co-authors:

- U. Lorenz, A. Martin, J. Wolf: Polyhedral and Algorithmic Properties of Quantified Linear Programs. ESA 2010: 512-523.
- T. Ederer, U. Lorenz, A. Martin, J. Wolf: Quantified Linear Programs: A Computational Study. ESA 2011: 203-214.
- T. Ederer, U. Lorenz, T. Opfer, J. Wolf: Modeling Games with the Help of Quantified Integer Linear Programs. ACG 2011: 270-281.
- T. Ederer, U. Lorenz, A. Martin, T. Opfer, J. Wolf: Polyhedral Properties and Algorithmic Aspects of Quantified Linear Programs. Submitted to: *"SIAM Journal on Discrete Mathematics"*, TU Darmstadt, Institute of Mathematics, 2012.
- U. Lorenz, J. Wolf: Solving Multistage Quantified Linear Optimization Problems with the Alpha-Beta Nested Benders Decomposition. Submitted to: *"EURO Journal on Computational Optimization"*, TU Darmstadt, Institute of Mathematics, 2013.
- U. Lorenz, T. Opfer, J. Wolf: Solution Techniques for Quantified Linear Programs and the links to Gaming. CG2013, August 13-15, 2013, Yokohama, Japan, Lecture Notes in Computer Science LNCS, Springer Verlag, Heidelberg/Berlin, 2014, 110-124.

1.4. Notation and Basic Definitions

In this section we provide some intuition and introduce preliminaries, which are needed in subsequent parts of the thesis. These include notation, basic definitions and well-known results. We adhere to standard notation of linear algebra [197]. $\mathbb{N}$ is the set of natural numbers, $\mathbb{Z}$ the set of integer numbers, $\mathbb{Q}$ the set of rational numbers and $\mathbb{R}$ the set of real numbers. Since many definitions and theorems hold for $\mathbb{Q}$ and $\mathbb{R}$ we use the symbol $\mathbb{K} \in \{\mathbb{Q}, \mathbb{R}\}$ to avoid their distinction.

Small italic letters (a, b, ...) denote column vectors, while italic letters (A, B, ...) denote matrices. a_i denotes the i-th element in a, a^T denotes the transposed vector of a and $\cdot$ denotes the inner product. $\mathbb{K}^n$ is the set of n-vectors with components from $\mathbb{K}$. Vectors are column vectors $x = (x_1, x_2..., x_n)^T \in \mathbb{K}^n$. Row vectors are denoted by x^T with $x \in \mathbb{K}^n$. $\mathbb{K}^{m \times n}$ is the set of all $m \times n$ matrices with elements from $\mathbb{K}$.

Quantified linear and integer programming is mainly motivated by (mixed-integer) linear programming, two-person games, and complexity theory. As a preparation for the rest of this thesis, we briefly review basic definitions and theory from linear algebra and linear programming in the following. For a full treatment including the missing proofs, we refer the reader to one of the standard text books [197, 161, 75, 76].

Polyhedra, Extreme Points, Extreme Rays

Definition 1.1 (Hyperplane)**.** *A subset $G \subseteq \mathbb{K}^n$ is called **hyperplane**, if there is an $\alpha \in \mathbb{K}$ and a vector $a \in \mathbb{K}^n \setminus \{0\}$ (a normal) such that*

$$G = \{x \in \mathbb{K}^n | a^T x = \alpha\}.$$

Definition 1.2 (Half-space)**.** *A subset $H \subseteq \mathbb{K}^n$ is called **half-space**, if there is an $\alpha \in \mathbb{K}$ and a vector $a \in \mathbb{K}^n \setminus \{0\}$ (a normal) such that*

$$H = \{x \in \mathbb{K}^n | a^T x \leq \alpha\}.$$

A half-space contains the set of all points that lie above or beyond a hyperplane.

Definition 1.3 (Polyhedron)**.** *A set $P \subseteq \mathbb{K}^n$ is called **polyhedron**, if P results from the intersection of finitely many half-spaces.*

Remark 1. *A polyhedron can be written as*

$$P = P(A, b) = \{x \in \mathbb{K}^n | Ax \leq b\},$$

with $A \in \mathbb{K}^{m \times n}$ and $b \in \mathbb{K}^m$.

Definition 1.4 (Polytope)**.** *A polytope is a bounded polyhedron.*

Definition 1.5 (Convex Set)**.** *A set $S \subseteq \mathbb{K}^n$ is convex, if for any x, $y \in S$ and $0 \leq \lambda \leq 1$, $\lambda x + (1 - \lambda)y \in S$.*

Definition 1.6 (Convex Combination)**.** *A point $x \in \mathbb{K}^n$ is a convex combination of a set $S \subseteq \mathbb{K}^n$ if x can be expressed as $x = \sum_i \lambda_i x^i$ for a finite subset $\{x^i\}$ of S and $\lambda \geq 0$ with $\sum_i \lambda_i = 1$*

Definition 1.7 (Convex Hull)**.** *Given a non-empty set of points X, the smallest convex set that contains X is called the convex hull of X.*

Remark 2. *A bounded polyhedron is the convex hull of a finite set of points.*

Definition 1.8 (Extreme Point)**.** *A point $x \in P$ is an extreme point of P, if there do not exist two distinct points x_1 and x_2 such that $x = \lambda x_1 + (1 - \lambda)x_2$ for $0 < \lambda < 1$.*

Definition 1.9 (Extreme Ray)**.** *Given a non-empty polyhedron $P = \{x \in \mathbb{K}^n \mid Ax \leq b\}$, a vector $r \in \mathbb{K}^n$, $r \neq 0$, is a ray of P if $Ar \leq 0$. Two rays r_1 and r_2 are distinct if there is no constant $\lambda > 0$ such that $r_1 = \lambda r_2$. A ray r of P is an extreme ray of P id there do not exist two distinct rays r_1 and r_2 such that $r = \lambda r_1 + (1 - \lambda)r_2$ for $0 < \lambda < 1$.*

Remark 3. *A polyhedron P has a finite number of extreme points and extreme rays [161].*

Linear and (Mixed-)Integer Linear Programs

Definition 1.10 (Linear Program)**.**
Let $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ be given. Then

$$z_{LP} = \min_{x \in \mathcal{P}_{LP}} \{c^T x\} = \min\{c^T x | Ax \leq b\,, x \in \mathbb{R}^n\} \tag{LP}$$

is called a (deterministic) linear program (LP). $\mathcal{P}_{LP} = P(A, b)$ is called solution space of the LP.

Remark 4. *A vector $x \in \mathbb{R}^n$ is valid for LP, if $x \in \mathcal{P}_{LP}$. In this case we call x a solution for LP. A vector $\overline{x} \in \mathbb{R}^n$ is called optimal for LP, if $\overline{x} \in \mathcal{P}_{LP}$, and $c^T x \geq c^T \overline{x}$ for all $x \in \mathcal{P}_{LP}$. The value z_{LP} is called the objective function value of LP.*

Definition 1.11 (Integer (Linear) Program)**.**
Let $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ be given. Then

$$z_{IP} = \min_{x \in \mathcal{P}_{IP}} \{c^T x\} = \min\{c^T x | Ax \leq b\,, x \in \mathbb{Z}^n\} \tag{IP}$$

is called an integer (linear) program (IP). $\mathcal{P}_{IP} = conv(\mathcal{P}_{LP} \cap \mathbb{Z}^n)$ is called integral convex hull.

Definition 1.12 (LP-relaxation of an IP)**.**
Given an integer program

$$z_{IP} = \min\{c^T x \,|\, Ax \leq b,\, x \in \mathbb{Z}^n\},$$

its LP-relaxation is defined as

$$z_{LP} = \min\{c^T x \,|\, Ax \leq b,\, x \in \mathbb{R}^n\}.$$

Remark 5. *Since $\mathcal{P}_{IP} \subseteq \mathcal{P}_{LP}$, it is clear that $z_{LP} \leq z_{IP}$.*

Definition 1.13 (Mixed-Integer (Linear) Program)**.**
Let $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ be given. Then

$$z_{MIP} = \min\{c^T x \,|\, Ax \leq b,\, x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \text{ with } p \in \mathbb{Z},\, 0 \leq p \leq n\}, \tag{MIP}$$

is called a mixed-integer (linear) program (IP).

Linear Programming Duality and Farkas' Lemma

Proposition 1.1 (Farkas' Lemma)**.** *Given a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$, then exactly one of the following systems has a feasible solution:*

$$\begin{array}{r} Ax = b \\ x \geq 0 \end{array} \qquad \dot{\bigvee} \qquad \begin{array}{r} v^T A \geq 0 \\ v^T b < 0 \end{array} \tag{Farkas' Lemma}$$

Remark 6. *Given an infeasible LP problem, i.e., its feasible region $\mathcal{P}_{LP}$ is empty, we call a vector v satisfying the hypothesis of Farkas' Lemma a certificate of infeasibity for the LP problem. Note that the certificate also holds if the LP problem has an objective function.*

Definition 1.14 (The Dual LP)**.** *Given an LP problem*

$$z = \min\{c^T x \,|\, Ax \leq b,\, x \in \mathbb{R}^n\}, \tag{PLP}$$

its dual problem is defined as follows:

$$w = \max\{y^T b \,|\, y^T A = c^T,\, y \leq 0\}. \tag{DLP}$$

Proposition 1.2 (Weak Duality)**.** *If $\hat{x}$ is a feasible solution for PLP with objective function value $\hat{z} = c^T \hat{x}$, and $\hat{y}$ is a feasible solution for DLP with objective function value $\hat{w} = \hat{y}^T b$ then $\hat{z} \leq \hat{w}$.*

Proposition 1.3 (Strong Duality)**.** *If either the primal problem PLP or dual problem DLP is feasible and bounded, then both problems are feasible and bounded with the same optimal objective function value $\overline{z} = \overline{w}$.*

Corollary 1.4. *If the primal problem is unbounded then the dual problem is infeasible. Similarly, if the dual problem is unbounded then the primal problem is infeasible. Note that the converse is not true, both the primal and dual problem may be infeasible.*

2. Quantified Linear and Integer Programming

In this chapter we introduce quantified linear and integer programming as an independent mathematical programming paradigm. In Section 2.1 we formally describe the problem statement. In Section 2.2 we give an overview of related work and finally, in Section 2.3, we illustrate how to model some interesting real-world problems with the help of the quantified linear and integer programming approach.

2.1. The Problem Statement

The term "quantified linear programming" was coined by Subramani [208] to describe linear programming problems in which one or more of the variables are universally quantified over some domain and the rest of the variables are existentially quantified in the usual sense. The semantics of universal variable quantification is that all constraints must be satisfiable for all possible assignments to it. We first introduce decision problems and their integer variants, afterwards we derive optimization problems and show the strong relationship to two-person zero-sum games with the help of some illustrating examples. We follow the notation as presented in [147, 88].

2.1.1. Quantification of Linear and Integer Programs

A linear program in which some of the decision variables are universally quantified is called a quantified linear program and can be defined as follows:

Definition 2.1 (Quantified Linear Program (QLP))**.**
Let there be a vector of variables $x = (x_1, \ldots, x_n)^T \in \mathbb{Q}^n$*, integral lower and upper bounds* $l \in \mathbb{Z}^n$ *and* $u \in \mathbb{Z}^n$ *with* $l_i \leq x_i \leq u_i$ *, a coefficient matrix* $A \in \mathbb{Q}^{m \times n}$*, a right-hand side vector* $b \in \mathbb{Q}^m$ *and a vector of quantifiers* $Q = (Q_1, \ldots, Q_n)^T \in \{\forall, \exists\}^n$*, let the term* $Q \circ x \in [l, u]$ *with the component-wise binding operator* $\circ$ *denote the* quantification vector $(Q_1 x_1 \in [l_1, u_1], \ldots, Q_n x_n \in [l_n, u_n])^T$ *such that every quantifier* Q_i *binds the variable* x_i *ranging over the interval* $[l_i, u_i]$*. We call* (Q, l, u, A, b) *with*

$$Q \circ x \in [l, u] : Ax \leq b \tag{QLP}$$

a quantified linear program (QLP).

We call $Ax \leq b$ the *constraint system* and denote the quantification vector $Q \circ x \in [l, u]$ as quantification sequence $Q_1 x_1 \in [l_1, u_1] \ \ldots \ Q_n x_n \in [l_n, u_n]$. In a similar manner, we

may denote Q as a quantifier sequence $Q_1 \ \dots \ Q_n$ and x as a variable sequence $x_1 \ \dots \ x_n$. We require that no variable appears twice in Q. The statement $x_i \in x$ means that variable x_i is an item of sequence x. Each maximal consecutive subsequence of Q consisting of identical quantifiers is called a *quantifier block* – the corresponding subsequence of x is called a *variable block*. The total number of blocks less one is the number of *quantifier changes*.

We say a variable x_i is *universal (existential)* if its quantifier Q_i is $\forall(\exists)$. Let I be the (ordered) index set of x and let $I_\exists \coloneqq \{i \in I : Q_i = \exists\}$ and $I_\forall \coloneqq \{i \in I : Q_i = \forall\}$ be (ordered) subsets with respect to a specific quantifier. That is, $x_{I_\exists}$ and $x_{I_\forall}$ denote the (ordered) sets of existentially quantified, and respectively, universally quantified variables of the vector x. For convenience, we generally omit the letter I and write $x_\exists$ for $x_{I_\exists}$ and $x_\forall$ for $x_{I_\forall}$. Likewise, let $x_{\exists i} \coloneqq x_{((I_\exists)_i)}$ denote the i-th existentially quantified variable. Accordingly, we denote (ordered) variable blocks by a superscripted index x^t for $t = 1, \dots, H$, where H is the number of quantifier blocks. Likewise, we use $x^t_\exists$ to denote the t-th block of existentially quantified variables for all $t = 1, \dots, H_\exists$. Take for example the quantifier string $\exists x_1 \forall x_2 \forall x_3 \exists x_4$, with four variables and three variable blocks $t = \{1, 2, 3\}$, we identify variable x_4 with $x_{\exists 2}$ and variable block $x^2 = x^1_\forall = (x_2, x_3)^T$. The total number of variables of a QLP is denoted by $|x|$, $|x_\exists|$ and $|x_\forall|$ denote the number of existentially and universally quantified variables respectively.

Furthermore, let A^t denote the (ordered) set columns of A that belong to the t-th block of variables. It is also customary to separate the coefficients of the existentially quantified variables and the universally quantified variables. That is, $A_\exists$ denotes the (ordered) set of columns of A that belong to the existential variables $x_\exists$ and $A_\forall$ the (ordered) set of columns that belong to the universal variables $x_\forall$. Accordingly, let $A^t_\exists$ denote the (ordered) set of columns of $A_\exists$ that belong to the t-th block of variables in $x_\exists$ and $A^t_\forall$ the (ordered) set of columns of the t-th universal variable block in $x_\forall$.

Example 1. *The mathematical program represented by*

$$\exists\, x_1 \in [0,1]\ \forall\, x_2 \in [0,1]\ \exists\, x_3 \in [0,1]\ \forall\, x_4 \in [0,1]\ \exists\, x_5 \in [0,1] :$$

$$\begin{pmatrix} 0 & -1 & -1 & 0 & 0 \\ -1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & -2 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 2 & 2 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} -1 \\ 1 \\ 3 \\ 1 \\ 2 \\ 3 \end{pmatrix}$$

is a simple QLP with four quantifier changes. It consists of three existential and two universal variables. The problem statement can be verbalized as follows: "Is there an allocation for variable x_1 such that for all possible allocations of x_2 there is an allocation for x_3 such that for all possible allocations of x_4 there is an allocation for x_5 such that all constraints in the constraint system $Ax \leq b$ are satisfied?"

The idea behind universal variables is that they must be able to take any of their values and they can be used to model actions or events which are uncertain, or just those which are not in the control of the decision maker but an adversary. The values in the domain of a universal variable capture all possible outcomes and in this way QLPs can model bounded uncertainty. Variables under the control of the decision maker can be modeled using existentially quantified variables, those outside the control of the decision maker are modeled using universally quantified variables.

The domain of the universal variables is a k-dimensional hyperrectangle and results from the cartesian product of the intervals that correspond to the universal variables:

$$E = \prod_{i=1}^{k=|x_\forall|} [l_{\forall i}, u_{\forall i}] = [l_{\forall 1}, u_{\forall 1}] \times [l_{\forall 2}, u_{\forall 2}] \times \cdots \times [l_{\forall k}, u_{\forall k}].$$

The semantics of quantifier alternation is to specify the timing of the decisions, relative to the resolution of uncertainty or the action of an adversary. In many real world problems, decisions and uncertainty are often interwoven over time. That is, a decision must be taken and then an event is observed. Afterwards again a decision must be taken followed by a repeated observation again, and so on. Thus, QLPs are multi-stage decision problems. For sake of simplicity, we can always assume a strictly alternating appearance of existential and universal quantifiers with an existential quantifier at the beginning and at the end of the quantification vector, because we can w.l.o.g. always add dummy variables without affecting the correctness or complexity of the problem [165]. This more general version will often be used in our technical results, so that the analysis is simplified. In general, each variable is bounded from above and below in the quantification vector, however, the bounds of existentially quantified variables can be alternatively placed within the constraint system instead ($x_i \in [l_i, u_i]$ can be represented by the two constraints $x_i \geq l_i$ and $x_i \leq u_i$). For universally quantified variables this is not possible, otherwise the resulting QLP problem would be trivially false. However, according to the results from [147] each QLP can furthermore be re-written to the effect that all universally quantified variables are from a $[0, 1]$ interval. Note that in this case the domain of the universal variables is a k-dimensional hypercube with $k = |x_\forall|$. Thus, if needed, a QLP can be alternatively written as:

$$\exists x_1 \; \forall x_2 \in [0,1] \ldots \exists x_{n-2} \; \forall x_{n-1} \in [0,1] \; \exists x_n : (A_\exists, A_\forall) \cdot \begin{pmatrix} x_\exists \\ x_\forall \end{pmatrix} \leq b.$$

The QLPs discussed thus far have all been continuous, in the sense that the decision variables are only allowed to be fractional. However, as with linear programs and integer programs, we can restrict the variables of a QLP to a discrete domain.

Definition 2.2 (Quantified Integer Program (QIP))**.**
A quantified linear program as in Definition 2.1 with

$$Q \circ x \in [l, u] \cap \mathbb{Z}^n : Ax \leq b \tag{QIP}$$

is called quantified integer program (QIP).

Thus, QIPs are a generalization of IPs to the quantified case, and they are also closely related to *quantified boolean satisfiability (QSAT)* problems and extend them by arbitrary integral values and arbitrary linear constraints instead of binary variables and clauses (see also Section 2.2.3). Quantified integer programming can also be thought of as a restriction of Presburger Arithmetic where only conjunctions of linear inequalities are allowed [211]. In QIPs, the variables can assume arbitrary values, but the case where integer variables are restricted to be 0 or 1 comes up surprisingly often.

Definition 2.3 (Binary QIP (0/1-QIP))**.**
A quantified integer program as in Definition 2.2, in which all integer variables are binary is called binary quantified integer program (0/1-QIP).

Note that any bounded integer variable can be expressed as a combination of binary variables [197] (e.g. an integer variable $0 \leq x \leq U$, can be expressed using $\lfloor \log_2 U \rfloor + 1$ binary variables), in the same fashion QIPs can be transformed to 0/1-QIPs. As in the case of traditional LPs and IPs we can also mix continuous and integer variables in the quantified case.

Definition 2.4 (Quantified Mixed Integer Program (QMIP))**.**
A quantified linear program as in Definition 2.1 with some $x_i \in \mathbb{Z}$ is called a quantified mixed-integer program (QMIP). QMIP instances in which all existential variables are continuous and all universal variables are discrete are called universally discrete*, instances with discrete existential variables and continuous universal variables are called* existentially discrete.

When the discrete ranges of the universally quantified variables of a QIP or a QMIP are relaxed into continuous ranges, we call the resulting QLP the *QLP-relaxation*.

Definition 2.5 (QLP-relaxation of a QIP)**.**
Given a quantified integer program

$$Q \circ x \in [l, u] \cap \mathbb{Z}^n : Ax \leq b \tag{2.1}$$

its QLP-relaxation is defined as

$$Q \circ x \in [l, u] \cap \mathbb{Q}^n : Ax \leq b. \tag{2.2}$$

Enlarging the domain of existentially quantified variables is the direct equivalence to the LP-relaxation of an IP in the classical case. As we will later see, there are several cases where QIPs or QMIPs can be relaxed to QLPs without altering the solution space. In Section 3.3 we show that the generality of QLPs allows the definition of a number of different forms of relaxations not available in classical linear programming.

2.1.2. Special Subclasses

Opposed to traditional linear programs, where the order of the variables is meaningless, the order of the variable blocks in the quantification vector of QLPs is of particular importance.

Consider the following example

$$\exists x_1 \in [0,1]\ \forall x_2 \in [0,1] :\ x_1 - x_2 = 0,$$

which is infeasible, while the QLP

$$\forall x_2 \in [0,1]\ \exists x_1 \in [0,1] :\ x_1 - x_2 = 0$$

with the same constraint system $Ax \leq b$ but different variable order in Q is feasible.

Consequently, the structure of the quantification vector and the number of quantifier changes are important properties to classify QLPs. Thus, we can restrict the number of quantifier changes to a fixed constant c, with $c = \infty$ resulting in general QLPs.

Definition 2.6 (c-QLP)**.**
A quantified linear program as in Definition 2.1 with a number of quantifier changes that is bounded by a fixed constant c is called c-QLP.

Note that neither the number of variables nor the number of constraints is restricted in the c-QLP problem. Two border cases result for $c = 0$ when there is no quantifier change, and thus all variables in the quantification vector are existentially quantified, or all variables are universally quantified respectively. Existential-only quantification results in conventional linear programs and the notation can be verbalized as *"There exist variables allocations in the range that satisfy the given constraints"*.

Definition 2.7 ($\exists$-QLP)**.**
A quantified linear program as in Definition 2.1 with

$$\exists^n \circ x \in [l,u] : Ax \leq b \qquad (\exists\text{-QLP})$$

is called a $\exists$*-QLP.*

Accordingly, universal-only quantification can be verbalized as *"All possible variables allocations in the range satisfy the given constraints"*.

Definition 2.8 ($\forall$-QLP)**.**
A quantified linear program as in Definition 2.1 with

$$\forall^n \circ x \in [l,u] : Ax \leq b \qquad (\forall\text{-QLP})$$

is called $\forall$-QLP.

Two interesting special cases of the c-QLP problem arise for $c = 1$ when there are exactly two blocks of quantifiers. In this case, we distinguish whether the quantification vector starts with the $\forall$-block, or with the $\exists$-block.

Definition 2.9 ($\exists\forall$-QLP)**.**
A quantified linear program as in Definition 2.1 with

$$(\exists_1\ \ldots \exists_k\ \forall_{k+1}\ \ldots\ \forall_n)^T \circ x \in [l,u] : Ax \leq b \qquad (\exists\forall\text{-QLP})$$

is called $\exists\forall$-QLP.

Definition 2.10 ($\forall\exists$-QLP)**.**
A quantified linear program as in Definition 2.1 with

$$(\forall_1 \dots \forall_k \exists_{k+1} \dots \exists_n)^T \circ x \in [l, u] : Ax \leq b \qquad (\forall\exists\text{-QLP})$$

is called $\forall\exists$-QLP.

In general there are more than one quantifier changes and to reflect the close relationship to stochastic programming (see Section 2.2.2), we adopt the terms *two-stage* and *multi-stage* to describe that a QLP has two or respectively a multiple number of existential variable decision blocks.

The corresponding integer and mixed-integer variants of these special subclasses arise accordingly. In traditional integer programming, which is **NP**-hard in general, interesting subclasses of lower complexity often arise from restrictions to the constraint system. Concerning this matter, QIPs were studied in [211] and a number of special cases where analyzed from the perspective of computational complexity. Subclasses of the QIP problem arise either by a restriction of the constraint system or the quantifier specification. For more details see [211].

2.1.3. QLPs and QIPs as two-person zero-sum games

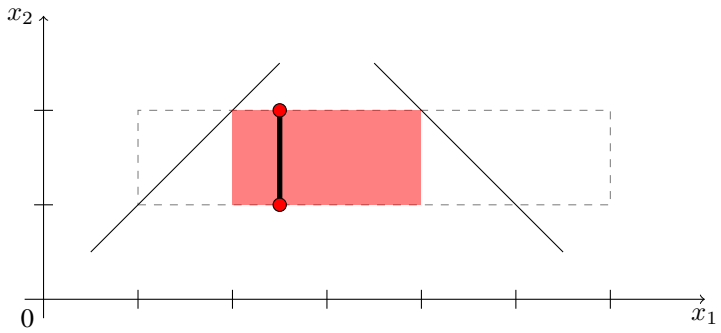
From a game-theoretic viewpoint, the semantics of quantifier alternation in a QLP or QIP can be interpreted as a two-person zero-sum game between an *existential player* setting the $\exists$-variables and a *universal player* setting the $\forall$-variables of a QLP. Each fixed vector $x \in \mathbb{Q}^n$, that is when the existential player has fixed the existential variables and the universal player has fixed the universal variables, is called a *game* or *playout*. If x satisfies the linear program $Ax \leq b$, we say *the existential player wins*, otherwise *he loses* and *the universal player wins*. The variables are set in consecutive order according to the variable sequence. Consequently, we say that a player makes the move $x_k = z$, if he fixes the variable x_k to the value z. At each such move, the corresponding player knows the settings of $x_1, \dots, x_{k-1}$ before taking his decision. The $\exists$-player tries to satisfy all constraints in the constraint system, while the $\forall$-player (the adversary) tries to break at least one constraint. The goal is to decide whether the $\exists$-player can certainly win the game independently of the possible moves of the $\forall$-player. If we assume an uninterested $\forall$-player that plays at random then the game becomes a *Game Against Nature* as studied in [164].

Example 2. *The QLP*

$$\exists x_1 \in [1, 6]\ \forall x_2 \in [1, 2] : x_1 + x_2 \leq 6\ \wedge\ x_2 - x_1 \leq 0$$

has the following graphical representation of bounding box (dashed lines) and constraints (solid lines). We say a solution *to this problem is a move for the existential player such that he wins the game regardless of the universal player's reaction, or rather, the set of games* $(x_1, x_2)^T$ *that can result from the existential player's decision (black line segment*

in the figure below). Naturally, a game can be interpreted as a point in $\mathbb{Q}^n$. The set of all solutions, i.e., the set of existential sticks fitting in the specified trapezoid, is called the solution space *(filled rectangle). Later on, we will see that it indeed suffices to analyze discrete points (filled dots) to find a solution. Note again that the order in which the quantifiers occur is crucial.*



Observe that in general a solution of a QLP is not a numeric vector as in the case of an LP, but a *policy*, e.g. an algorithm $\mathcal{A}$ that implements a set of computable functions of the form $x_i = f_i(x_1, \ldots, x_{i-1})$ for all existentially quantified variables x_i. The question of interest is, whether there is a *winning policy* for the existential player that guarantees him to win the game, independently of the universal player's actions.

Definition 2.11 (Policies)**.** *Given a QLP (Q, l, u, A, b) with $Q \circ x \in [l, u] : Ax \leq b$. An algorithm that fixes all existential [universal] variables x_i with the knowledge, how $x_1, \ldots, x_{i-1}$ have been set before, is called an* existential [universal] policy.

A policy can be represented as a set of computable functions $x_i = f_i(x_1, \ldots, x_{i-1})$ for all existentially [universally] quantified variables x_i. A policy is called a *winning policy* for the existential [universal] player, if these functions ensure that the existential [universal] player wins all games that can result from this policy. When universal quantification is removed, then a policy is simply a static assignment to the variables and it is said to be winning if all constraints are satisfied.

Note that each instance of the QLP problem has a winning policy – either for the existential or for the universal player. That is, exactly one of the two players can always win the game by playing perfectly. Now, the question if the existential player (no matter if he is the first or second player) is able to (certainly) win the game can be stated as follows:

Definition 2.12 (QLP Feasibility Problems)**.** *Given a QLP, the decision problem: "Is there a winning policy for the existential player" is called the* QLP feasibility problem.

The chronological order of all possible moves as determined by the quantifier string of a QIP can be represented by a game-tree.

Definition 2.13 (Game-Trees)**.** *Let $G = (V, E, L)$ be the edge-labeled finite arborescence with a set of nodes $V = V_\exists \dot\cup V_\forall$, a set of edges E and a vector of edge labels $L \in \mathbb{Q}^{|E|}$.*

All paths have the same length and each level of the tree (the set of all nodes at a given depth, with the root node at level 0) consists either of nodes from $V_\exists$ or $V_\forall$ only.

Given an instance of $Q \circ x \in [l, u] \cap \mathbb{Z}^n : Ax \leq b$, the i-th variable of the QIP is represented by the inner nodes of depth $i - 1$. Each inner node has as many children as there are integers in the corresponding variable range. Its outgoing-edge labels enumerate these values and represent all possible moves of the player at the current node. A path from the root to a leaf represents a game of the QIP and the sequence of edge labels encodes its moves and the leaf node corresponds to its outcome. In principle, a quantified integer program can be solved by evaluating the whole game-tree. However, in general it suffices to investigate only a small part of the game-tree, because each QIP policy can be represented by a (unique) subtree. We call these subtrees *strategies* for the QIP.

Definition 2.14 (Strategies)**.** *A subtree $\mathcal{S} = (V', E', L')$ of a game-tree $G = (V, E, L)$ is called a* strategy*, if*

- *each node $v'_\exists \in V'_\exists$ which is an inner node of G has exactly one child in $\mathcal{S}$ and each node $v'_\forall \in V'_\forall$ which is an inner node of G has as many children in $\mathcal{S}$ as in G* (existential strategy)*, or*
- *each node $v'_\forall \in V'_\forall$ which is an inner node of G has exactly one child in $\mathcal{S}$ and each node $v'_\exists \in V'_\exists$ which is an inner node of G has as many children in $\mathcal{S}$ as in G* (universal strategy)*.*

A strategy for the existential player, also called $\exists$-*strategy*, defines how to react on each possible move of the universal player. Branches of the tree are *scenarios*, leafs correspond to *final positions* of the game. As above, this tree can formally be defined as a set of functions specifying the value that should be assigned to each existential variable, given the assignment of its existential and universal predecessors.

Given an instance of $Q \circ x \in [l, u] \cap \mathbb{Z}^n : Ax \leq b$, a strategy is called a *winning strategy* if all paths from the root node to a leaf represent a vector x such that $Ax \leq b$. Winning strategies are a central concept of quantifier alternation and this terminology is also very similarly used in game-tree search [171]. When universal quantification is removed from the game, then a strategy is simply a static assignment to the variables and corresponds to a single path from the root to a leaf, which is again said to be winning if all constraints are satisfied.

Example 3. *The QLP*

$$\exists\, x_1 \in [0,1]\ \forall\, x_2 \in [0,1]\ \exists\, x_3 \in [0,1] :$$

$$\begin{pmatrix} 0 & -1 & -1 \\ -1 & 1 & 1 \\ 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} -1 \\ 1 \\ 3 \end{pmatrix}$$

has two quantifier changes. Figure 3.6 shows a visualization of the constraint polyhedron restricted to the unit cube.

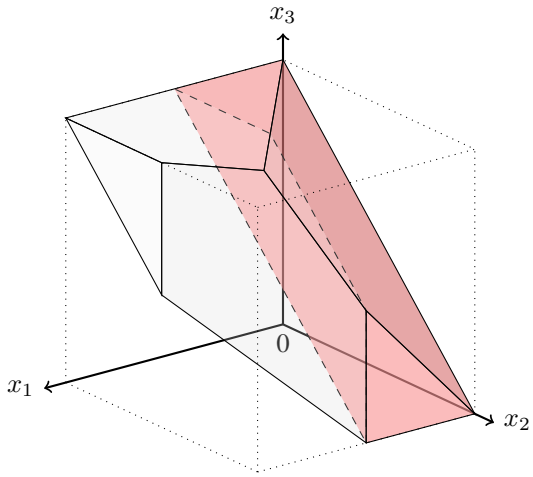


Figure 2.1.: Constraint polyhedron restricted to the unit cube and the winning polytope resulting from all winning policies (cf. Section 3.2.2).

Since this example is rather small, we can guess a winning policy for the existential player from the picture: Choose $x_1 \in [0, \frac{1}{2}]$, then choose x_3 apt to x_2, e.g. $x_3 = 1 - x_2$. The dark-shaded solution space visualizes the set of all games with a definite winning outcome for the existential player. Note that depending on the first two moves, the choice of x_3 may be unique. At least in this example, the solution space of the QLP is polyhedral and a subset of the solution space of its LP-relaxation where universal quantification is dropped.

The corresponding QIP results from restricting all three variables to their binary bounds. The game-tree of this instance is shown in Figure 2.2. A square represents an existential node and a circle represents a universal node. A plus leaf depicts a winning outcome for the existential player and a minus leaf a losing outcome.

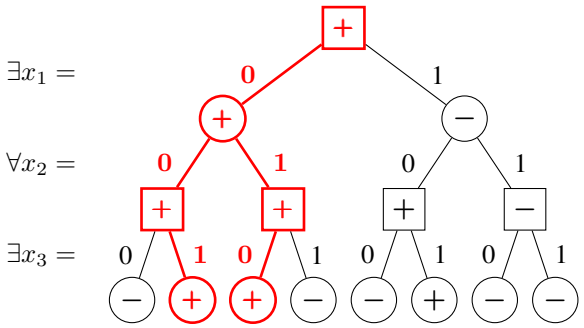


Figure 2.2.: Game-tree and winning strategy for the existential player.

The sign in an existential or universal node indicates the best or respectively worst value of its successors, i.e., if a player can choose between a plus and a minus node, the

existential player would certainly choose a plus node and the universal player a minus node. Therefore, a plus sign in any node means the existential player can still win the game, if he makes no mistake in his remaining moves. In this respect, a winning strategy for the existential player (like the highlighted subtree) can be understood as an existential strategy of plus nodes.

Finding a winning strategy among a finite set of possible moves is a purely combinatorial problem. On the contrary, real-valued variables typically represent coordinates in the n-dimensional space. These two problems illustrate the two sides of quantified linear (integer) programs, which both raise challenging computational issues.

Note that not only QIP policies but also universally-discrete QMIP policies for the existential player can be represented by trees similar to strategies. The only difference is that these trees do not stem from a game-tree, because it would have infinitely many edges leaving each continuous node. Nevertheless, the notion of a strategy remains reasonable for the continuous player.

From the viewpoint of a two-person game, the nature of different sequences in the quantification vector for the same constraint system determines whether or not the existential player can use information about previous moves of the universal player, when he has to make his move. There are three possibilities:

1. The existential player knows about all the moves of the universal player even before he has to move ($\forall\exists$-QLP).
2. The existential player does not know about the moves of the universal player, even after the variable has been fixed by the universal player ($\exists\forall$-QLP).
3. The existential player knows about each move of the universal player after he has fixed the corresponding variable.

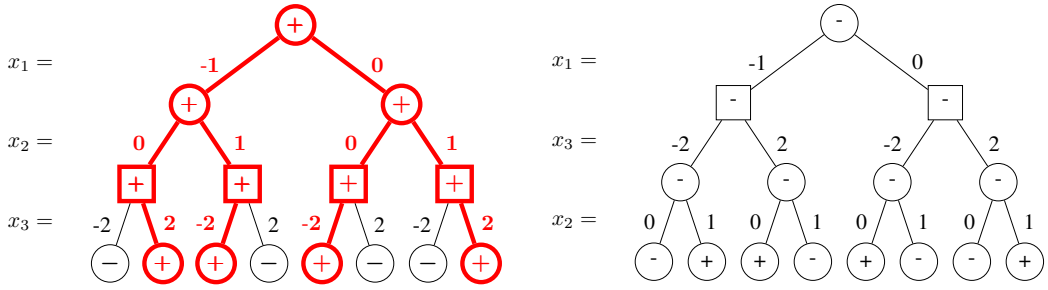
Judging from the perspective of a two-person game, case 1 is the easiest, whereas case 2 is the hardest. In the former we simply have to check whether for any possibly move sequence of the universal player, a corresponding winning move sequence for the existential player does exist. However, these existential move sequences are independent of each other and do not have to be consistent. Opposed to this, in the latter case the existential player must find one single solution that holds for all possible move sequences of the universal player and the only information available is the constraint system and the bounds on the variables of the universal player. Case 3 is the standard QLP with an alternating quantifier sequence of variable blocks.

Example 4. *If we restrict the variables of the three-dimensional QLP*

$$\forall\, x_1 \in [-1, 0]\ \forall\, x_2 \in [0, 1]\ \exists\, x_3 \in [-2, 2] :$$

$$\begin{pmatrix} 10 & -4 & 2 \\ 10 & 4 & -2 \\ -10 & 4 & 1 \\ -10 & -4 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 4 \\ 12 \\ 8 \end{pmatrix}$$

to the integer bounds of their domains, we observe game-tree with a winning strategy for the existential player as shown in Figure 2.3a. For example, if the universal player moves to -1 *and* 0 *(i.e., he sets* $x_1 = -1$ *and* $x_2 = 0$*) the existential player has to move to* 2*. However, if the existential player must choose his move* x_3 *before the universal player in turn chooses* x_2 *as shown in Figure 2.3b, then there is no winning strategy for the existential player at all.*



(a) Winning strategy for $Q = \forall x_1 \forall x_2 \exists x_3$.

(b) No winning strategy for $Q = \forall x_1 \exists x_3 \forall x_2$.

Figure 2.3.: Game-tree for different orders in the quantifier sequence.

In the operations research literature, the term linear programming is used concurrently to describe the problem of deciding whether a set of linear inequalities has a feasible solution (e.g. checking the non-emptyness of a polytope $\mathcal{P}_{LP}$) and the problem of optimizing a linear function over a system of linear inequalities (e.g., finding the optimal extreme point of a polytope $\mathcal{P}_{LP}$). It is well known that the optimization problem and the feasibility problem are polynomially equivalent [197, 161] and in this thesis we treat both cases. In [208] QLPs describe the decision problem only and up to now we also restricted ourselves to the feasibility of QLPs. However, if there is more than one winning strategy for the existential player, it can be reasonable to search for a certain (the "best") one. We can therefore modify the QLP decision problem to include an objective function to improve the quality of the solutions as shown in the following (where we note that transposes are suppressed when they are clear from the context to avoid excessive notation).

Definition 2.15 (QLPs with Objective Function)**.** *Let* $Q \circ x \in [l, u] : Ax \leq b$ *be given as in Definition 2.1 with the variable blocks being denoted by* x^i*. Let there also be a vector of objective coefficients* $c \in \mathbb{Q}^n$*. We call*

$$\begin{gathered} z = \min_{x^1}(c^1x^1 + \max_{x^2}(c^2x^2 + \min_{x^3}(c^3x^3 + \max_{x^4}(\ \ldots \min_{x^m} c^m x^m))))) \\ Q \circ x \in [l, u] : Ax \leq b \end{gathered} \tag{QLP*}$$

a QLP with objective function *(for a minimizing existential player). W.l.o.g we assume* $\mathcal{Q}_1 = \exists$ *and* $\mathcal{Q}_n = \exists$*.*

The semantic of the objective function is at follows: the existential player tries to satisfy all constraints and, subject to this, minimizes an objective function with respect to the maximum possible loss that can result from the universal players possible decisions. The universal player, who sets his variables in an adversarial manner, tries to invalidate at least one constraint, or if this is not possible, he tries to worsen the objective function value.

Like in the unquantified case, we can maximize by minimizing the negative of the objective function. If the quantifier string begins with an universal variable, the objective function does analogously begin with a maximizing instruction for the universal player. Note that the variable vectors $x^1, ..., x^i$ are fixed when a player minimizes or maximizes over variable block B_{i+1}. Consequently, it is a dynamic multi-stage decision process, similar as it is also known from multi-stage stochastic programming (see Section 2.2.2). However, whereas in the latter an expected value is minimized, in our case we try to minimize the possible worst case (maximum loss) scenario that can result from the universal player's decisions, as it is e.g. also done in robust optimization (see Section 2.2.2). For the rest of this thesis we use the abbreviation $\min c^T x$ for the objective function and denote by

$$z_{\mathcal{Q}} = \min\{c^T x \,|\, \mathcal{Q} \circ x \in [l, u] : Ax \leq b\}$$

a *quantified linear program with objective function*.

Definition 2.16 (QLP Optimization Problems)**.** *Given a QLP with objective function, the problem "Is it feasible? If yes, what is the best objective value of the existential player's winning policies?" is called* QLP optimization problem.

The integration of an objective function is more faithful to the definition of a conventional linear program, since the quantified linear decision problem version could also be called a *quantified constraint satisfaction problem*. For the rest of this thesis, we therefore use the term QLP interchangeably to identify QLPs with and without an objective function (e.g., QLPs with an arbitrary constant objective function). The objective function value resulting from a single game or playout is called its *outcome*. The objective function value of a policy is the worst outcome of all matches resulting from this policy. Thus, the optimal value of a QLP or QIP minimization problem, is the value of the best winning policy or winning strategy respectively.

Note that to find the existential player's optimal objective value it is not enough to fix the universal players variables to their worst-case values regarding the objective function. For clarification, consider the following program:

$$\min\{-x_1 - 2x_2 \,|\, \forall\, x_1 \in [0, 1]\; \exists\, x_2 \in [0, 1] : x_1 + x_2 \leq 1\}$$

Judging from the objective function the universal player should fix $x_1 = 0$, but this results in a better objective value for the existential player than forcing the existential player to fix $x_2 = 0$ in the constraint system.

Given a feasible QIP with objective function, a winning strategy $\mathcal{S}$ can be evaluated following the *minimax principle*, which is a decision rule used e.g., in decision theory, game theory, and statistics for minimizing the possible loss for a worst case scenario. The *minimax value* of a winning-strategy $\mathcal{S}$ can be recursively defined as follows:

Definition 2.17 (Minimax Value)**.** *Let $\mathcal{S} = (V, E, L')$ a winning strategy and $v \in V$ a node of S. $L(\mathcal{S})$ denotes the set of leafs of S. The function $minimax : V \to \mathbb{R}$ is recursively defined as:*

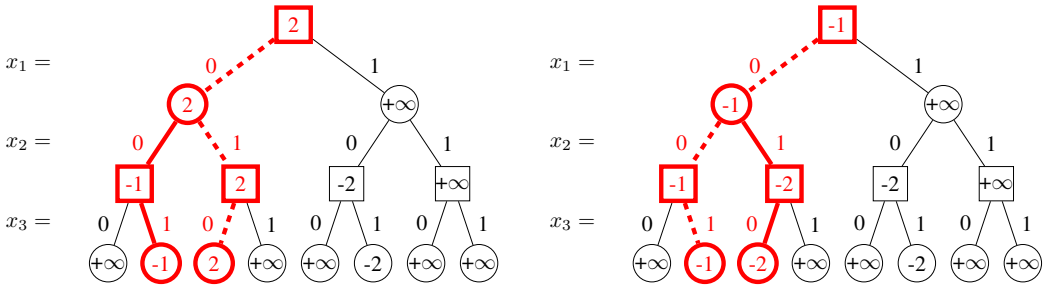
$$minimax(v) = \begin{cases} f(v) & \text{if } v \in L(\mathcal{S}) \\ \min\{minimax(v') | (v, v') \in E\} & \text{if } v \notin L(\mathcal{S}) \text{ and } v \in V_\exists \\ \max\{minimax(v') | (v, v') \in E\} & \text{if } v \notin L(\mathcal{S}) \text{ and } v \in V_\forall \, . \end{cases}$$

The function $f(v)$ is the so-called *weighting function* and is used to evaluate the outcome of the final position x using the rules of the game and represents how good or bad the sequence of moves from the root to the leaf is for the existential player. For QIPs it can be defined as follows:

Definition 2.18 (Weighting Function)**.** *The value $f(x)$ of a move sequence x computes as:*

$$f(x) = \begin{cases} c^T x & \text{if } x \text{ is winning for the existential player} \\ +\infty & \text{if } x \text{ is winning the the universal player.} \end{cases}$$

Figure 2.4 shows the evaluation of two winning strategies for the QIP from Example 3, the only difference is the objective function. Figure 2.4a shows the evaluation of the winning strategy for the objective function $-x_0 + 2x_1 - x_2$ is -1 resulting in a minimax value of 2, whereas the minimax value for the objective function $-x_0 - 2x_1 - x_2$ is -1 as shown in Figure 2.4b. Obviously, the minimax value of a winning strategy is equal to the maximum value at the leafs of the strategy (c.f. [171]). Note that the minimax value of a winning strategy is an upper bound, depending on the moves of the universally player. If the latter does not choose its moves perfectly according to the winning strategy of the existential player, the value of the objective function might be better eventually.



(a) $z = -x_0 + 2x_1 - x_2$. (b) $z = -x_0 - 2x_1 - x_2$.

Figure 2.4.: Winning strategies for different objective functions.

When solving a LP with objective function, one of the three possibilities can happen:

1. The model is *infeasible*, i.e., there are no values for the variables that satisfy all constraints simultaneously.

2. The model has a finite *feasible* solution, i.e., there exist a set of values for the variables which give a finite optimal to the objective function.

3. The model has an *unbounded* solution, i.e., the objective value can be decreased (or increased for a maximization problem) without limit by choosing values for the variables.

Note that unboundedness can not occur in the context of QLPs and its integer variants, since all decision variables are bounded by Definition 2.1.

2.2. Related Work

The purpose of this section is to provide a detailed overview of related work in the context of quantified linear and integer programming. We first review existing literature regarding the quantification of LPs and (M)IPs. Afterwards we discuss and contrast the classical solution approaches for optimization under uncertainty, stochastic programming and robust optimization. We complete this section with a detailed survey of primary research in the context of variable and constraint quantification.

2.2.1. Literature Review

In [194] a QLP formulation was used to model problems from the domain of real-time scheduling [57], with the goal to reduce the inflexibility of static scheduling in hard real-time systems. One of the fundamental features of real-time systems is the uncertain nature of execution times of tasks and the presence of complex timing constraints among them. In traditional models variable execution times are modeled through fixed worst-case values and relationships are limited to those that can be represented by precedence graphs [206]. Using QLPs as modeling tool allows to schedule tasks with varying execution times, represented by universally quantified variables, and complex timing constraints among tasks described by linear constraints (see also Section 2.3).

In this context, a first polynomial time algorithm to solve QLPs based on quantifier elimination techniques was proposed in [108] for a restricted class of constraints [194], which are a subset of totally unimodular constraint systems [197]. For arbitrary linear constraints the worst-case time complexity of the proposed algorithm is double exponential [197, 125]. Subramani proposed a framework to formalize scheduling problems in real-time systems using QLPs in [207]. There he focussed on the variability in the execution times, complex relationships between tasks using arbitrary linear constraints and different scheduling schemes – static ($\exists\forall$-QLPs), co-static ($\forall\exists$-QLPs) or parametric (alternating quantifier changes) – depending on the information available when dispatching. A detailed introduction to QLPs and methodologies to solve them are available in [209, 147, 88]. QIPs were studied in [210], where a number of special cases where analyzed from the perspective of computational complexity. In that paper the authors discussed conditions under which QIPs can be relaxed to QLPs without altering the solution space.

Research in this context is also supported by the *Deutsche Forschungsgemeinschaft (DFG)*[1]. The DFG founded project *"Erweiterung mathemtische Optimierungsmethoden zur Lösung* **PSPACE***-vollständiger Probleme mit Hilfe quantifizierter linearer Programme (Lo 1396/2-1)"*[2] at TU Darmstadt deals with the development of algorithms to solve QIPs based on extended LP- and QLP-techniques, which are the subject of this thesis. As already mentioned in the introduction, the DFG founded Collaborative Research Centre (SFB) 805 *"Sonderforschungsbereich 805 - Beherrschung von Unsicherheit in lasttragenden Systemen des Maschinenbaus"* at TU Darmstadt deals with the control of uncertainty in mechanical systems. In this context QMIPs are used to model and optimize process chains under uncertainty. Also in the context of game modeling QIPs can be applied. In [89] the **PSPACE**-complete two-person game *Gomoku* was modeled with the help of an QIP formulation, in [148] the model was adopted to the **PSPACE**-complete two-person game *Connect-6* resulting in a 0/1-QIP that was solved with an extended variant of the algorithm proposed in [88].

Recently, extensions of traditional QLPs were presented in [91], where the standard quantification was extended to implications of quantified linear systems, called *quantified linear implications* (QLI). QLPs and QLIs that arise when universally quantified variables are unbounded were studied in [185]. Whereas in the above citations the focus was on analyzing various special cases of the general problem, with a view on subclasses that are tractable, the focus of this thesis lies on general QLPs with two or multiple stages without any restrictions to the quantification sequence or the constraint system. Furthermore, our focus lies on QLP optimization problems, whereas the above citations consider quantified decision problems in general.

2.2.2. Optimization under Uncertainty: State of the Art

Optimization under uncertainty has experienced rapid development in both theory and practice from the very beginning in the 1950s, starting with the seminal works of Beale [15], Dantzig [72], Bellman [16] as well as Charnes and Cooper [65]. The particular importance of optimization under uncertainty was also demonstrated in the seminal paper by Ben-Tal and Nemirovski [23], where they showed in a detailed computational study that even a small perturbation of the problem data can make the nominal optimal solution completely meaningless from a practical viewpoint. We quote from their case study:

"In real-world applications of linear programming, one cannot ignore the possibility that a small uncertainty in the data can make the usual optimal solution completely meaningless from a practical viewpoint."

Due to limited computational capabilities in the past, decision models often replaced those uncertainties with averages or best estimates but recent computational advantages greatly expanded the range of optimization techniques under uncertainty. Today *stochastic programming* and *robust optimization* – two fundamentally different approaches that address optimization with data uncertainty – are the most prominent modeling paradigms and presently the subject of intense research in this field. The main difference lies in the type

[1] `http://www.dfg.de`

[2] `http://gepris.dfg.de/gepris/projekt/194664946`

of uncertainty that is handled by the two methodologies. Stochastic programming assumes the uncertainty to follow a probability distribution, while robust optimization in contrast is concerned with models that contain deterministic set-based uncertainties [36]. In the following we give a brief introduction to stochastic programming and robust optimization and compare them with quantified linear programming.

Stochastic Programming

The approach for uncertainty quantification used in stochastic programming (see, e.g. [128, 174, 47, 189] and the references therein) is the representation of random parameters in the input data (c, A, b) by random variables. This approach has a long and active history dating at least as far back as Dantzigs original paper [72] and draws upon a variety of traditional operations research techniques. During the last four decades a vast quantity of literature on the subject has appeared (see, e.g. [218, 217] for a stochastic programming bibliography maintained by Maarten van der Vlerk). A variety of stochastic programming applications can be found in [224]. The underlying optimization problem can be a linear program, a (mixed) integer program, but also a nonlinear program. Usually, stochastic programming models are furthermore subdivided in *recourse models* and *chance constraint models*.

Recourse Models The more important and widely studied case, where most of the applications reported in the literature belong to and which has strong similarities to QLPs, is that of a *stochastic linear program with recourse*, which first appeared in Beale [15] and Dantzig [72]. These problems arise when some of the decision variables must be taken before the outcomes of some (or all) random events are known, and some after they become known. Recourse variables represent decisions that can be made on a "wait and see" basis, after the uncertainty is resolved, and they allow to adapt a solution to the specific outcome observed. Stochastic programming problems with recourse can be essentially divided into two-stage and multi-stage problems (TSSLPs and MSSLPs). In the former, a set of initial decisions are taken first, followed by a random event. After this, recourse decisions, which are based on this event, are taken. The multi-stage problem, accordingly consists of multiple stages, with a random event occurring between each stage.

The classical two-stage linear program with fixed recourse (originated by Dantzig [72] and Beale [15]) is defined as follows:

Definition 2.19 (Two-stage stochastic linear program (TSSLP))**.**

$$\begin{aligned} \min_{x} \;& c^T x + E_\xi \left(\min_{y(\omega)} q(\omega)^T y(\omega) \right) \\ s.t.\;& Ax \leq b \\ & T(\omega)x + W(\omega)y(\omega) \leq h(\omega) \\ & x \geq 0,\, y(\omega) \geq 0, \end{aligned} \tag{TSSLP}$$

where A and b define the deterministic part of the first-stage decisions x. The second stage depends on the realization of a random event $\omega \in \Omega$. $T(\omega)$, $W(\omega)$ and $h(\omega)$ de-

fine the stochastic linear constraints linking the recourse decisions $y(\omega)$ to the first stage decisions. The objective function comprises the deterministic first-stage term $c^T x$ and the expectation of the second-stage objective $q(\omega)^T y(\omega)$ taken over all realization of the random event ω. The random N-Vector $\xi(\omega)^T = (q(\omega)^T, h(\omega)^T, T(\omega)^T, W(\omega)^T)$ contains the stochastic components of the second stage. After the first-stage decisions x have been made and after the random event ω has been realized, the optimal second stage decision can be computed by solving the remaining recourse subproblem

$$\begin{aligned} Q(x, \xi(\omega)) = \min_{y(\omega)} \; & q(\omega)^T y(\omega) \\ s.t. \; & W(\omega) y(\omega) \leq h(\omega) - T(\omega) x \\ & y(\omega) \geq 0. \end{aligned}$$

Thus, the expected second-stage value function can be defined as

$$\mathcal{Q}(x) = E_\xi[Q(x, \xi(\omega))],$$

and the TSSLP can be rewritten as

$$\begin{aligned} \min_{x} \; & c^T x + \mathcal{Q}(x) \\ s.t. \; & Ax \leq b \\ & x \geq 0. \end{aligned}$$

In many practical cases, there is a specific structure in the recourse subproblems, which can be exploited for computational advantage. The most common types are *simple recourse*, *fixed recourse* and *complete recourse*. A simple recourse model arises, when the recourse Matrix $W(\omega)$ of the second stage forms an identity matrix I. In a fixed recourse model, the recourse matrix of the second stage subproblems is not subject to uncertainty (the coefficients are fixed as in a traditional LP). Complete recourse is a property of the recourse subproblem that results, if the subproblem is feasible for any first stage variable allocation x over all realization of the random event ω (e.g. by adding artificial variables to the subproblems that guarantee feasibility, but penalize deviations in the objective function). In most cases fixed recourse subproblems are used.

Apart from the different recourse models, many practical problems not only involve a single observation and reaction, but a sequence of them. Accordingly, the classical form of a multi-stage stochastic linear program with fixed recourse (cf. [47]) can be defined as follows

Definition 2.20 (Multi-stage stochastic linear program (MSSLP))**.**

$$\begin{aligned}
\min_{x^1}\ & c^1x^1 + E_{\xi^2}\left(\min_{x^2(\omega)} c^2(\omega)x^2(\omega^2) + \cdots + E_{\xi^H}\left(\min_{x^H(\omega)} c^H(\omega)x^H(\omega^H)\right)\right) \\
s.t.\ & W^1x^1 = h^1 \\
& T^1(\omega)x^1 + W^2x^2(\omega^2) = h^2(\omega) \\
& \cdots \\
& T^{H-1}(\omega)x^{H-1}(\omega^{H-1}) + W^H + x^H(\omega^H) = h^H(\omega) \\
& x^1 \geq 0,\ x^t(\omega^t) \geq 0,\ t = 2, ..., H,
\end{aligned} \tag{MSSLP}$$

where $c^1 \in \mathbb{R}^{n_1}$, $h^1 \in \mathbb{R}^{m_1}$ and the fixed recourse matrix $W^t \in \mathbb{R}^{m_t \times n_t}$ are known. $\xi^t(\omega)^T = (c^t(\omega)^T, h^t(\omega)^T, T_1^{t-1}(\omega), ..., T_{m_t}^{t-1}(\omega))$ is a random N^t-Vector defined on the probability space (Ω, Σ^t, P)(where$\Sigma^t \subset \Sigma^{t+1}$) for all $t = 2, .., H$. The decisions $x^t(\omega^t)$ depend on the history up to time t, which is indicated by ω^t. A multi-stage stochastic integer problem with only discrete probabilities is also called a game against nature [164]. As for the TSSLP, we can use a recursive node-based definition for the MSSLP as follows

$$\begin{aligned}
\min_{x^1}\ & c^1x^1 + \mathcal{Q}^2(x^1) \\
s.t.\ & W^1x^1 = h^1 \\
& x^1 \geq 0,
\end{aligned}$$

and

$$\mathcal{Q}^{t+1}(x^t) = E_{\xi^{t+1}}[Q^{t+1}(x^t, \xi^{t+1}(\omega))] \text{ for } t = 2, \ldots, H-1,$$

where x^t indicates the current state of the system (all variable assignments up to stage t), with

$$\begin{aligned}
Q^t(x^{t-1}, \xi^t(\omega)) = \min_{x^t(\omega)}\ & c^t(\omega)x^t(\omega) + \mathcal{Q}^{t+1}(x^t) \\
s.t.\ & W^tx^t(\omega) = h^t(\omega) - T^{t-1}(\omega)x^{t-1} \\
& x^t(\omega) \geq 0,
\end{aligned}$$

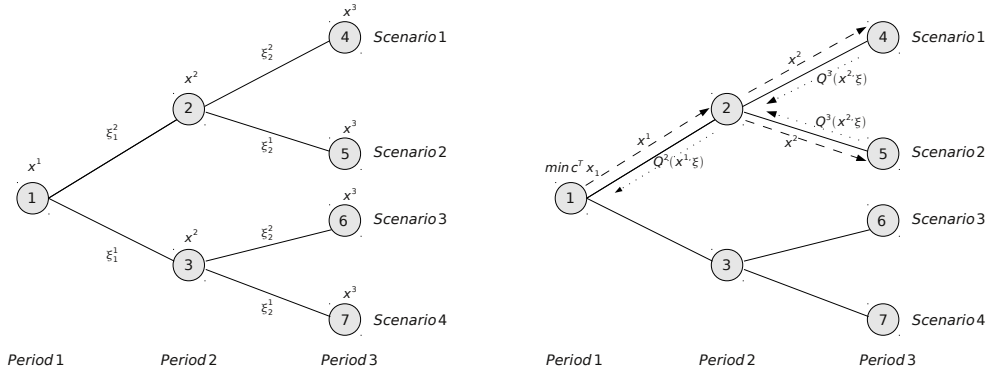
and finally, for terminal conditions, we have

$$\begin{aligned}
Q^H(x^{H-1}, \xi^H(\omega)) = \min_{x^H(\omega)}\ & c^H(\omega)x^H(\omega) \\
s.t.\ & W^Hx^H(\omega) = h^H(\omega) - T^{H-1}(\omega)x^{H-1} \\
& x^H(\omega) \geq 0.
\end{aligned}$$

This behavior is very similar to QLPs, but whereas in a TSSLP or MSSLP a random event occurs between each decision stage, which affects the coefficients of the constraint system and the objective function, an adversarial opponent assigns the values of universal

variables in a QLP.

The main challenge in designing algorithms for stochastic programming problems arises from the need to calculate conditional expectations associated with random variables, and even for small problems approximations must be used. A common strategy to solve a stochastic program is an approximation that assumes a finite time horizon and a discrete probabilistic representation of uncertainty. Thus, the uncertainty can be represented through a so-called *scenario-tree* as depicted in Figure 2.5, which defines the possible sequence of realizations over all time-stages. Nodes in the scenario-tree at stage t are decision points where variable allocations $x^t(\omega)$ must be determined, with respect to all realized data $(\xi^1 \dots \xi^{t-1})$ of the last $t-1$ stages. Arcs of the tree represent realizations of the random variables. The root node is associated with the first stage decision variables and a path from the root to a leaf is called a *scenario*.



(a) The scenario-tree of a MSSLP [147].

(b) Solving the scenario-tree [147].

Figure 2.5.: Scenario-tree with three stages.

Such a tree formulation can be used to transform the recursive definition into a so-called deterministic equivalent program (DEP) (see e.g. [229, 47]), which can be solved with the help of traditional linear programming algorithms. However, SP formulations can lead to very large scale DEPs, since their size grows exponential with the size of the input. Thus, methods based on decomposition [188] and approximation [221] become paramount. In [74] and [123] algorithms based on the Dantzig-Wolfe decomposition principle [71] (see also Section 5.1.1) were proposed as a possible solution method. The L-Shaped method, which basically is an adaption of Benders' Decomposition [25] (see also Section 5.1.2) to two-stage recourse problems was proposed in [219]. It was also successfully extended to multi-stage problems with recourse [45, 47], where the L-shaped method is used in a nested application.

The (nested) L-shaped method and its various variants are currently the most prominent solution approaches for TSSLP and MSSLPs, and their algorithmic improvement over the last decades is highly relevant and ongoing research. Solving a MSSLP with this method can be interpreted as solving a tree of linear programs (the scenario-tree) as depicted in Figure 2.5b. The scenario-tree is therefore traversed multiple times and information are

passed among nodes of the tree. Starting from the root problem at stage $t = 1$, all subproblems in the tree are examined in sequence and solved either to optimality or until they have been detected to be infeasible. Information are passed along the tree in two directions: primal information (proposals for the variables) are passed down the tree used to create the variable right-hand sides, dual information are passed back up the tree in order to generate benders *feasibility* and *optimality cuts* (see also Section 5.1.2 for more details).

Several improvements to the original algorithm were proposed over the last decades. These include various tree traversing strategies [231], a multi-cut version [48] and serious progress was made by parallelizing the solution process [80, 46] exploiting the natural independence of subtrees. The so-called regularized L-shaped method was proposed in [186], and an enhanced version was presented in [190]. This method adds a quadratic term to the objective of each subproblem to keep solutions of successive iterations closer together in order to stabilize the progress of the algorithm. In [239] the regularized L-shaped method was compared to the level decomposition method developed in [94]. They report that the regularized version outperforms the non-regularized version in many instances of their test set. Recent achievements include a variant of the L-shaped method with aggregated cuts [216], an approach to generate tighter feasibility cuts was proposed in [7]. In [232] a cut consolidation technique to remove unused constraints and a hybrid sequencing protocol were presented in a detailed computational study.

However, all these methods suffer from the fact that TSSLP and MSSLP formulations can result in models with a very large number of scenarios, making it impossible to include all of them in the solution process. To handle this problem a Monte Carlo simulation based approach was proposed in [134, 221], called sample average approximation (SAA) method. The basic idea of the SAA-method is that a random sample is generated and consequently the expected value function is approximated by the corresponding sample average function. The obtained sample average optimization problem is solved and the procedure is repeated several times, until a stopping criterion is satisfied. For the two-stage case the proposed algorithms work very well, in [221] problems with up to 2^{1694} scenarios were solved to within an estimated 1% optimality. Nevertheless, it was argued that when being applied to MSSLPs, the complexity of the SAA-method grows fast with an increasing number of stages and the problems become computationally unmanageable [200]. Another critical aspect of this approach (and many other statistical approximation schemes used in SP) is that in the absence of *relatively complete recourse*, the solution obtained by an approximation can be actually infeasible, because some critical scenarios might not have been obtained in the sample. Recently, also hybrid algorithms where Benders Decomposition and Genetic Algorithms are combined were proposed [203, 173, 139].

Chance Constraints Model The essence of recourse models is that infeasibilities in the second or higher stages are allowed and can be compensated at a certain penalty. In chance constraint models, developed by Charnes and Cooper [65], the focus lies on the reliability of the system in contrast. This reliability is expressed as a minimum requirement

on the probability of satisfying constraints stated as

$$\mathbb{P}(A^i_\omega x \leq h^i_\omega) \geq \alpha^i,$$

where $0 < \alpha^i < 1$ and $i = 1, \ldots, I$ is the index set of constraints that must hold jointly. The resulting program is called a *chance-constrained* stochastic program. Under certain assumptions on the probability distribution (like e.g. Normal, Cauchy, Uniform), chance-constraints can be converted into a deterministic equivalent form and then be treated as in a traditional recourse problem as mentioned above. More details on chance-constrained stochastic programming can be found in [174].

Over the last 20 years, also stochastic programming with integer or binary recourse variables has received tremendous research attention in both application and algorithmic aspects. However, compared to the continuous case of a TSSLP or MSSLP, relatively little is known on theory and algorithms for linear mixed-integer stochastic programming in general. Given that duality results do not hold in general IP and MIP formulations, it is also difficult to extend current algorithms for the linear case in order to generate cutting planes, except when only the first stage contains integer variables or other special cases. A survey of developments can be found in [198, 199], a stochastic integer programming bibliography with publications from the years 1996-2007 can be found at [217].

Compared to stochastic programming models, a QLP can be interpreted as a worst-case MSSLP with fixed recourse and only the right-hand side being affected by uncertainties. The main advantage of quantified linear and integer programming is that uncertainty can be easily modeled without detailed knowledge of the underling probability distribution. Furthermore, while SP models minimize the expected value of an objective function with respect to a set of scenarios, QLPs minimize an objective function with respect to the possible worst case (maximum loss). In Section 4.1.3 we propose a recursive node-based definition for QLPs, similar as for MSSLPs, and show how a DEP of a QLP can be constructed. In Section 5.2 we propose an decomposition based algorithm to solve the corresponding DEP.

Robust Optimization

In contrast to stochastic programming, robust optimization is a more recent approach in the context of optimization under uncertainty, in which the uncertainty model is not stochastic, but rather deterministic and set-based. Instead of seeking to immunize the solution in some probabilistic sense to stochastic uncertainty, here the decision-maker constructs a solution that is valid for any realization of uncertainty in a given set [36]. While applications of stochastic programming have been reported over many years in the literature, robust optimization appeared recently, with most research in the past ten years (see, e.g. [24, 38, 42, 18, 36, 215] and the references therein). The roots can be found in the field of robust control and in the work of Soyster [204]. However, high interest in both theoretical aspects and practical applications started in the first place with the work of Ben-Tal and Nemirovski [20, 21], and El-Ghaoui et al [109, 110] in the late 1990s.

The idea of robust optimization is to define a so-called *uncertanty set* $\mathcal{U}$ for the uncertain

parameters in the input data (c, A, b) and then to require that the constraint system should hold for *any* possible realization of the data from $\mathcal{U}$. The optimization problem modeling this requirement is called the *robust counterpart problem* (RCP), an optimal solution of the RCP is called robust optimal solution. The robust optimization methodology can be applied to any optimization problem (e.g. quadratically constrained quadratic programs (QCQPs) or semidefinite programs (SDPs)), in which the uncertain data is known to lie in a given uncertainty set. Though it is still a relatively new approach, it has already proved very useful in many applications. In [18] a list of problem classes for which robust optimization is applicable is listed.

One might imagine that the addition of robustness to a general optimization problem comes at the expense of a significantly increased computational complexity, and indeed, the RCP of an arbitrary convex optimization problem is in general intractable [18]. Nevertheless, depending on the structure of the underlying optimization problem and the uncertainty set $\mathcal{U}$, the corresponding RCP can be solved or at least be approximated in polynomial time for many application cases [21]. Ben-Tal and Nemirovski [22] showed that the RCP of a linear optimization problem is essentially always tractable for many practical uncertainty sets. Consider, for example, a linear program

$$\min\{c^T x \mid Ax \leq b,\, x \in \mathbb{R}^n\} \tag{2.3}$$

with uncertain parameters (c, A, b). The corresponding RCP can be written as follows:

Definition 2.21. *(Robust Counterpart (RCP))*
Given a linear program as in Definition 1.10, the system

$$\min\{c^T x \mid Ax \leq b,\, \forall (c, A, b) \in \mathcal{U}\}, \tag{2.4}$$

is the corresponding robust counterpart problem (RCP) and $\mathcal{U}$ is a given uncertainty set for the uncertain parameters (c, A, b).

Due to the results in [18], in the case of robust *linear* optimization, the objective and the right-hand side of the constraints can be assumed to be deterministic and w.l.o.g. constraint-wise uncertainty can be assumed. It hence suffices to focus on single constraints of the form

$$a_i^T x \leq b_i, \forall a_i \in \mathcal{U}_i, \tag{2.5}$$

and this constraint holds if and only if $\max_{a_i \in \mathcal{U}} a_i^T x \leq b_i \forall i$. These are referred as the subproblems which must be solved in order to solve the entire RCP. Of course, a major modeling decision concerns the choice of the uncertainty set $\mathcal{U}$ and the resulting RCP might not longer be an LP [17]. In particular, the RCP of an LP with a polyhedral uncertainty set becomes a linear programming problem, while the RCP of an LP with an ellipsoidal uncertainty set becomes a second-order cone problem respectively [24]. Other interesting results have been achieved recently, e.g. under very natural assumptions, robust LPs can be formulated as semi-definite programming problems and thus solved by a polynomial time approximation scheme [21, 18]. Several attempts to extend the ideas to discrete robust optimization have been e.g. made in [37, 40, 39]. However, unlike the case of continuous

robust optimization, the conditions on the uncertainty sets are rather restrictive to still ensure the efficient solvability of the RCP as shown in [39]. Therefore, approximation techniques based on piecewise linearization are currently the means of choice for such problems [42].

While stochastic programming problems can result in large deterministic linear models when considering many scenarios, the RCP models grow only slightly when uncertainty is added in general and therefore, they can often be solved efficiently. Another crucial difference is that in the stochastic programming approach constraints may be violated with a certain probability (chance-constraints) or at a given penalty (recourse model), whereas robust optimization is associated with *hard constraints*, which must be satisfied whatever the realization of the data (c, A, b) in the uncertainty set $\mathcal{U}$ is [22]. Thus, in many cases a single-stage robust optimization model tends to be to conservative [11], especially in situations where some recourse decisions can be made after the uncertainty is revealed. To address this issue, two-stage models – also called *robust adjustable* or *adaptable optimization* – were proposed e.g. in [19]. However, two-stage problems are very challenging to compute, even formulations with LP problems in both stages can be **NP**-hard [19]. Nevertheless, in many real-world problems, decisions and uncertainty are often repeatedly interwoven over time and RO model are not capable to handle this efficiently due to the difficulty in incorporating multiple stages [215].

From the viewpoint of quantified linear and integer programming, robust optimization problems can be viewed as QLPs with one quantifier change ($\exists\forall$), whereas in QLPs multiple quantifier changes occur in general. However, from the viewpoint of robust optimization, where arbitrary uncertainty sets may appear (though not necessarily being tractable at all), the uncertainty set of QLPs is a simple hyperrectangle.

Other LP-based solution techniques

Apart from stochastic programming and robust optimization as mentioned above, typical solution approaches for problems that are affected by uncertainty are sensitivity analysis [179], LP-based approximation techniques [155], dynamic programming [16], and the exploration of Markov-chains [237] for example. Sensitivity analysis is a simple classical method of addressing uncertainty, which is typically applied as a post-optimization tool for quantifying the goodness or robustness of a solution, e.g. the change in cost for small perturbations in the underlying problem data. It is not particularly helpful for generating solutions that are robust to data changes in the sense of an a priori ensured feasibility when the problem parameters vary within the prescribed uncertainty set [234]. A further possibility to deal with these uncertainties is an approximation where a given probability distribution is aggregated to a single estimated number. Then, the optimum concerning these estimated input data can be computed with the help of traditional optimization tools. In some fields of application, as e.g. the fleet assignment problem of airlines, this procedure was successfully established [117]. In other fields, like production planning and control, this technique could not be successfully applied, although mathematical models do exist [172]. Dynamic programming was developed by Richard Bellman in the early1950s [16] and is a computational method to deal with multi-stage decision processes in a dynamic

environment. In fact, decision-making under uncertainty was the original and intended application of this technique [164]. Dynamic programming usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time. It starts solving all subproblems at the final stage and then uses the solution to recursively compute solutions for higher stages until the original problem at the root is solved eventually. For more information on solution approaches in the context of optimization under uncertainty we refer the reader to [193, 41] and the references therein.

2.2.3. Quantification of Variables and Constraints

The explicit quantification of variables and constraints allows to express many problems that cannot be modeled using classical modeling paradigms. Quantifiers are a powerful tools to model *uncertainty* or *adversarial* situations. For example, if a decision is based on a value that is not exactly known in advance, one might look for *robust* solutions that are valid *for all* possible values that might occur. As the efforts of extending languages with quantifiers have not only been made for linear programs in terms of QLPs and QIPs, we give an overview in the following, where we briefly sketch the achievements from other fields in this context.

Boolean Satisfiablility (SAT) and Constraint Satisfaction (CSP)

Many real-world problems are combinatorial search problems that can be represented in terms of decision variables and constraints. Besides integer programming (IP), boolean satisfiability problems (SAT) and constraint satisfaction problems (CSP) are very successful frameworks that are used to model and solve these problems.

The SAT problem is one of the most important and extensively studied problems in computer science. Given a propositional boolean formula, the SAT problem asks for an assignment of variables such that the formula evaluates to true, or a proof that no such assignment exists [10, 44].

Definition 2.22 (Boolean Satisfiability Problem (SAT))**.**
Let $x_1, \dots, x_n$ be boolean variables, and let $\mathcal{C} = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a logic formula in conjunctive normal form (CNF). Each clause $C_i = l_1^i \vee \cdots \vee l_{k_i}^i$ is a disjunction of literals. A literal $l \in L = \{x_1, \dots, x_n, \overline{x}_1, \dots, \overline{x}_n\}$ is either a variable x_j or the corresponding negation $\overline{x}_j$. The task of the boolean satisfiability problem (SAT) is to either find an assignment $\overline{x} \in \{0,1\}^n$ such that the formula $\mathcal{C}$ is satisfied, or the conclusion that $\mathcal{C}$ is unsatisfiable. That is each clause c_i evaluates to 1, or for all $x \in \{0,1\}$ at least one clause C_i evaluates to 0.

SAT is one of the classic problems in computer science, it is of theoretical interest because it is the canonical **NP**-complete problem [70] (cf. Section 4.2 for details). Apart from its high theoretical relevance, SAT has many practical applications as e.g. model checking, design verification, or AI planning [152, 68].

IP generalizes the standard SAT problem, by allowing the possible values for the variables to be chosen from an arbitrary finite set, and allowing the constraints to be arbitrary

linear constraints rather than just clauses. Whereas the solution approaches for IPs often rely on LP-relaxations combined with branch-and-cut procedures to find optimal integer solutions, the core of most modern SAT solvers is the *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm [78, 77], which is essentially a depth-first-search with some pruning techniques like unit propagation [235], conflict-driven clause learning [153], and backtracking [150].

The constraint satisfaction problem (CSP) also generalizes the standard SAT problem, by allowing the possible values for the variables to be chosen from an arbitrary finite set. Furthermore, the constraints are allowed to to be arbitrary constraints rather than just clauses or linear constraints and thus, CSP also generalizes IP. The CSP problem consists of a set of variables, each with a finite domain of values and a set of constraints on subsets of these variables.

Definition 2.23 (Constraint Satisfaction Problem (CSP))**.**
A constraint satisfaction problem (CSP) is a 3-tuple $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X}$ is a finite set of variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, $\mathcal{D}$ is a set of finite domains $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ where the domain $D(x_i)$ is the finite set of values that variable x_i can take, and a set of constraints $\mathcal{C} = \{c_1, \ldots, c_m\}$. Each constraint c_i is defined by the ordered set $var(c_i)$ of the variables it involves, and a set $sol(c_i)$ of allowed combinations of values. An assignment of values to the variables in $var(c_i)$ satisfies c_i if it belongs to $sol(c_i)$. A solution to a CSP is an assignment of a value from its domain to each variable such that every constraint in C is satisfied. Note that there are no further restrictions imposed on the constraints.

CSPs provide a general framework to express a wide variety of combinatorial search problems and they play an important role in many areas of computer science and artificial intelligence [52]. They also find their application in the Operations Research community in the context of Location Planning, Scheduling, Car Sequencing, Cutting Stock Problems, Vehicle Routing, Timetabeling, Rostering and many more [54]. SAT, CSP and the IP feasibility problem are three different paradigms to model and solve combinatorial search problems, and as each of the problem classes is **NP**-complete, they can be reduced among each others in polynomial time [225, 32, 238]. However, each paradigm has its strengths and weaknesses. Due to their complementary strengths, there is an increasing belief that hybrid methods may often perform better than pure methods. An approach to integrate techniques from constraint programming and (mixed-integer) programming in order to exploit the strengths of both fields, is the constraint integer programming framework SCIP [2, 4]. However, there are classes of problems containing uncertainty that cannot be expressed within these frameworks. QCSPs and QSAT, which are the quantified extensions of CSP and SAT and allow for universally quantified variables, make it possible to model such problems that contain uncertainty. As a result, these extensions have been attracting significant interest in recent years. In the following, we highlight the key ideas of both approaches and specify their basic solution strategies.

Quantified Boolean Satisfiability (QSAT)

A quantified boolean formula is a propositional boolean formula with existential and universal quantifiers preceding it. Given a quantified boolean formula, the question whether it is satisfiable is called the quantified boolean satisfiability problem (QSAT).

Definition 2.24 (Quantified Boolean Satisfiability Problem (QSAT))**.**
Let $x_1, \ldots, x_n$ be boolean variables, and let $\mathcal{C} = C_1 \wedge \cdots \wedge C_m$ be a logic formula in conjunctive normal form (CNF) as in Definition 2.22. A satisfiability problem of the form:

$$Q_1 x_1 \; Q_2 x_2 \; \ldots Q_n x_n \; \mathcal{C}$$

where $\mathcal{Q}_i \in \{\forall, \exists\}$ is called a quantified boolean satisfiability (QSAT) problem.

QSAT is a natural paradigm for characterizing the complexity class **PSPACE** (see Section 4.2), but apart from being an important theoretical problem by itself, QSAT problems have many relevant application domains because they allow the compact encoding of problems like bounded model checking [81], formal verification [30], non-monotonic reasoning [8], two-player game solving [105] and AI planning [12].

Deciding the satisfiability of a quantified boolean formula is an important research issue and engaged considerable interest in last two decades (see e.g. [60, 106, 90, 114, 115, 58] and the references therein), and development still improves rapidly. Search-based solvers are mainly based on the DPLL algorithm adapted to the quantified case [60, 236, 145, 112]. Typical techniques being applied are variants of unit propagation [61], the pure literal rule [61], and backtracking [115]. Other solvers tackle the problem by expanding out quantifiers until a single quantifier type is left, then the resulting problem is handed to a SAT solver [12, 43, 144, 127]. The interest in QSAT is also witnessed by a number of QSAT test sets [113] and by the presence of an annual competition of QSAT solvers (QBFEVAL) [160, 169]. As in the case of SAT and IP, QIP generalizes the standard QSAT problem, by allowing the possible values for the variables to be chosen from an arbitrary finite set, and allowing the constraints to be arbitrary linear constraints rather than just clauses.

Quantified Constraint Satisfaction (QCSP)

The quantified constraint satisfaction problem (QCSP) is a generalization from the purely existential CSP where some of the variables become universally quantified [53, 151, 104].

Definition 2.25 (Quantified Constraint Satisfaction Problem (QCSP))**.**
A quantified constraint satisfaction problem (QCSP) has the form

$$Q_1 x_1 \in D(x_1) \ldots Q_n x_n \in D(x_n) \; \mathcal{C} \qquad \text{(QCSP)}$$

with $\mathcal{X}, \mathcal{D}, \mathcal{C}$ given as in Definition 2.23. Q is a sequence of quantifiers over $x_1, \ldots, x_n$ where each Q_i with $1 \leq i \leq$) is either an existential, $\exists$, or a universal, $\forall$, quantifier. The expression $\exists x_i c$ means that "there exists a value $a \in D(x_i)$ such that the assignment

(x_i, a) satisfies c". Similarly, the expression $\forall x_i c$ means that "for every value $a \in D(x_i)$, the assignment (x_i, a) satisfies c".

As in the case of SAT and QSAT, quantification increases the expressiveness of the framework, but at the same time the complexity also increases. While the classical CSP feasibility problem is known to be **NP**-complete in general, the QCSP problem is **PSPACE**-complete [67]. QCSPs can be used to model various combinatorial problems such as scheduling [27], planning [192], adversary game playing [28], model checking [162], and solving mechanical design problems [175] for example.

The QCSP problem recently started to attract interest and only very few solution methods have been proposed yet. They incorporate a variety of techniques that are either extensions of techniques used in CSPs and SAT to the quantified case, or are specifically designed for QCSPs. A solver based on an encoding of QCSPs to QSAT was proposed in [103]. A top-down solver that uses generalizations of well-known techniques in CSP like arc-consistency [51, 151], intelligent backtracking, and some other SAT techniques, was proposed in [104]. The first bottom-up solver, which instantiates variables from the innermost to the outermost, was proposed in [220]. The algorithm processes a QCSP problem as if it is composed of pieces of classical CSPs and thus only uses standard CSP techniques. A QCSP solver equipped with many advanced CSP techniques was proposed in [26] and it is built on top of the high performance CSP solver Gecode [1].

As in the case of SAT and IP, QCSP is the generalization of QSAT and QIP to arbitrary constraints. However, another important difference between QIPs and QCSPs is that the latter are restricted to decision problems and do not naturally support the optimization of an objective function. Thus, an optimization problem must be expressed as a sequence of QCSPs. By setting a threshold value on the objective function value, an *"objective constraint"* can be added to the model. Successive adjustments to the threshold value according to whether there are values of the variables that satisfy all constraints, allow the optimal value of the objective function value to be obtained [54]. An approach for quantified constraint optimization has been proposed in [29].

Quantified Constraints over the reals (QCSP problems with interval constraints)

The paradigms QSAT, QIP, and QCSP are essentially discrete. Strongly related to QLPs in the context of constraint satisfaction are real-valued quantified constraints (e.g. first-order formulae over the real numbers). They provide quantifiers ($\exists, \forall$), connectives ($\wedge, \vee, \neg$), predicate symbols (e.g. $=, \neq, <, \leq$), function symbols (e.g. $+, -, \times$), rational constants, and variables ranging over real numbers. They have been considered for more than fifty years in the field of mathematical programming and there is a significant body of work on quantified problems with continuous domains (e.g. [31, 177]). Ratschan gives numerous references to papers on this subject [178].

Quantified formulas over the reals can be solved with quantifier elimination methods [226, 85]. Although quantifier elimination is in the worst case doubly exponential in the number quantifier alternations and exponentially in the number of variables [226], several

approaches have been proposed and successfully applied. The first method was proposed in the seminal work by Tarski in the 1930s (published in the 1950s) [213], where he showed that it is decidable to determine the truth of any proposition in the first-order theory of the real numbers with addition and multiplication. Adding additional function symbols (e.g., sin, tan), usually removes this property [176]. Several improvements to Tarskis algorithm were proposed and to date the best known procedure is *Cylindrical Algebraic Decomposition* (CAD) by Collins [69].

The main drawback of algorithms for real-valued quantified constraints is that they can be dramatically slow and inefficient in practice. Quantifier elimination in real algebra is doubly exponential, even when there is only one free variable and all polynomials in the quantified input are linear [56]. Large instances are currently out-of-reach for existing solvers, but some problems in low dimensions and with a small number of quantifier alternations can be solved [50]. A promising alternative approach is to approximate the problem using finite-precision interval arithmetic [31, 176] and thereby avoid the use of CAD. Nevertheless, CAD-based quantifier elimination remains, to the best of our knowledge, the most efficient general quantifier elimination algorithm to be implemented. In Section 3.1 we introduce the concept of projection in polyhedral spaces in order to eliminate quantified variables from a system of linear equations.

In Section 6.2 we compare the solvers QEPCAD-B [55], Redlog [84], and RSolver [177] when being applied to QLPs and contrast them with our own decomposition-based algorithm (see Section 5.2) and the algorithm proposed in [108] (see Section 3.1).

2.3. Application Examples for Quantified Linear and Integer Programs

QLPs are very expressive and can be used to model problems from a wide variety of domains. QIPs and QMIPs can even be used to encode any **PSPACE**-complete problem. In this section we demonstrate how to model with the help of quantified linear and integer programs and start on a few very simple examples. At the end, we show how the design of a Booster Station can be optimized with the help of an quantified mixed-integer program (QMIP).

2.3.1. A QLP model for a class of Real-Time Scheduling Problems

Scheduling strategies for real-time systems often confront planners with principal issues that are not addressed by traditional scheduling models, like parameter variability in the execution time of tasks and the existence of complex timing constraints among them [108]. Execution time variability and execution time interaction is of both theoretical and practical relevance. Let a set of *ordered*, *non-preemptive* tasks $\mathcal{J} = \{J_1, \dots J_n\}$, with linear constraints imposed on their start times $s = \{s_1, \dots s_n\}$ and their execution times $e = \{e_1, \dots e_n\}$ be given. The non-constant execution time e_i of the i-th task J_i is from the interval $e_i = [u_i, l_i]$ and independent of the start times of the tasks, however they can have complex interdependencies among themselves expressed in constraint system in

matrix form as

$$A(s,e)^T \leq b$$

The problem whether a valid schedule satisfying all timing constraints under all possible execution time combinations does exist or not, can be answered by the following QLP instance:

$$\exists\, s_1 \in [0,T]\ \forall\, e_1 \in [l_1,u_1]\ \exists\, s_2 \in [0,T]\ \forall\, e_2 \in [l_2,u_1]\ \ldots \exists\, s_n \in [0,T]\ \forall\, e_n \in [l_n,u_n]:$$

$$A(s,e)^T \leq b.$$

Note that one cannot simply take the largest possible values for the execution times, because otherwise the schedule might be incorrectly declared infeasible [108].

Example 5. *Considering a set of three tasks $\mathcal{J} = \{J_1, J_2, J_3\}$ with the following constraints on the execution times of the tasks:*

1. *task J_1 finishes after 4 time units at earliest: $4 \leq s_1 + e_1$,*
2. *task J_1 finishes before J_2: $s_1 + e_1 \leq s_2$,*
3. *task J_2 finishes before J_3: $s_2 + e_2 \leq s_3$,*
4. *task J_2 commences within 4 time units of J_1 concluding: $s_2 \leq s_1 + e_1 + 4$,*
5. *task J_3 finishes within 12 time units: $s_3 + e_3 \leq 12$, with*
6. *execution times: $e_1 \in [2,4], e_2 \in [1,4], e_3 \in [2,5]$.*

The corresponding QLP has the following form:

$$s_1 \in [0,25]\ \forall\, e_1 \in [2,4]\ \ s_2 \in [0,25]\ \forall\, e_2 \in [1,4]\ \exists\, s_3 \in [0,25]\ \forall\, e_3 \in [2,5]:$$

$$\begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ -1 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} s_1 \\ e_1 \\ s_2 \\ e_2 \\ s_3 \\ e_3 \end{pmatrix} \leq \begin{pmatrix} -4 \\ 0 \\ 0 \\ 4 \\ 12 \end{pmatrix}.$$

The scheduling model is feasible and task J_1 must not start later than time unit 7 and not earlier than time unit 2.

2.3.2. A QIP model for the Quantified Boolean Satisfiability Problem

Let

$$F = Q_1\, x_1\, Q_2\, x_2\, \ldots Q_n\, x_n\, \mathcal{C} \tag{2.6}$$

be a QSAT problem as in Definition 2.24. F has the following QIP formulation

$$Q_1\, x_1 \in \{0,1\}\; Q_2\, x_2 \in \{0,1\}\; \ldots Q_n\, x_n \in \{0,1\} : Ax \leq b,$$

where each Q_i is either $\exists$ or $\forall$. A has n columns corresponding the n variables and m constraints corresponding to the m clauses in $\mathcal{C}$. Each clause is replaced by a linear constraint, for instance the clause $x_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4$ results in the constraint

$$x_1 + (1 - x_2) + (1 - x_3) + x_4 \geq 1,$$

Example 6. *Consider the following QSAT instance*

$$\forall\, a\, \exists\, b \forall\, c\, \exists\, d\, (\overline{a} \vee c \vee d) \wedge (\overline{b} \vee \overline{d}) \wedge (a \vee b \vee \overline{d}) \wedge (\overline{a} \vee b). \tag{2.7}$$

The corresponding QIP has the following form:

$$\forall\, x_1 \in [0,1]\; \exists\, x_2 \in [0,1]\; \forall\, x_3 \in [0,1]\; \exists\, x_4 \in [0,1] :$$

$$\begin{pmatrix} 1 & 0 & -1 & -1 \\ 0 & 1 & 0 & 1 \\ -1 & -1 & 0 & 1 \\ 1 & -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, x \in \mathbb{Z}.$$

The instance is infeasible.

We will use this type of reformulation in Section 4.2 in order to proof complexity results for QLPs and QIPs.

2.3.3. A QIP model for the Worst-Case Dynamic Graph Reliability Problem

The following example shows how to model a very simple, yet **PSPACE**-complete example: the worst-case dynamic graph reliability (wcDGR) problem as proposed in [89]. This is a simple graph game, where a person has to travel on a graph from a given source node to a desired target node. While he is traveling, an (evil) opponent erases some edges and thus destroys the graph. However, also the opponent must follow certain rules.

Input A directed acyclic graph $G = (V, E)$ with two special nodes s and t. Moreover, there is a mapping $f : (V \times E) \to \{0,1\}$ with $f(v,e) = 1$ if and only if an evil opponent is allowed to erase edge e when we arrive at node v. But the opponent has to follow some rules. For some edges, the opponent is not allowed to erase more than one of two specific edges. In other words, there is another mapping $g : E \times E \to \{0,1\}$ with $g(e_1, e_2) = 1$ if and only if it is allowed to erase e_1 and e_2.

Output Is there a strategy to reach the target node t from start node s, no matter how the opponent acts?

The starting point of the wcDGR is a directed acyclic graph (DAG). An example is shown at the bottom of Figure 2.6. There, we assume that an opponent may erase at most one of the edges e_4 and e_5. He can make them fail when we arrive at node v_2 or at node v_3. Anyway, never both edges are allowed to fail, and no other edges can fail. The optimization problem is to firstly make a choice whether to travel via edge e_1 or e_2. Then, the opponent erases none or one of the edges e_4 and e_5. Thereafter, we choose a remaining path to target t, if one exists. If we move from node v_2 to node v_3, our opponent is again allowed to make one of the two edges e_4 or e_5 fail, if he did not fail an edge with his first choice.

$$\exists x_1, x_2 \forall y_{2,4}, y_{2,5} \exists x_3 \forall y_{3,4}, y_{3,5} \exists x_4, x_5, x_\Delta \text{ (all binary)} :$$

$$\left.\begin{aligned} x_1 &= x_4 + x_3 \\ x_2 + x_3 &= x_5 \\ x_1 + x_2 &= 1 \\ x_4 + x_5 &= 1 \end{aligned}\right\} \text{flow constraints (1)}$$

$$\left.\begin{aligned} x_4 &\leq (1 - y_{2,4}) + (1 - x_1) + x_\Delta \\ x_4 &\leq (1 - y_{3,4}) + (1 - x_2 - x_3) + x_\Delta \\ x_5 &\leq (1 - y_{2,5}) + (1 - x_1) + x_\Delta \\ x_5 &\leq (1 - y_{3,5}) + (1 - x_2 - x_3) + x_\Delta \end{aligned}\right\} \begin{array}{l}\text{failure constraints} \\ \text{for the existential} \\ \text{player (2)}\end{array}$$

$$\left.\begin{aligned} 2x_\Delta &\leq y_{3,4} + y_{3,5} + y_{2,4} + y_{2,5} \end{aligned}\right\} \begin{array}{l}\text{(3) critical failure} \\ \text{constraint for the} \\ \text{universal player.}\end{array}$$

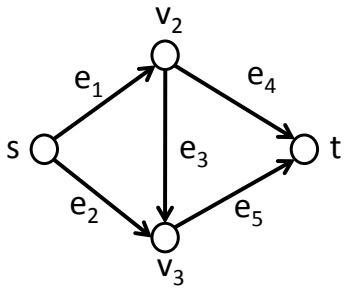


Figure 2.6.: QIP description and graph of a wcDGR example.

Let us introduce variables $x_1, ..., x_5$ for edge choices. $x_i = 1$ means e_i is chosen for traveling. The first block of constraints encodes the flow constraints of the classic shortest path problem on graphs. The constraints are applied to the x-variables (Fig. 2.6, (1)). $y_{2,4}, y_{3,4}, y_{2,5}, y_{3,5}$ determine whether the opponent makes the edges e_4 or e_5 fail when we reach the nodes v_2 or v_3, i.e., $y_{i,j} = 1$ means that edge e_j fails when arriving at node v_i.

The second block (2) couples the decision variables x_i to the $y_{j,k}$-variables of the opponent. E.g. $x_4 \leq (1 - y_{2,4}) + (1 - x_1) + x_\Delta$ means that the existential player will have to set x_4 to zero and will not be allowed to use edge e_4 if the existential player first moves via edge e_1 and then the universal player sets $y_{2,4} = 1$. Strictly seen, we have to test whether the existential player has moved via node v_2. However, a directed graph can always be pre-manipulated such that all nodes of the original graph can be entered via one specific incoming edge. Of course, for this purpose, additional nodes and edges must be added, in general. The variable x_Δ is used to ensure that also the universal player follows his rules. The last constraint (3) $2x_\Delta \leq y_{3,4} + y_{3,5} + y_{2,4} + y_{2,5}$ expresses that the universal player is constrained by $y_{3,4} + y_{3,5} + y_{2,4} + y_{2,5} \leq 1$. If he breaks this rule, the existential player can set $x_\Delta = 1$ and the constraints of the second block are trivialized. The existential player can then trivially win the game. Last but not least, we can express the problem as shown at the top of Figure 2.6, with the given quantifier prefix, because the graph of a wcDGR is a DAG and therefore a partial order (in time) of the nodes can be computed.

Application to other Games

In [89] the **PSPACE**-complete two-person game Gomoku was modeled as a QIP. *Gomoku* or *Five in a Row* is a two-person strategy game played with *Go* pieces on a *Go* board. Both players (black and white) alternately place stones, until the first player gets a row of five horizontally, vertically or diagonally connected stones. A stone cannot be moved or removed from the board once it is placed. With this standard set of rules, it is known that black always wins on some board sizes (e.g. 15×15), but the problem is open for arbitrary $n \times n$ boards. This model was also adopted to a very similar **PSPACE**-complete game: Connect6. Here two players playing on a Go board try to achieve a connected row of six stones. At the beginning, black places one stone, then white and black take turns placing two stones. The player, who has the first connected row of six stones, wins.

2.3.4. A QMIP model for a Booster Station

In the following we demonstrate the applicability of the quantified linear programming approach for the optimization of technical systems. In the development and operation of technical systems that are based on a large number of design and configuration decisions to meet the performance requirements with a limited amount of resources and costs. Problems of this kind are hard to solve as the number of available solutions drastically increases with the number of decisions between discrete alternatives and furthermore decisions often have to be made with incomplete knowledge.

Figure 2.7 illustrates the ideas of Technical Operations Research (TOR)[3]. The following example describes the first four steps of the pyramid for the design of a *booster station* – a component in a fluid network which can cause a pressure increase – with the help of a quantified mixed-integer program proposed by T.Ederer [87]. It is an extension of the model proposed in [168] and uses universal quantification to increase the robustness of the solution with respect to unpredicted pressure demands.

[3] `http://www.fst.tu-darmstadt.de/forschung_fst/systemoptimierung`

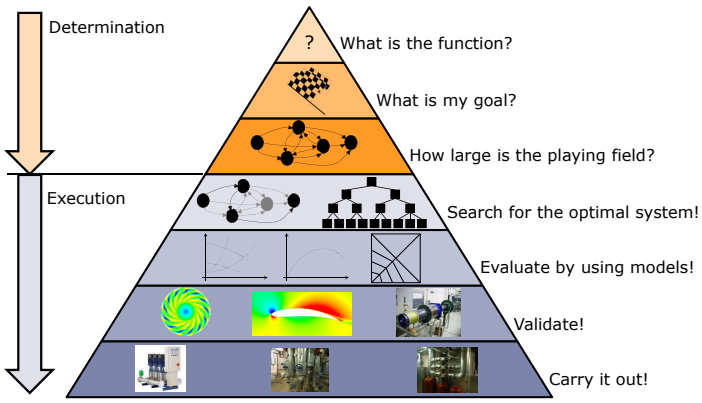


Figure 2.7.: The TOR paradigm [166].

Problem Statement

A booster station is a component in a fluid network which can cause a pressure increase. This component may be a single pump or rather a subnetwork of multiple pumps. If the pumps are not operated in sync and if at most one of these pumps is speed-controlled, the booster station can be regarded as a *black box* pump with a compound head curve. However, such a booster station is no improvement over a single pump with the same head curve. (A single large pump may even be more efficient due to scaling laws.)

Thus, the advantage of a booster station over a single pump consists in the freedom to deactivate individual pumps or to speed-control more than one pump such that the active pumps may operate near their optimal working point. If the average demand of volume flow or pressure head is significantly smaller than the maximal demand, the system developer might therefore provide two or three smaller pumps in parallel or respectively serial connection.

The problem with this approach is that a system developer has to make his decision about the system topology in advance. It might even be reasonable to plan a serial-parallel subnetwork of three or four small pumps, but it is hard to judge which topology is the best one and even then, one is bound by this initial decision. The developer can free himself from this restriction if he allows for redundant or *contradictory* pipes between several pumps, such that disabling some of them (e.g., by digital valves) results in different reasonable topologies. We propose a optimization model for this *extended* booster station design problem in the following.

Modeling Principles

The quantified model consists of two stages: First, find a low-priced investment decision from an adequate set of pumps and pipes (with optional digital valves). Second, activate a subset of these pumps and pipes for each quasi-static demand such that they satisfy the demand and operate near their optimal working point. The overall goal is to minimize the sum of investment costs and energy costs over the expected life cycle.

An interconnection of pumps can be abstracted as a directed graph $G = (V, E)$ with vertices V representing pumps and edges E representing pipes. To connect this pump subnetwork to the main network, we need two additional vertices representing plugs, namely a water supply s and a water outlet t, with corresponding pipes. Therefore, all possible topologies of a booster station can be modeled as a *complete* directed graph of all available pumps and two water plugs. Table 2.1 shows an example for a booster construction kit. The respective head curves are depicted in Figure 2.8.

Table 2.1. Example of a booster construction kit.

pump	speed-controlled	rotary speed	
		min	max
1	yes	730 rpm	2920 rpm
2	yes	350 rpm	1400 rpm
3	yes	725 rpm	2900 rpm

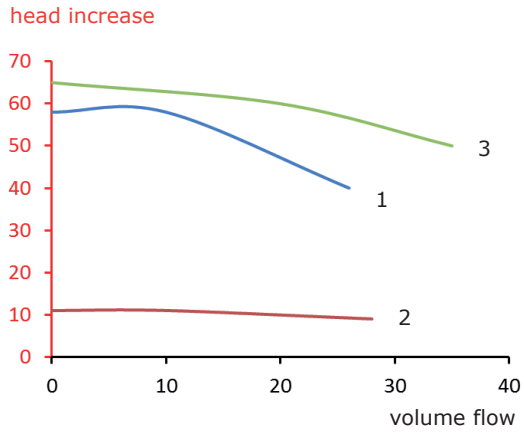


Figure 2.8.: Head curves at maximum rotary speeds.

The first-stage decision (i.e., the investment) is given by a set of binary variables (indicators) y_v for each vertex v and $y_{i,j}$ for each edge (i, j) of the graph. Of course, at least both plugs, one pump and two edges need to be selected for a reasonable booster station. In the objective function, theses indicators are weighed with the purchase costs of their respective components.

After the investment has been made, the demand of volume flow Q and head increase H^+ will change over the booster station's life time. For the sake of simplicity we present a quasi-static model, i.e., we identify similar demands to *demand scenarios* σ and give a discrete probability distribution. Table 2.2 shows an example demand profile. In this case, the booster station is almost always confronted with one of three representative scenarios in arbitrary succession. The system must be able to satisfy each of these demands and the cost of operation can be estimated as a weighed sum of the cost per scenario with its respective probability. We thereby assume that switching between scenarios is fast compared to the

contiguous operation phases and that it involves no significant cost.

Table 2.2. Example of a quasistatic demand distribution.

scenario	probability	volume flow	head increase
1	50 %	$25.0\,\mathrm{m}^3/\mathrm{h}$	$30\,\mathrm{m}$
2	25 %	$50.0\,\mathrm{m}^3/\mathrm{h}$	$30\,\mathrm{m}$
3	25 %	$25.0\,\mathrm{m}^3/\mathrm{h}$	$60\,\mathrm{m}$

A second-stage decision (i.e., the activation and operation) is dependent on the investment and on a specific scenario. It mainly consists of the following collection of variables:

component activation: a set of binary variables x_v for each vertex v and $x_{i,j}$ for each edge (i, j), which indicates if the respective pump is switched on or off and if the respective valve is open or closed.

fluid network: a set of three continuous variables $q_{i,j}$ for each edge (i, j) and h_v^s, h_v^t for each vertex v, indicating the volume flow through each pipe and the pressure head before and after each pump. (The absolute pressure head is arbitrary. We may fix the pressure head at the source to $0\,\mathrm{m}$.)

pump operation: a set of four continuous variables q_v, h_v^+, p_v and n_v for each vertex v, indicating the volume flow through each pump or plug, its pressure increase, its power drain and its rotary speed.

Apart from these, we need several auxiliary variables, which will be introduced later in connection with their corresponding constraints.

Figure 2.9 shows an optimal solution to the example. The investment decision consists of 2 pumps and 5 pipes. Depending on the scenario, only one or both pumps are running. When the pumps are operated in parallel, the booster station can satisfy a high volume flow and when the pumps are configured serially, they achieve a high head increase. The optimality of the solution implies that each cheaper investment either cannot satisfy all scenarios or has much higher expected operational costs, and that no more expensive investment can be justified by sufficiently smaller operational costs.

The corresponding mathematical model in form of a mixed-integer linear program has the following main components to describe the constraints of the system.

Investment Constraints Water supply and water outlet are necessary.

$$y_s = 1,\ y_t = 1 \tag{2.8}$$

There must be at most one pipe between two pumps.

$$\forall i, j: \quad y_{i,j} + y_{j,i} \leq 1 \tag{2.9}$$

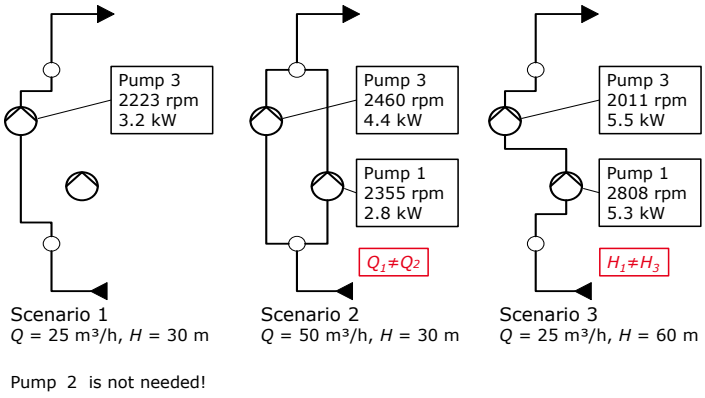


Figure 2.9.: Depiction of an optimal solution.

Activation Constraints Components may only be activated, if they have been purchased.

$$\forall v : x_v \leq y_v \quad \text{and} \quad \forall i,j : \quad x_{i,j} \leq y_{i,j} \tag{2.10}$$

Fluid Network Constraints The demand must be satisfied in each scenario. Depending on piecewise linear approximations in later parts of the model, it might be reasonable to soften these equations to inequalities with a small gap.

$$q_s = Q_\sigma,\ q_t = Q_\sigma \quad \text{and} \quad h_s^t = 0,\ h_t^s = H_\sigma^+ \tag{2.11}$$

The volume flow has to satisfy the continuity equation, i.e., volume flow is preserved.

$$\forall v : \quad \sum_i q_{i,v} = q_v \quad \text{and} \quad \forall v : \quad q_v = \sum_j q_{v,j} \tag{2.12}$$

Pressure head propagates through the network. Note that for simplicity, we leave out pressures losses.

$$\forall v : \quad h_v^s + h_v^+ = h_v^t \quad \text{and} \quad \forall i,j : \quad h_i^t = h_j^s \tag{2.13}$$

Pump Operation Constraints The pump operation variables, i.e., volume flow, pressure increase, power drain and rotary speed, are coupled by head curves as in Figure 2.8. In general, these are nonlinear relations which cannot be modelled exactly in a linear program. Instead, they are modeled by piecewise linear approximations, i.e., by interpolating or approximating linear splines. An overview of several possible formulations is given in [222]. Therefore, we need to introduce auxiliary variables at each vertex, e.g. some binary variables ζ to select a part of the spline and continuous variables λ to interpolate between two nodes. Depending on the chosen formulations, they are coupled by some generic constraints.

Adding Feasibility Robustness with the help of universal Quantifiers

Up to now, we discussed a two-stage optimization model with a handful of scenarios. In principle, we can approach this problem with standard *branch and cut solvers* by formulating and solving the deterministic equivalent program (DEP)(see also Section 4.1.3 for more details). That is, we duplicate the second-stage variables and constraints for each scenario, which results in a large mixed-integer linear program with a block-ladder structure. Thus, building the DEP is no reasonable solution approach if the number of scenarios becomes huge.

In practice, we need to analyze more than a couple of scenarios. In the given example, the model guarantees that all three scenarios can be satisfied - however, we do not know if every demand *in between* can also be fulfilled. Those demands may be rare, but it nevertheless is not acceptable that the booster station fails for any reasonable demand.

To our best knowledge, there are – to date – no algorithms to solve average-case two-stage mixed-integer linear programs with many scenarios exact *and* fast. However, there are promising solution techniques for their worst-case counterparts. We therefore propose a mixed-quantified model, which finds the best-priced booster station with respect to a handful of frequent scenarios, out of all booster stations that are simultaneously guaranteed to satisfy a huge number of infrequent scenarios α (whatever the operational costs). The proposed model takes the following schematic form:

$$\begin{aligned} &\min \big(\text{ investment costs} + E_{\text{frequent scenarios}}(\text{ operational costs }) \big) \\ &\text{s.t.} \quad \exists \text{ investment decision} \quad \forall \text{ scenarios} \quad \exists \text{ operational solution} \end{aligned}$$

To load this problem with a worst-case two-stage solver, we have to expand the expectation value in the objective function to its deterministic equivalent form. The size of the program grows by a factor of e.g. 5 or 10, but the huge number of infrequent scenarios can remain in the two-stage model and take full advantage of two-stage solution techniques. In the following complete example, let σ denote the frequent scenarios with probability P_σ and associated cost variable c_σ, and let the set b_k be the binary encoding and u_n be the unary encoding of the infrequent scenarios β. Then, the worst-case two-stage model can be written as follows:

$$\begin{aligned} &\min \left(C^\top(y_v,\, y_{i,j}) + \sum_\sigma (P_\sigma \cdot c_\sigma) \right) \\ &\text{s.t.} \quad \exists\, y_v,\, y_{i,j} \quad \forall\, b_k \quad \exists\, u_n,\, \tilde{q},\, \tilde{h}^+,\, x_v,\, x_{i,j} \in E, \ldots \end{aligned}$$

$$(2.8), \quad (2.9), \quad (2.10)$$

$$q_s = \tilde{q},\ q_t = \tilde{q} \quad \text{and} \quad h_s^t = 0,\ h_t^s = \tilde{h}^+ \tag{2.11'}$$

(2.12), (2.13)

$$\begin{aligned}\sum_k 2^k \cdot b_k &= \sum_n n \cdot u_n \\ 1 &= \sum_n u_n \\ \tilde{q} &= \sum_n Q_n \cdot u_n \\ \tilde{h}^+ &= \sum_n H_n^+ \cdot u_n\end{aligned}$$

3. Polyhedral and Algorithmic Properties

In this chapter we consider the basic properties of quantified linear programming problems. Quantified linear programming is a much harder problem than linear programming and neither the theory nor the computational aspects are as developed as they are for traditional linear programs. As throughout this thesis, the emphasis is on results that have direct application for the solution of QLPs, however, we also include those results we consider most central to the overall development in this field. We start in Section 3.1 with quantifier elimination techniques as a preparation for the technical results. Afterwards, in Section 3.2 we study the polyhedral properties of QLPs, and this chapter finishes with the introduction of certain kinds of QLP relaxations in Section 3.3.

3.1. Elimination of Quantified Variables

In the following, we introduce the concept of *projection in polyhedral spaces* in order to eliminate quantified variables. Given a polyhedron in $\mathbb{K}^n$, one can project this system to a lower dimension space $\mathbb{K}^m$ with $m < n$, while preserving the set of solutions of the original system [197].

Definition 3.1 (Projection Function)**.** *Let $S, H \subseteq \mathbb{K}^n$ and $c \in \mathbb{K}^n \setminus \{0\}$. The set*

$$\{x \in H | \exists \lambda \in \mathbb{K} \text{ with } x + \lambda c \in S\}$$

*is called **projection** from S to H along c. If for a $\gamma \in \mathbb{K}$*

$$H = \{x \in \mathbb{K}^n | c^T x = \gamma\}$$

*holds, the projection is called **orthogonal projection**.*

A technique to eliminate existentially quantified variables is the Fourier-Motzkin elimination method [73], though, in general every polyhedral projection method suffices [146, 228]. For the special case of universally quantified variables, whose domain forms an n-dimensional hyperrectangle, we can apply a simple variable substitution in order to eliminate variables from these set [197]. However, whereas the order in which quantifiers are eliminated is irrelevant when all of them are existential or all of them are universal, it is well-known that the order is of particular importance in the presence of quantifier alternation (strict alternation and alternation of different quantifier blocks). For this reason existing elimination procedures eliminate the innermost variable first, followed by the next innermost and so on [227].

3.1.1. Elimination of Existentially Quantified Variables

To eliminate a variable in a QLP that looks as follows

$$\exists^n \circ x \in [l, u] : \; Ax \leq b, \tag{3.1}$$

which in fact is a traditional LP with $x \in \mathbb{Q}^n$, we apply the Fourier-Motzkin elimination method (FME-method) [73, 126, 140], which was originally proposed by Fourier in 1827 and reinvented several times, e.g. by Motzkin in 1936 [159]. It is a variable elimination scheme based on polyhedral projection, a computationally expensive but powerful method for solving a system of linear inequalities. It can be thought of as the linear programming equivalent to *Gaussian elimination* for solving a system of linear equations.

When the FME-method is applied to a problem 3.1, variables are eliminated, in turn between the inequalities in which they occur. Geometrically this can be visualized as projecting a polyhedron down to a lower dimensional space. Mathematically, an elimination step is realized by an orthogonal projection $\pi : \mathbb{Q}^n \rightarrow \mathbb{Q}^{n-1}$ such that the polyhedron $P(A, b)$ with $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$, is projected to a subspace of lower dimension, as depicted in Figure 3.1.

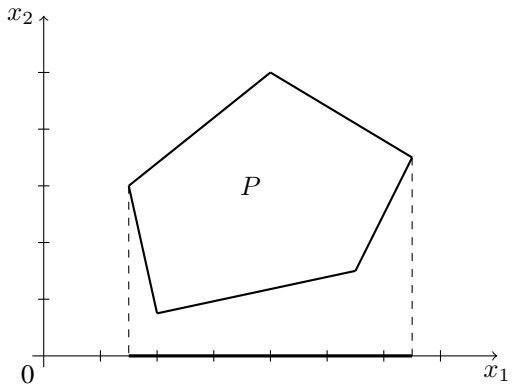


Figure 3.1.: Projection to the x_1-plane.

For the resulting polyhedron $P(B, d)$ with $B \in \mathbb{Q}^{k \times (n-1)}$ and $d \in \mathbb{Q}^k$ holds:

$$\pi[P(A, b)] = P(B, d), \text{ and } P(A, b) \neq \emptyset \Leftrightarrow P(B, d) \neq \emptyset.$$

This constitutes that the FME-method is solution preserving, the new system with $(n-1)$ variables has a solution if and only if the original system with n variables has a solution. Suppose we wish to eliminate variable x_1, starting from a system of linear inequalities

$$a_i^T x = \sum_{j=1}^{n} a_{ij} x_j \leq b_i \; (\forall i \in I).$$

The goal is to create an equivalent system $A'x' \leq b'$ defined on the variables $x' = (x_2, \ldots, x_n)$. The FME-Method accomplishes this as follows:

1. The inequalities are sorted into three subsets, depending on the coefficient of variable x_1. As a result we obtain a set of lower and upper bound constraints and a set of constraints that do not contain x_1. Let

$$I_+ = \{i \in I | a_{i1} > 0\}, I_- = \{i \in I | a_{i1} < 0\}, \text{and } I_0 = \{i \in I | a_{i1} = 0\}$$

   denote these sets. (Note that if x_1 occurs in an equality constraint, we can use this constraint to substitute x_1 out of the other equations with a simple pivot operation.)

2. Inequalities in I_+ and I_- are divided by a_{i1} and x_1 is isolated on the left-hand side of the constraints. The equivalent linear system results as:

$$\begin{aligned} x_1 &\leq \tfrac{1}{a_{r1}}(b_r - \sum_{j=2}^{n} a_{rj}x_j) \ (r \in I_+), \\ x_1 &\geq \tfrac{1}{a_{s1}}(b_s + \sum_{j=2}^{n} a_{sj}x_j) \ (s \in I_-), \\ \sum_{j=2}^{n} a_{tj}x_j &\leq b_t \ (t \in I_0). \end{aligned}$$

3. To eliminate x_1, the set of all combinations of the lower and upper bound constraints are computed ($I = I_0 \cup (I_+ \times I_-)$). Afterwards x_1 can be eliminated in in all resulting inequalities such that for all pairs $s \in I_-$ and $r \in I_+$ holds:

$$\tfrac{1}{a_{s1}}(b_s + \sum_{j=2}^{n} a_{sj}x_j) \leq \tfrac{1}{a_{r1}}(b_r - \sum_{j=2}^{n} a_{rj}x_j).$$

These steps can be repeated with the resulting system $A'x' \leq b'$ for variable x_2, and so on, until all variables are eliminated. If the resulting b' (after eliminating x_n) has no nonnegative entries, the original system $Ax \leq b$ (and all intermediate) can be declared feasible, otherwise it can be declared infeasible. The Function `ELIM-EXIST-VARIABLE()` as presented in Algorithm 1 is an algorithmic description of the FMW-method[1].

The correctness of the FMW-method has been shown in [73, 161]. Eliminating an existentially quantified variable can lead to a quadratic increase of the number of constraints. If there are m constraints prior to the elimination step, there could be $\frac{m^4}{2}$ constraints after the elimination [73]. Eliminating k variables using the FME method, thus can result in $O(m^{2^k})$ new constraints [197]. However, there are a class of constraints that are closed under Fourier-Motzkin elimination and for which the FMW method only has polynomial worst-case time complexity [73, 197]. Apart from that, this method is a useful tool for proving theorems on polyhedral spaces [64].

[1]Note that m_j and n_k in are linear functions of x.

Algorithm 1 ELIM-EXIST-VARIABLE (A, b, x_i, l_i, u_i)

1: Form the set $L_{\leq}$ of constraints that can be written as $x_i \leq m_j$.
2: Form the set $L_{\geq}$ of constraints that can be written as $x_i \geq n_k$.
3: Form the set $L_{=}$ of constraints that do not contain x_i.
4: $L_{\leq} := L_{\leq} \cup \{x_i \leq u_i\}$.
5: $L_{\geq} := L_{\geq} \cup \{x_i \geq l_i\}$.
6: $L := L_{=}$.
7: **for** each constraint $x_i \leq m_j \in L_{\leq}$ **do**
8: **for** each constraint $x_i \geq n_k \in L_{\geq}$ **do**
9: $L := L \cup \{n_k \leq m_j\}$
10: **end for**
11: **end for**
12: Let x' be x without x_i. Create the new matrix A' and the new vector b', such that $A'x' \leq b'$ reflects all constraints of L.
13: (A', b')

3.1.2. Elimination of Universally Quantified Variables

To eliminate a variable in a QLP of the form

$$\forall^n \circ x \in [l, u] : Ax \leq b, \tag{3.2}$$

we use the fact that the domain of the universally quantified variables

$$E = \prod_{i=1}^{n} [l_{\forall i}, u_{\forall i}] = [l_1, u_1] \times [l_2, u_2] \cdots \times [l_n, u_n], \tag{3.3}$$

forms a $n-$dimensional hyperrectangle ($n-$dimensional analogue of a rectangle $(n = 2)$). To check whether for all possible variable allocations from the set E, the constraint system $Ax \leq b$ is feasible, we must simply check if all vertices of E are inside the polytope $P = P(A, b)$ as shown in Figure 3.2. This can be done independently for each dimension of E and each constraint in $Ax \leq b$ by the function `ELIM-UNIV-VARIABLE()` as shown in Algorithm 2. It proceeds by eliminating one universal variable at a time, until

Algorithm 2 ELIM-UNIV-VARIABLE (A, b, x_i, l_i, u_i)

1: Substitute $x_i := l_i$ in constraints that can be written as $x_i \geq n_k$.
2: Substitute $x_i := u_i$ in constraints that can be written as $x_i \leq m_j$.
3: Let x' be x without x_i. Create the new coefficient matrix A' and the vector b', such that $A'x' \leq b'$ reflects the altered constraints where x_i has either been substituted by l_i or u_i.
4: (A', b')

all variables are eliminated. At this point we obtain a system of inequalities with constant left-hand side and constant right-hand side that can trivially be checked for feasibility. The elimination of a universally quantified variable does not increase the number of constraints and can be computed in $O(m \cdot n)$ time when the matrix has n variables and m constraints.

The correctness of this procedure was shown in [197, 209].

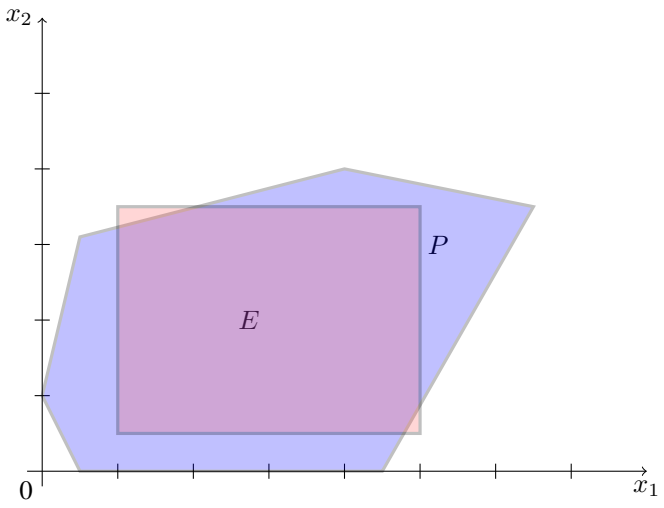


Figure 3.2.: Polytope inclusion.

3.1.3. Elimination of Nested Quantified Variables

In the following, an algorithm to solve general continuous QLPs is presented. It is based on the elimination techniques mentioned above and a slight variation of the procedure proposed in [108] (for restricted constraint classes) and later for general linear constraints [212]. We present the algorithm in a more general form. The main difference of our variant to the procedure in the literature is that the quantifiers need not necessarily alternate. Clearly, the given generalization does not influence the correctness arguments as presented in [212]. Function `QLP-DECIDE()` as described in Algorithm 3 combines the two operations `ELIM-EXIST-VARIABLE()` and `ELIM-UNIV-VARIABLE()` in a loop that eliminates variable by variable starting from the inner most one going up to the first one. Each step in the algorithm is solution preserving and after the last elimination step, the original system is reduced to a system without variables. If for all remaining constraints, the left-hand side is less than or equal to the right-hand side, that is if $a \leq b$, then the original system is feasible, otherwise not.

The constraints that originate when a variable (existential or universal) is eliminated can make an existing constraint redundant, be redundant themselves, or create an inconsistency. The first two cases cases are handled by the function `PRUNE-CONSTRAINTS()`. If two constraints have the same coefficients, then the constraint with the smaller right-hand side of a $\leq$ relation is retained, and the constraint with the bigger right-hand side of a $\geq$ relation respectively. However, more advanced checks are possible but the computational effort must be balanced against the savings. An inconsistency is detected by the function `CHECK-INCONSISTENCY()`, whereupon the entire QLP is declared infeasible. The following cases can simply be checked:

Algorithm 3 QLP-DECIDE(Q, A, b)

```
1: A'_n = A, b'_n = b
2: for i = n down to 1 do
3:    Q_i
4:    ∀: (A'_{i-1}, b'_{i-1}) = ELIM-UNIV-VARIABLE(A'_i, b'_i, x_i, l_i, u_i)
5:    ∃: (A'_{i-1}, b'_{i-1}) = ELIM-EXIST-VARIABLE(A'_i, b'_i, x_i, l_i, u_i)
6:
7:    if CHECK-INCONSISTENCY() then
8:       false (System is infeasible)
9:    end if
10:   PRUNE-CONSTRAINTS()
11: end for
12: true (System is feasible)
```

1. Two constraints of the form $x_i \leq a$ and $x_i \geq b$, with $a \leq b$ result in a inconsistency.

2. One constraint of the form $y_i \geq k$, with $k \leq l_i$ is redundant, otherwise $k \geq l_i$ it creates an inconsistency.

3. One constraint of the form $y_i \leq k$, with $k \geq u_i$ is redundant, otherwise $k \leq u_i$ it creates an inconsistency.

The correctness of Algorithm 3 (`QLP-DECIDE()`) was proved in [108] using polyhedral arguments, and follows from the correctness of Algorithm 1 and Algorithm 2 as shown in [197]. In [212] the correctness of Algorithm 2 when used to eliminate the inner most variable x_n if it is a universal variable, was shown with a completely different proof using two-person game-theoretic arguments. Here, the given instance is interpreted as a game between the existential and the universal player. The existential player tries to enforce a game x such that $Ax \leq b$, while the universal player tries to make at least one of the inequalities invalid such that $Ax \not\leq b$ for at least one constraint. Thus, no matter how $x_1, \ldots, x_{n-1}$ have been fixed, there is a winning strategy for the existential player if and only if there is a winning strategy in a modified constraint system where the universal player has replaced the variable x_n by the worst substitution between l_n and u_n for the existential player, inequality by inequality.

Eliminating a universally quantified variable does not increase the number of constraints but clearly, the super exponential blowup of the FME method makes Algorithm 3 impractical for general QLPs, however, it is very useful for theoretical investigations and special subclasses of QLPs [108]. For the rest of this thesis Algorithm 3 is called *Quantifier Elimination Algorithm (QEA)* or for short *QEA-method*.

Example 7. *In the following we illustrate with an simple example how the algorithm works in practice and we furthermore show up the geometrical interpretation of the single elimination steps. Given the QLP from Example 1*

$$\exists\, x_1 \in [0,1]\ \forall\, x_2 \in [0,1]\ \exists\, x_3 \in [0,1]:$$

$$\begin{pmatrix} 0 & -1 & -1 \\ -1 & 1 & 1 \\ 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} -1 \\ 1 \\ 3 \end{pmatrix},$$

with a solution space as shown in Figure 3.3,

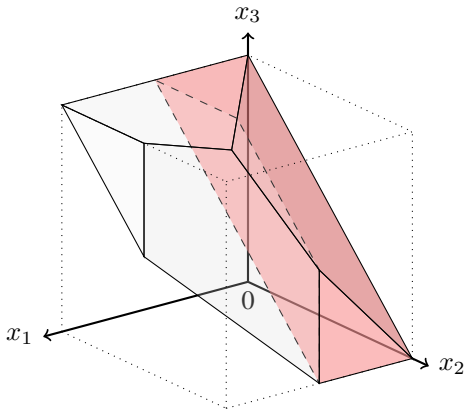


Figure 3.3.: Constraint polyhedron before variable elimination.

we first push the bounds of the existential variables x_1 and x_2 into the constraint system. As determined by Algorithm 1, we get the following sets of normalized constraints for the existential variable x_3:

$$L_{\leq} := \{x_3 \leq 1, x_3 \leq 1 + x_1 - x2\} \tag{3.4}$$
$$L_{\geq} := \{x_3 \geq 0, x_3 \geq 1 - x2\} \tag{3.5}$$
$$L_{=} := \{x_1 \geq 0, x_1 \leq 1, 2x_1 + 2_{x2} \leq 3\}. \tag{3.6}$$

After eliminating variable x_3 by computing all combinations of the lower and upper bound constraints ($L_{\leq} \times L_{\geq}$) the following new constraints are formed

$$0 \leq 1 \tag{3.7}$$
$$-x_2 \leq 0 \tag{3.8}$$
$$-x1 + x2 \leq 1 \tag{3.9}$$
$$x_1 \geq 0 \tag{3.10}$$

but only inequality 3.8 *and* 3.9 *are kept because the others are trivially redundant (although it is also easy to see that* 3.8 *and* 3.9 *are redundant). Thus the resulting set of inequalities after the elimination of variable x_3 is*

$$L := \{x_1 \geq 0, x_1 \leq 1, 2x_1 + 2x_2 \leq 3, -x_1 + x_2 \leq 1, -x_2 \leq 0\} \tag{3.11}$$

whose constraint polyhedron results from the projection of the original constraint polyhedron to the x_1, x_2 plane as visualized in Figure 3.4. As determined by algorithm 2 we get

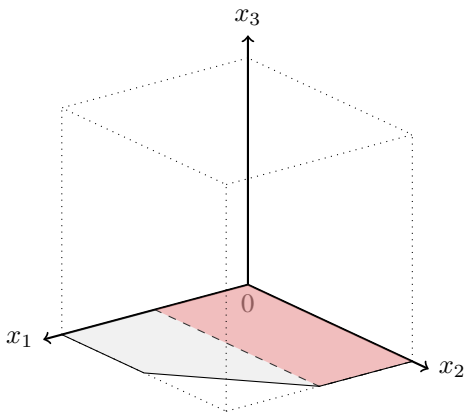


Figure 3.4.: Constraint polyhedron after elimination of variable x_3.

the following sets of normalized constraints for the universal variable x_2

$$L_{\leq} := \{x_1 + x_2 \leq 1.5, -x_1 + x_2 \leq 1\} \tag{3.12}$$
$$L_{\geq} := \{\} \tag{3.13}$$
$$L_{=} := \{x_1 \geq 0, x_1 \leq 1\}. \tag{3.14}$$

After substituting the upper and lower bounds the remaining constraints form the interval $[0, 0.5]$ *as shown on the* x_1*-axis in Figure 3.5. Thus, the initial QLP is feasible for any*

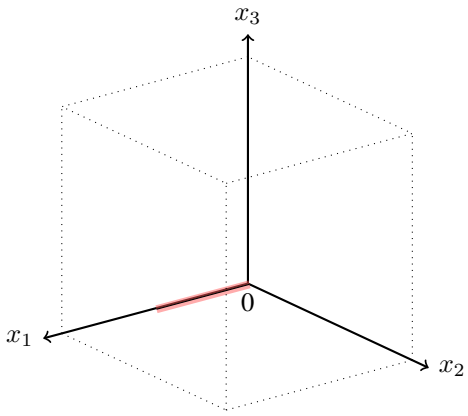


Figure 3.5.: Constraint polyhedron after elimination of variables x_2 and x_3.

$x_1 \in [0, 0.5]$.

3.2. Polyhedral Properties of Quantified Linear Programs

3.2.1. Convexity

In the following section we give some definitions of convexity as presented in [147] for the models are analyzed in the following.

Definition 3.2 (Convex Combination of Existential Strategies)**.** *Let $\mathcal{S}_0$ and $\mathcal{S}_1$ be two strategies for the same universally discrete QMIP instance. We call $\mathcal{S} = (V, E, L)$ a convex combination of $\mathcal{S}_0 = (V, E, L^0)$ and $\mathcal{S}_1 = (V, E, L^1)$ if and only if there is some $\alpha \in [0, 1]$ such that $L_e = (1 - \alpha)\, L_e^0 + \alpha\, L_e^1$ holds for each edge $e \in E$. We write $\mathcal{S} = (1 - \alpha)\, \mathcal{S}_0 + \alpha\, \mathcal{S}_1$.*

Note that the convex combination of universal-edge labels is trivial, since corresponding universal edges have the same label in both strategies.

Lemma 3.1 (Existential Strategy Convexity)**.** *If $\mathcal{S}_0$ and $\mathcal{S}_1$ are existential winning strategies for the same universally discrete QMIP instance, then for each $\alpha \in [0, 1]$ the convex combination of these strategies $\mathcal{S} = (1 - \alpha)\, \mathcal{S}_0 + \alpha\, \mathcal{S}_1$ is also an existential winning strategy.*

Proof. Let p be any path from the root to some leaf in $\mathcal{S}$ and let p_0 and p_1 be the corresponding paths in $\mathcal{S}_0$ and $\mathcal{S}_1$ such that each of the three paths p, p_0 and p_1 represents the same set of edges $\{e_1, \ldots, e_n\}$ with the corresponding labels c_i, c_i^0 or c_i^1. Because $\mathcal{S}_0$ and $\mathcal{S}_1$ are winning strategies, p_0 and p_1 represent feasible solutions x_0 and x_1 with $Ax_0 \leq b$ and $Ax_1 \leq b$. This implies for the vector $x = (1 - \alpha)\, x_0 + \alpha\, x_1$, $\alpha \in [0, 1]$ that $Ax \leq b$. Therefore, $\mathcal{S}$ is a strategy and all of its paths lead to points within the polytope $\{x : Ax \leq b,\, l \leq x \leq u\}$, and thus also $\mathcal{S}$ is a winning strategy. □

Definition 3.3 (Convex Combination of Existential Policies)**.** *Let $\mathcal{P}_0$ and $\mathcal{P}_1$ be two existential policies for the same QLP instance. Define the algorithm $A_{\mathcal{P}}$ that computes $x_i^{\mathcal{P}} = f_i^{\mathcal{P}}(x_1, \ldots, x_{i-1})$ for each $i \in I_\exists$ as follows: In order to compute $x_i^{\mathcal{P}}$ it supplies $\mathcal{P}_0$ and $\mathcal{P}_1$ with the input $x_1, \ldots, x_{i-1}$ and then combines the outputs $x_i^{\mathcal{P}_0}$ of $\mathcal{P}_0$ and $x_i^{\mathcal{P}_1}$ of $\mathcal{P}_1$ to $x_i^{\mathcal{P}} = (1-\alpha)\, x_i^{\mathcal{P}_0} + \alpha\, x_i^{\mathcal{P}_1}$ for an arbitrary but fixed $\alpha \in [0, 1]$. We say A_P computes the convex combination $\mathcal{P}$ of the two policies $\mathcal{P}_0$ and $\mathcal{P}_1$ and write $\mathcal{P} = (1-\alpha)\, \mathcal{P}_0 + \alpha\, \mathcal{P}_1$.*

Lemma 3.2 (Existential Policy Convexity)**.** *If $\mathcal{P}_0$ and $\mathcal{P}_1$ are winning policies for the existential player, then for each $\alpha \in [0, 1]$ the convex combination $\mathcal{P} = (1 - \alpha)\, \mathcal{P}^0 + \alpha\, \mathcal{P}^1$ of these policies is also a winning policy for the existential player.*

Proof. Because $\mathcal{P}_0$ and $\mathcal{P}_1$ are existential winning policies for the same QLP instance, every game that can result from these policies is won by the existential player. Now for each possible universal move sequence, the following is valid: If $x^{\mathcal{P}_0}$ is a point generated by $\mathcal{P}_0$ and $x^{\mathcal{P}_1}$ is a point generated by $\mathcal{P}_1$, and both are in the polytope of $Ax \leq b$, then, by convexity, $x^{\mathcal{P}} = (1 - \alpha)\, x^{\mathcal{P}_0} + \alpha\, x^{\mathcal{P}_1}$ is also in this polytope. □

Lemma 3.3 (Existential Policy Extensibility)**.** *Let Q be a QLP instance with the first variable being universally quantified. Let Q_0 and Q_1 be the remaining QLPs after fixing the first variable of Q to its lower and upper bound respectively. If $\mathcal{P}_0$ is an existential winning policy for Q_0 and $\mathcal{P}_1$ is an existential winning policy for Q_1, there is a natural extension to an existential winning policy $\mathcal{P}$ for Q.*

Proof. For each possible universal move sequence, the following is valid: The first universal move $x_1 = (1-\alpha)\, l_1 + \alpha\, u_1$ gives us an $\alpha \in [0,1]$ such that the convex combination $\mathcal{P} = (1-\alpha)\, \mathcal{P}_0 + \alpha\, \mathcal{P}_1$ has the following properties: Each game resulting from this policy is a convex combination of the games which would have resulted from the universal player moving $x_1 = l_1$ and $x_1 = u_1$ while preserving his remaining moves. Since these games are won by the existential player, their convex combination is as well. □

Note that there is nothing to show if the first variable is *existentially* quantified: A winning policy which is given for any fixation of the first variable can trivially be extended to an existential winning policy for the original problem by applying the fixation.

Proposition 3.4 (Existential Strategy Interpolation)**.** *Let $\bar{Q}$ be a universally discrete QMIP instance and let Q be its relaxed QLP, i.e., the same problem without restricting the universal variables to the integer domain. Each winning strategy $\mathcal{S}$ for the existential player in $\bar{Q}$ implies a winning policy $\mathcal{P}$ for the existential player in Q.*

Proof. We claim that any algorithm which computes existential decisions by interpolating games of the strategy games of the QMIP that belong to the winning strategy $\mathcal{S}$ is an existential winning policy. We will We use an inductive argument and construct such an algorithm by successive policy extension. Let Q^i for $i \in \{1, \ldots, |x_\forall|\}$ be the QMIP resulting from Q by restricting the first i universally quantified variables to the integer domain. Of course, a winning policy for the existential player in problem $Q^{|x_\forall|} = \bar{Q}$ is to follow the winning strategy $\mathcal{S}$. Now suppose we have a winning policy for the existential player in Q^i: if we fix all variables before the i-th universal one according to the existential winning policy in Q^i and consider the remaining QLP, we may interpolate a winning policy by Lemma 3.3. It follows that we can interpolate an existential winning strategy S in $Q^{|x_\forall|} = \bar{Q}$ to an existential winning policy $\mathcal{P}$ in $Q^0 = Q$. □

Indeed, strategy interpolation implies that some games of existential winning strategies for QMIPs with more than two integer values in any universal variable range are redundant: The games or rather subtrees belonging to inner universal choices are superfluous since they can be interpolated. This insight leads to a natural extension of the strategy term to continuous instances.

Corollary 3.5 (Universal Bounding)**.** *Given a QLP instance, there is a winning policy for the existential player if and only if there is an existential winning strategy $T = (V, E, L)$ where each node $v_\forall \in V_\forall$ has exactly two children with the edge labels being the corresponding variables' lower and upper bound. (cf. [212])*

Proof. A winning strategy implies a winning policy by Proposition 3.4 and its subsequent paragraph. A winning policy implies a winning strategy by evaluating the policy on all universal variable bound combinations. □

Therefore, we only need to probe all possible combinations of the bounds of the universal player's variable ranges to solve the QLP problem and the domain of the universal variables is not longer the k-dimensional hyperrectangle

$$E = \prod_{i=1}^{|x_\forall|} [l_{\forall i}, u_{\forall i}] = [l_{\forall 1}, u_{\forall 1}] \times [l_{\forall 2}, u_{\forall 2}] \times \cdots \times [l_{\forall k}, u_{\forall k}],$$

but becomes a discrete set

$$E = \prod_{i=1}^{|x_\forall|} \{l_{\forall i}, u_{\forall i}\} = \{l_{\forall 1}, u_{\forall 1}\} \times \{l_{\forall 2}, u_{\forall 2}\} \times \cdots \times \{l_{\forall k}, u_{\forall k}\}.$$

Thus we can rewrite a QLP instance as

$$\exists x_1 \; \forall x_2 \in \{l_2, u_2\} \; \ldots \forall x_{n-1} \in \{l_{n-1}, u_{n-1}\} \; \exists x_n \; : (A_\exists, A_\forall) \cdot \begin{pmatrix} x_\exists \\ x_\forall \end{pmatrix} \leq b, \; x : \exists \in \mathbb{Q}^n.$$

A completely different proof based on game-theoretic arguments can be found in [212]. The fact that it suffices to check the bounding values of the universally quantified variables in order to answer the question whether the existential player can certainly win the game, can be exploited in terms of asking whether a *winning strategy* for the existential player does exist, as shown in Figure 3.8 for the QLP instance of Example 3, where the blue points are the games that result from the winning strategy. In Section 4.1.3 we will show that this result can be used to create a deterministic linear program of a QLP or QIP instance, similar as in the case of an MSSLP, and which can also be solved with standard LP and IP algorithms.

In the following, we argue that it also suffices to probe all possible combinations of the bounds of the universal player's variable ranges to solve a QLP with objective function as in Definition 2.15. As a preparation, we first show that the formulation

$$\begin{gathered} z_Q = \min_{x^1}(c^1 x^1 + \max_{x^2}(c^2 x^2 + \min_{x^3}(c^3 x^3 + \max_{x^4}(\ldots \min_{x^m} c^m x^m))))) \\ Q \circ x \in [l, u] : Ax \leq b \end{gathered} \tag{QLP$_o$}$$

is equivalent to the following formulation

$$\min\{k \,|\, (\exists, Q) \circ (k, x) \in [l', u'] \cap \mathbb{Q}^n : Ax \leq b \wedge c^T x \leq k\} \tag{QLP$_k$}$$

where the objective function is pushed into the constraint system and used to restrict an existentially quantified auxiliary variable $k \in \mathbb{Q}$ that is placed in front of the QLP and which shall be minimized. This means that similar as in the case of traditional linear programming, the objective function can be encoded in the constraint system.

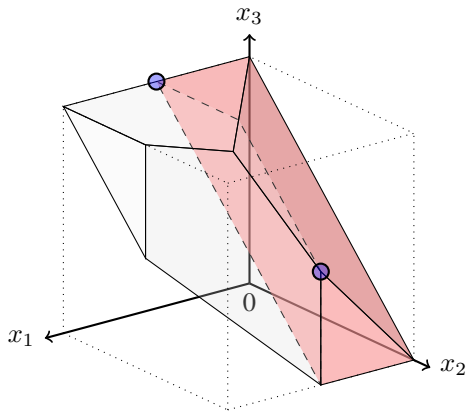


Figure 3.6.: Solution space of Example 2.

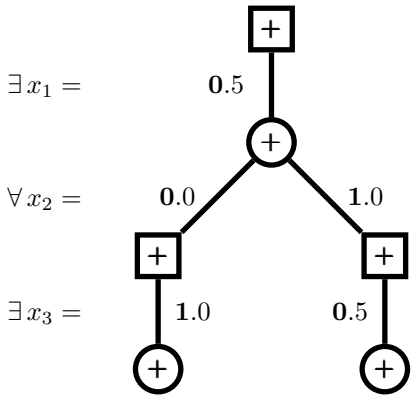


Figure 3.7.: Winning strategy for Example 2.

Lemma 3.6. *A strategy S is a winning strategy for QLP_o with objective function value z_Q if an only if S is also a winning strategy for QLP_k where $k = z_Q$ holds.*

Proof. Let S_0 be a winning strategy for QLP_k with k being minimal. From this follows that for each possible game x^{S_0} of S_0 the inequality $c^T x^{S_0} \leq k$ is satisfied at all leafs of S_0. However, since k is minimal at least one leaf satisfies $c^T x^{S_0} = k$ as depicted in Figure 3.8. The universal player can choose the path in S_0, but whatever he does, the existential player will certainly win the game and achieve an objective function value of at least k. To reach exactly k, the universal player must choose the path that leads to a leaf of S_0 where $c^T x^{S_0} = k$. Obviously, the minimax value of a strategy is equal to the maximum value at the leafs of a winning strategy (c.f. [171]), and therefore $k = z_Q$. Conversely let S_1 be a winning strategy for QLP_o and let

$$z = \min_{x^1}(c^1 x^{S_1 1} + \max_{x^2}(c^2 x^{S_1 2} + \min_{x^3}(c^3 x^{S_1 3} + \max_{x^4}(\ldots \min_{x^m} c^m x^{S_1 m})))).$$

Therefore S_1 consist of a set of games such that for each game x^{S_1} of S_1 $Ax^{S_1} \leq b$ and $c^T x^{S_1} \leq z$ at all leafs of S_1. Because the existential player minimizes the objective function, there must be at least one leaf in S_1 with $c^T x^{S_1} = z$. Therefore z is also minimal for QLP_k and $z = k$. □

To find the optimal minimax objective function value of a QLP instance, it still suffices to inspect only the bounds of the universally quantified variables (c.f. Lemma 3.7).

Lemma 3.7. *For each universally quantified variable x_i in QLP_k, it suffices to inspect the bounds l_i and u_i. Moves of the universal player in the range $l_i < x_i < u_i$ do not increase or decrease the value of k.*

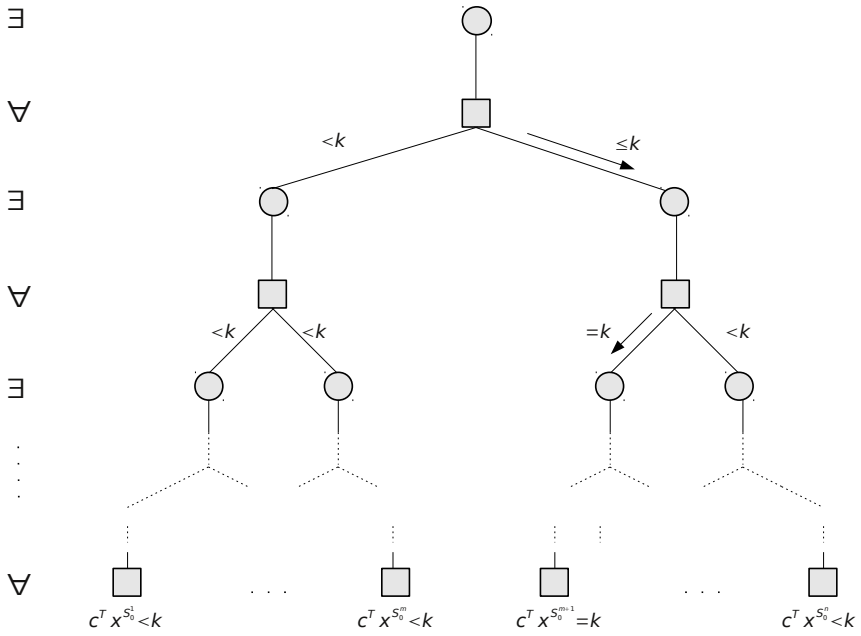


Figure 3.8.: Winning strategy S.

Proof. Let S be a winning strategy for

$$Q \circ x \in [l, u] : Ax \leq b \wedge c^T x \leq k.$$

Let w.l.o.g. all universally quantified variables be binary (since in the linear case $Ax \leq b$ can be rescaled to the effect that all variables are from a $[0, 1]$ interval). Let furthermore k be minimal for the QLP and let $c^T x = k$ in exactly one leaf of S as depicted in Figure 3.9. After the first move of the existential player where he fixes the first variable to $x_0 = \overline{x}_0$, the winning strategy S splits up into two sub strategies S_0 and S_1. These are winning strategies for the remaining moves of the existential player when the universal player fixes the next universally quantified variable x_2 to the lower bound $x_2 = 0$ or to the upper bound $x_2 = 1$. Let k_{S_0} be the corresponding maximal value $c^T x \leq k$ that can result at the leafs from all possible games $x^{S_0} = (\overline{x}_0, 0, x_2, \ldots, x_n)$ of S_0, and k_{S_1} be the same value with respect to possible games $x^{S_0} = (\overline{x}_0, 1, x_2, \ldots, x_n)$ of S_1 as depicted in Figure 3.9. Then it holds that $max\{k_{S_0}, k_{S_1}\} = k$. If the universal player is allowed to choose an $x_i = \alpha$ with $0 \leq \alpha \leq 1$ (e.g. $x_2 = \alpha$ as in Figure 3.9), we can construct a winning strategy S_α from S_0 and S_1 for any α and it holds that for all possible resulting games x^{S_α}, the corresponding values $c^T(\overline{x}_0, x_{S_\alpha})$ are less than k. To compute S_α, the existential player can compute x^α for any possible move sequence of the universal player by setting variable to

$$x_i^\alpha = \alpha x_i^{S_0} + (1 - \alpha) x_i^{S_1}.$$

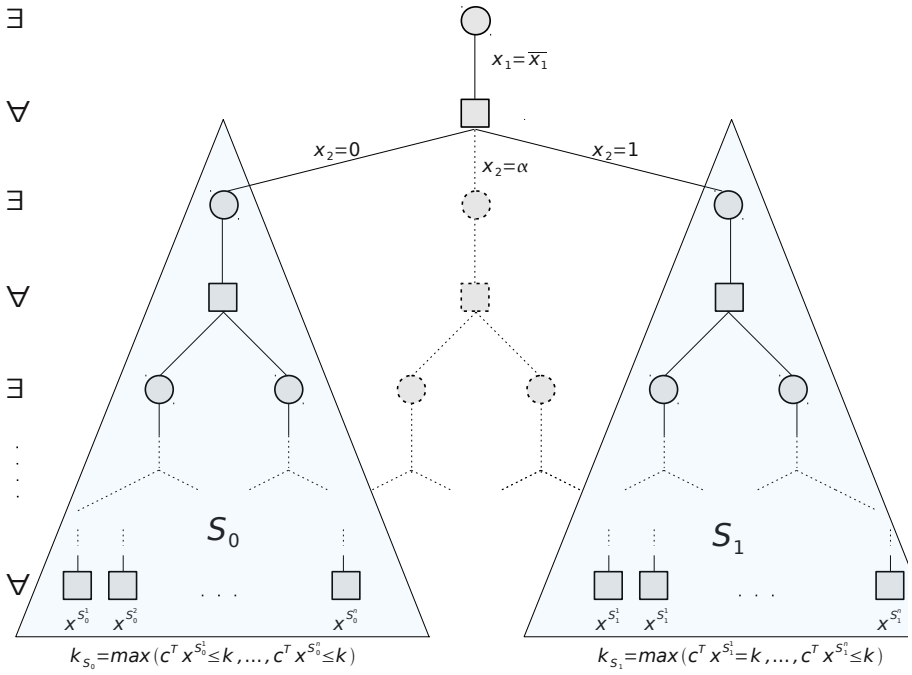


Figure 3.9.: Winning strategies S, S_0, S_1.

Then, for all possible games x^α in S_α holds,

$$A\overline{x}^\alpha = \alpha(Ax^0) + (1-\alpha)(Ax^1) \leq \alpha b + (1-\alpha b) \leq b,$$
$$c^T\overline{x}^\alpha = \alpha(c^T x^0) + (1-\alpha)(c^T x^1) = \alpha k_0 + (1-\alpha)k_1 \leq k,$$

and Lemma 3.7 follows using induction over the number of universally quantified variables. □

3.2.2. Polyhedral Properties

An important question for the development of algorithms to solve QLPs and QIPs is how their solution spaces can be characterized. The structure of the solution space of a QLP is not exactly known, and only implicitly given, the inequalities of the constraint system only specify the polyhedral solution space $\mathcal{P}_{LP} = P(A, b)$ of its LP-relaxation, which is very similar to the situation we are confronted in classical integer programming. There, the knowledge of the polyhedral properties of their LP-relaxations is of particular importance and can be used in the solution process. It seems natural that QLP-relaxations can affect the solution process of QIPs in a similar way. However, therefore we need a better understanding of their polyhedral properties. As we have already seen in the previous examples, these differ considerably from the solution spaces of their corresponding LP-relaxations. Consider also the following example for clarification.

Example 8. *The QLP*

$$\forall\, x_1 \in [0,1]\, \forall\, x_2 \in [0,1]\, \exists\, x_3 \in [0,1] :$$

$$\begin{pmatrix} -1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

has only one quantifier change. The problem is feasible, since the projection of the constraint polytope to the x_1-x_2 plane results in the unit square. Therefore, the solution space coincides with the whole constraint polytope, as Figure 3.10 shows. But note that - even though the solution space is polyhedral - there exists no linear function $x_3(x_1, x_2)$ for choosing the existential variable x_3

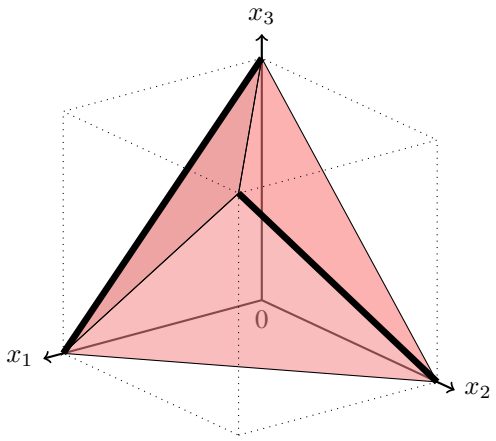


Figure 3.10.: A solution space with a linear description, where x_3 cannot be chosen linear in x_1 and x_2. (Note the bold lines are not in the same hyperplane.)

In the example above, the solution space is a polyhedron, although there is no linear description of a single solution. Also the graphical representations of former examples encouraged the expectation that the solutions spaces of QLPs are polyhedral, and below, we show that this assumption indeed holds in general [147].

In the following, let P be a winning policy for a given QLP instance. We speak of a *game of* P if this game can be generated by the policy P. Each game of P is identified with a point in $\mathbb{Q}^n$. We claim that the set of all points, which results from the union of all games from all existing winning policies, forms a polytope. The convex hull of these point is called the *solution polytope* or *solution space* of the QLP and is denoted by $\mathcal{P}_{QLP} = P_Q(Q \circ x \in [l,u], A, b)$ in the following.

Proposition 3.8 (Solution Polytopes)**.** *The solution spaces of quantified linear programs are polyhedral.*

Proof. Without loss of generality, let the quantifier sequence be alternating beginning with an existential quantifier. We again use an inductive argument and start by eliminating all variables except x_1 with the QLP elimination procedure. If there is no winning policy at all, the resulting solution space is empty, and thus a polytope. Otherwise, this process supplies us with two bounds $\bar{l}_1$ and $\bar{u}_1$ for x_1. That is, the process changes two implicit strategic constraints to explicit constraints. We know that an algorithm choosing $x_1 \in [\bar{l}_1, \bar{u}_1]$ can be extended to a winning policy. It can even be extended for every universal move of x_2. Therefore the rectangle created by $\bar{l}_1 \leq x_1 \leq \bar{u}_1$ and $l_2 \leq x_2 \leq u_2$ has the property that all points inside the rectangle can be interpreted as the beginning of a game of a winning policy, but no point outside the rectangle can be extended to a game of a winning policy. Let us collect the constraints for x_1 in D_1 and those for x_2 in D_2. Let us collect all constraints that remain after eliminating all variables except x_1 in the set D_1, and respectively, all constraints that remain after eliminating all variables except x_1 and x_2 in the set D_2.

In a next step, we pick up the original QLP instance and eliminate all variables except x_1, x_2 and x_3. As a result, we get a set D_3 of new constraints for x_3 of the form $x_3 \leq \bar{b} - \bar{a}_1 x_1 - \bar{a}_2 x_2$, with $\bar{a}_1$, $\bar{a}_2$ and $\bar{b}$ being coefficients resulting from the elimination procedure. All games which start with $x_1 \in D_1$, $x_2 \in D_2$, $x_3 \in D_3$ can be extended to a winning policy and again, it does not matter how the universal player chooses $x_4 \in D_4$. Moreover, there are no other game starts of winning policies: Every game that violates at least one of the constraint sets $D_1, \ldots, D_4$ cannot belong to a winning policy. Continuation of the process leads to the desired result.

□

Now we know that the solution space $\mathcal{P}_{QLP}$ is a polytope in $\mathbb{Q}^n$ and it is a subset of the solution space of its LP-relaxation, $\mathcal{P}_{QLP} \subseteq \mathcal{P}_{LP} \subseteq \mathbb{Q}^n$. In Section 4.6 we show that under certain conditions the vertices of this polytope can be described with polynomially many bits.

Similar as with traditional linear programs and integer programs we can define the integral convex hull of the QLP solution space $conv(\mathcal{P}_{QLP} \cap \mathbb{Z}^n)$. However, in contrast to the traditional case, where each integral vertex of the integral convex hull is a solution for the corresponding IP, $conv(\mathcal{P}_{QLP} \cap \mathbb{Z}^n)$ can contain vertices that are not part of the solution space of the QIP that results from all games of all winning strategies of the QIP and can be denoted by $\mathcal{P}_{QIP}$. However, although the actual structural properties of $\mathcal{P}_{QIP}$ are unknown, the following relationship also holds in the quantified case:

$$\mathcal{P}_{QIP} \subseteq conv(\mathcal{P}_{QLP} \cap \mathbb{Z}^n) \subseteq \mathcal{P}_{QLP}. \tag{3.15}$$

3.3. Relaxations

Most combinatorial decision and optimization problems are solved by a combination of enumeration and polyhedral methods. The effectiveness of partial enumeration when solving IPs with a branch-and-bound approach are closely related to the quality of the relaxations used to generate the bounds. The latter typically involves repeated solution of

variants of the problem's linear programming relaxation [13]. In this section we introduce certain types of relaxations of quantified linear and integer programs. While some of the proposed relaxations are familiar from classical linear (integer) programming, some of them are particular to the QLP framework and have no direct equivalence in the unquantified case. Opposed to traditional LP-relaxations, these relaxations do not only result when the integrality properties of the variables are removed, but when universal quantification is dropped or the order of the variables in the quantification sequence is changed. The resulting problems allow to draw conclusions about the feasibility of the original QLP and QIP problems and they can be used to compute lower and upper bounds for the objective function.

As in the traditional case we can think of relaxing the integrality of variables of an IP, resulting in a LPs. Enlarging the domain of quantified variables by relaxing the integrality property as proposed in Definition 2.5 is the direct equivalence to the LP-relaxation of an IP in the classical case (see Definition 1.12). The following relaxation properties already known from the unquantified case also hold for the QLP-relaxation of a QIP instance.

Proposition 3.9. *Given a QIP problem and its QLP-relaxation. If the QLP is infeasible, then also the QIP is infeasible.*

Proof. From a game-theoretic viewpoint, where existential variable relaxation can be interpreted as enlarging the set of possible moves for the existential player, it is easy to see that there definitely cannot be a winning strategy for a quantified integer program if the corresponding continuous program had no winning policy. □

From the above deliberations directly follows:

Corollary 3.10. *Given a QIP minimization problem and its QLP-relaxation with optimal objective function value z_{QLP}. If the QIP is feasible, then z_{QLP} is a lower bound for z_{QIP}.*

A useful viewpoint when considering relaxations of quantified linear programs is from the two-person game perspective between the existential and the universal player as mentioned in Section 2.1. In these terms we can regard the relaxation of quantifiers or the modification of their order in the quantification vector of a QLP as modifications that makes it easier or harder for the existential player to eventually win the game. The Relaxation of quantifiers from universal to existential corresponds to achieving the control over a variable previously considered to be controlled by the universal player. Accordingly, we use the term *LP-relaxation* in the context of QLPs to denote that universal quantification is dropped and all universal variables are treated as existential variables.

Definition 3.4 (LP-relaxation of QLP)**.**
Given a QLP instance

$$z_{QLP} = \min\{c^T x \mid Q \circ x \in [l, u] : Ax \leq b\},$$

its LP-relaxations is defined as follows:

$$z_{LP} = \min\{c^T x \mid \exists^n \circ x \in [l, u] \cap \mathbb{Q}^n : Ax \leq b\}.$$

From the viewpoint of a two-person game, this means that the existential player may choose the moves of the universal player and thus can choose the move sequence until the end of the game. As with QIPs and their QLP-relaxations, also the LP-relaxation of a QLP yields the relaxation properties as mentioned above.

Proposition 3.11. *Given a QLP problem and its LP-relaxation. If the LP is infeasible, then also the QLP is infeasible.*

Proof. Since $\mathcal{P}_{QLP} \subseteq \mathcal{P}_{LP}$, $\mathcal{P}_{LP} = \emptyset \implies \mathcal{P}_{QLP} = \emptyset$. Alternatively we can use the following game-theoretic argument. When universal quantification is dropped, the existential player makes the moves of the universal player, e.g., he fixes all variables $x_\forall \in [l_\forall, u_\forall]$. Of course, if the existential player cannot choose his moves and the moves of the universal player, such that the constraint system $Ax \leq b$ can be satisfied, then he also won't be able to do so if the universal player chooses the moves that correspond the universal variables $x_\forall$ in an adversarial manner. □

Proposition 3.12. *Given a QLP minimization problem and its LP-relaxation with optimal objective function value z_{LP}. If the QLP is feasible, then z_{LP} is a lower bound for z_{QLP}.*

Proof. Since $\mathcal{P}_{QLP} \subseteq \mathcal{P}_{LP}$ it directly follows that $z_{LP} \leq z_{QLP}$. Again, we can use the above gaming argument. When the existential player can choose the moves of the universal player and he can choose the best move sequence from his viewpoint, of course the optimal value of the minimization problem is a lower bound on the minimax value of the corresponding QLP when the universal player can choose in an adversarial manner. □

Figure 3.11 illustrates this for Example 3. The QLP solution space (red colored polytope) is a subset of the LP solution space (polytope encompassed by solid lines). The convex hull of the integral points of the QLP solution space, which correspond to the two winning path of the winning strategy as depicted in Figure 2.2, forms the blue colored line segment. furthermore assume the following objective function $\min -x_1 - x_3$, then the optimal solution of the corresponding LP is $z_{LP} = -2$, which corresponds to the solution $x = (1.0, 0.0, 1.0)^T$. z_{LP} is a lower bound for the optimal QLP solution $z_{QLP} = -1$, which corresponds to the solution $x = (0.5, 1.0, 0.5)^T$. z_{LP} and z_{QLP} themselves are lower bounds for the optimal QIP solution $z_{QIP} = 0$, which corresponds to the solution $x = (0.0, 1.0, 0.0)^T$. Another special form of relaxation restricted to quantified linear and integer programs is *quantifier shifting*. The order of the elements in the quantification vector $Q \circ x \in [l, u]$ is crucial, because it determines the sequence in which the variables must be fixed by the two players. However, changing the order in the quantification sequence can yield interesting information about the original problem. In this context, the two special cases when the universal quantified variables are shifted to the end of the quantification sequence, and respectively, when they are shifted to the front are of high interest.

Definition 3.5 ($\exists\forall$-relaxation of a QLP)**.** *Given a QLP as in Definition 2.15*

$$z_Q = \min\{c^T x \mid Q \circ x \in [l, u] \cap \mathbb{Q}^n : Ax \leq b\}, \tag{QLP}$$

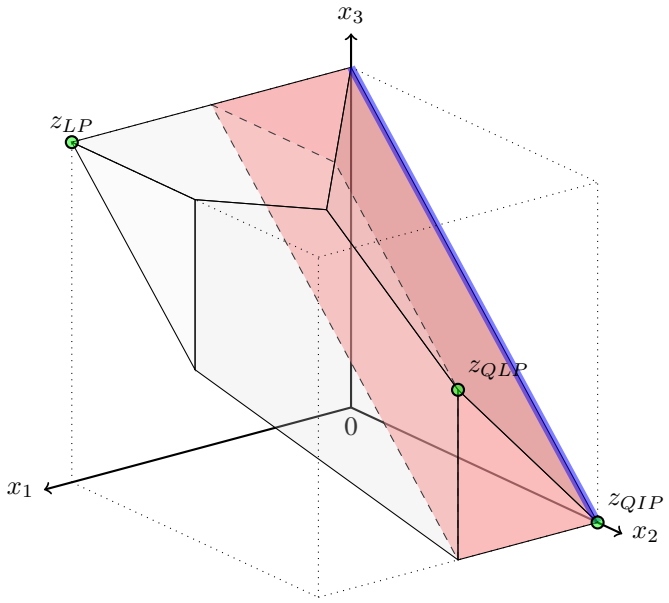


Figure 3.11.: Solution spaces ($\mathcal{P}_{QIP} \subseteq \mathcal{P}_{QLP} \subseteq \mathcal{P}_{LP}$) for Example 3 and optimal solution coordinates for $\min -x_1 - x_3$ with $z_{LP} \leq z_{QLP} \leq z_{QIP}$.

with arbitrary quantification vector $Q \circ x \in [l, u]$. The problem

$$z_{\exists\forall} = \min\{c^T x \,|\, \exists \circ x_\exists \in [l_\exists, u_\exists] \cap \mathbb{Q}^n \, \forall \circ x_\forall \in [l_\forall, u_\forall] \cap \mathbb{Q}^n : \, Ax \leq b\}, \quad \text{(QLP}_{\exists\forall}\text{)}$$

where the quantification sequence is re-ordered to the effect that all existential variables precede the universal ones, is called its $\exists\forall$-relaxation.

Definition 3.6 ($\forall\exists$-relaxation of a QLP)**.** *Given a QLP as in Definition 2.15*

$$z_Q = \min\{c^T x \,|\, Q \circ x \in [l, u] \cap \mathbb{Q}^n : Ax \leq b\}, \quad \text{(QLP)}$$

with arbitrary quantification vector $Q \circ x \in [l, u]$. The problem

$$z_{\forall\exists} = \min\{c^T x \,|\, \forall \circ x_\forall \in [l_\forall, u_\forall] \cap \mathbb{Q}^n \, \exists \circ x_\exists \in [l_\exists, u_\exists] \cap \mathbb{Q}^n : \, Ax \leq b\}, \quad \text{(QLP}_{\forall\exists}\text{)}$$

where the quantification sequence is re-ordered to the effect that all universal variables precede the existential ones, is called its $\forall\exists$-relaxation.

This means, in the case of QLP$_{\forall\exists}$ it suffices to find for any possible fixation $\overline{x}_\forall$ of the universally quantified variables, a fixation $\overline{x}_\exists$ of the existentially quantified variables such that $Ax \leq b$. In the case of QLP$_{\exists\forall}$ in contrast, we must find *one single* fixation $\overline{x}_\exists$ of the existentially quantified variables such that $Ax \leq b$ holds *for all* possible fixations $\overline{x}_\forall$ of the universally quantified variables. Judging from the viewpoint of game playing, the latter is of course much harder to achieve for the existential player. For the following propositions

and the corresponding proofs, we first introduce the concept of break vectors.

Definition 3.7 (∃-Break-Vector)**.** *A fixed vector $\overline{x}_\exists$ is said to be a ∃-break-vector for the existential player, if the resulting system*

$$\forall \circ x_\forall \in [l_\forall, u_\forall] : \; A_\forall \cdot x_\forall \leq \underbrace{b - A_\exists \cdot \overline{x}_\exists}_{:= b(\overline{x}_\exists)}$$

is feasible for any vector $x_\forall \in [l_\forall, u_\forall]$.

Definition 3.8 (∀-Break-Vector)**.** *A fixed vector $\overline{x}_\forall$ is said to be a ∀ break vector for the universal player, if the resulting system*

$$\exists \circ x_\exists : \; A_\exists \cdot x_\exists \leq \underbrace{b - A_\forall \cdot \overline{x}_\forall}_{:= b(\overline{x}_\forall)}$$

has no feasible solution at all.

Thus, an ∃-break-vector can be interpreted as an *certificate of feasibility* for the original QLP, a ∀-break-vector can be interpreted as a *certificate of infeasibility* for the original QLP respectively.

Proposition 3.13. *Let a QLP and its ∃∀-relaxation QLP$_{\exists\forall}$ be given. If QLP$_{\exists\forall}$ is feasible with solution vector $\overline{x}_\exists$, then also the QLP is feasible, and $\overline{x}_\exists$ is a ∃-break-vector for the QLP.*

Proof. QLP$_{\exists\forall}$ is solved with the help of a gaming argument (see e.g. Algorithm 2) [212]. Therefore, the constraint system $Ax \leq b$ is converted into a modified constraint system where the universal player has replaced all variables x_i from the set $x_\forall$ by the worst-case for the existential player between l_i and u_i *inequality by inequality* and also in the objective function. The modified constraint system does not longer contain any universally quantified variables and can be solved with standard linear programming algorithms to obtain the solution $\overline{x}_\exists$. This solution is also a solution for QLP$_{\exists\forall\ldots\forall\exists}$ because all variables x_i from the set $x_\forall$ must still be chosen between l_i and u_i, however not worst case for the existential player inequality by inequality but rather with one single fixation for all constraints in $Ax \leq b$ for each variable x from the set $x_\forall$. □

Proposition 3.14. *Let a QLP and its ∀∃-relaxation QLP$_{\forall\exists}$ be given. If QLP$_{\forall\exists}$ is infeasible for any fixed vector of universally quantified variables $\overline{x}_\forall$, then also the QLP is infeasible, and $\overline{x}_\forall$ is a ∀-break-vector for the QLP.*

Proof. If there is any move sequence $\overline{x}_\forall$ for the universal player in QLP$_{\forall\exists}$ such that

$$\exists \circ x_\exists : \; A_\exists \cdot x_\exists \not\leq b - A_\forall \cdot \overline{x}_\forall$$

is not feasible for any $x_\exists$, then then $\overline{x}_\forall$ is a ∀ break vector for the original QLP$_{\exists\forall\ldots\forall\exists}$ (Note that the inversion of the argument is not correct). □

Proposition 3.15. *Given a feasible QLP minimization problem and assume that its $\exists\forall$-relaxation QLP$_{\exists\forall}$ is also feasible, with $\overline{z}_{\exists\forall}$ being the corresponding optimal objective function value. Then $\overline{z}_{\exists\forall}$ is an upper bound for the optimal objective function value $\overline{z}$ of the QLP and it holds $\overline{z} \leq \overline{z}_{\exists\forall}$.*

Proof. The optimal solution z^* of QLP$_{\exists\forall\ldots\forall\exists}$ cannot exceed $z^*_{\exists\forall}$ because all constraints of QLP$_{\exists\forall}$ are also part of QLP$_{\exists\forall\ldots\forall\exists}$. However, because the universal player may not choose his variables worst case for the existential player inequality by inequality but rather with one single fixation for each variable x_i from the set $x_\forall$ for all constraints in $Ax \leq b$, some of the constraints in QLP$_{\exists\forall\ldots\forall\exists}$ get weakened with respect to their counterparts in QLP$_{\exists\forall}$ and thus, the objective function value cannot get worsened. □

Proposition 3.16. *Let a feasible QLP minimization problem and its $\forall\exists$-relaxation QLP$_{\forall\exists}$ be given, with $\overline{z}_{\forall\exists}$ being the optimal objective function value of the worst-case scenario. Then $\overline{z}_{\forall\exists}$ is a lower bound for the optimal objective function value $\overline{z}$ of the QLP and it holds $\overline{z}_{\forall\exists} \leq \overline{z}$.*

Proof. If QLP$_{\forall\exists}$ is feasible with $\overline{z}_{\forall\exists}$ being the optimal value of the worst-case scenario, then the optimal solution $\overline{z}$ of QLP$_{\exists\forall\ldots\forall\exists}$ can not become less than $\overline{z}_{\forall\exists}$ because the corresponding worst-case scenario is also part of QLP$_{\exists\forall\ldots\forall\exists}$. Apart from that, $\overline{z}$ can become a sight worse than $\overline{z}_{\forall\exists}$ because in QLP$_{\exists\forall\ldots\forall\exists}$ some variable fixations of the existential player most hold for more than one scenario, whereas for QLP$_{\forall\exists}$ the opposite is the case. □

As we will see in Section 5.2, the information from QLP-relaxations can be exploited in the solution process of a QLP, e.g. in a similar way the LP-relaxation of an IP is used in modern branch-and-bound algorithms.

4. Algorithms and Complexity

In this section we first review existing solution approaches to solve quantified linear and integer optimization problems in Section 4.1. Section 4.2 proceeds with a detailed complexity analysis for the continuous and integer case.

4.1. Algorithms to solve QLPs and QIPs

This section reviews algorithms to solve QLP and QIP optimization problems. When solving a QLP optimization problem, we are interested in the following information, assuming a QLP minimization problem that starts with an existential variable block. First of all, is the problem instance feasible at all? Second, if is is feasible, what is the optimal value of the objective function and which first-stage assignment of the existential variables guarantees to obtain at least this value? Note that the optimal minimax value of an objective function is an upper bound for the minimization problem assuming the worst-case scenario resulting from the possible universal variable assignments (e.g. the universal player plays perfect). However, this value might become significantly better, depending on the eventual universal variable assignment.

4.1.1. The Quantifier Elimination Algorithm

The QEA-method (see Algorithm 3) only decides the feasibility of a QLP instance. In order to solve QLP optimization problems and to obtain assignments of the existential variables in the first variable block, we propose the following slight modification. First, given a QLP minimization problem with existential variables in the first-stage

$$z_{\mathcal{Q}} = \min\{c^T x \,|\, Q \circ x \in [l,u] : Ax \leq b\}, \tag{QLP$_O$}$$

we use Lemma 3.6 and rewrite it as

$$\min\{k \,|\, (\exists, Q) \circ (k,x) \in [l',u'] \cap \mathbb{Q}^n : Ax \leq b \wedge c^T x \leq k\}, \tag{QLP$_k$}$$

where the bounds of k are sufficiently large (they can be computed using the variable bounds and the objective function of QLP$_O$). Second, we modify the QEA-method to the effect that only the variables up to the first stage are eliminated. Thus the variables $(k, x^1_\exists)$ remain, and the system that results after the elimination steps can be written as

$$z_k = min\{k \,|\, B^T(k, x^1_\exists) \leq d, x \in \mathbb{Q}^{n_1}\}, \tag{LP$_k$}$$

which is a traditional linear program that can be solved with standard LP algorithms. If LP_k is feasible, with an optimal objective function value $\overline{z}_k$ and corresponding variable assignment $\overline{x}^1_\exists$, this is also the optimal solution for QLP_O.

4.1.2. The Minimax-Algorithm and Alpha-Beta Pruning

Almost all game playing programs use a game-tree to represent positions and moves. The root node corresponds to the current position of the game, inner nodes represent possible future game positions, the branches of a node represent the legal moves from the position represented by the node. Leaf nodes represent end positions of the game and can be evaluated as a win, a lose, a draw, or a specific score, using the rules of the game.

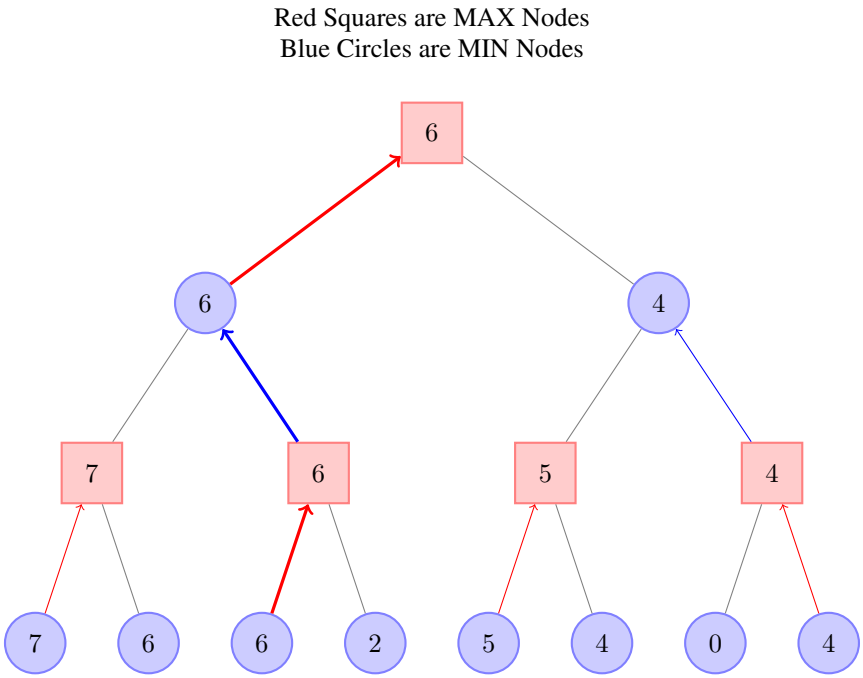


Figure 4.1.: Evaluated minimax-tree.

For traditional integer programming, where each variable can only take a finite number of discrete values, we can find an optimal solution by a complete enumeration of all possibilities represented in term of a game-tree that only consists of existential nodes (see Definition 2.13). Finding the optimal solutions can be achieved by calculating the value of the objective function at each leaf of the game-tree and choosing the feasible one with the optimal value from the viewpoint of the objective. This solution then corresponds to a single path from the root to a leaf. To solve a QIP in contrast, we can also create the corresponding game-tree by complete enumeration of all possibilities in the same way. However, it does not suffice to find a single (optimal) path from the root to a leaf, but we must check whether a winning strategy for the existential player does exist or not (as men-

tioned in Section 2.1.3). Then, if there is more than one winning-strategy, we can use the objective function to find a certain best one. Up to now, we did not explicitly show up a way how to find such a winning strategy, although the basic idea of the *Minimax*-Algorithm was already mentioned in the context of the minimax value of a winning strategy (see Definition 2.17). Minimax techniques have been used for a long time in the gaming community to solve two-player games, and some of the techniques can be directly applied to solve QIP decision and optimization problems.

A *minimax-tree* is one of the most important data structures that allows computers to play two-person zero-sum games such as checkers, chess, and go. Nodes of the tree are decision points for the two players and are therefore subdivided in MIN and MAX nodes. Nodes from different stages are connected with branches and represent the moves of the corresponding players, leaf nodes are end positions of the game and can be evaluated as a win, lose, or draw using the rules of the game. Often, a specific score from the MAX player's point of view is computed with the help of a weighting function and assigned to a leaf to represent how good or bad the sequence of moves from the root to the leaf is (of course this can also be done from the viewpoint of the MIN Player). With a complete game-tree, it is possible to solve the game with the Minimax-Algorithm (see Algorithm 4), which fills the inner node values of the tree bottom-up starting with the evaluated values at the leafs. Nodes that belong to the MAX player get the maximum values of their successors, while nodes for the MIN player get the minimum. Figure 4.1 illustrates this behavior.

Algorithm 4 Minimax(Node v)

```
1: if (v is a leaf) then
2:     /* Calculate the Value of this node with the weighting function f(v) */
3:     h^v := f(v);
4:     return h^v;
5: end if
6: if (v is a MAX node) then
7:     /* Determine the maximum of all successors */
8:     return max {Minimax(v') for all v' ∈ Γ(v)}
9: end if
10:
11: if (v is a MIN node) then
12:     /* Determine the minimum of all successors */
13:     return min {Minimax(v') for all v' ∈ Γ(v)}
14: end if
15:
```

Algorithm 4 is the same procedure as proposed in Section 2.1.3 to compute the minimax value (see Definition 2.17) of a winning strategy, now we also use it to find such a winning strategy. Thus, in order to solve a QIP instance with Algorithm 4 we simply assign the $\exists$-player and the $\forall$-player to the MIN and MAX nodes as follows

- MIN: the $\exists$-player is setting the variables $x_\exists$,
- MAX: the $\forall$-player is setting the variables $x_\forall$,

and use the following weighting function at the leaf nodes

$$f(x) = \begin{cases} c^T x & \text{if } x \text{ is winning for the existential player,} \\ +\infty & \text{if } x \text{ is winning the universal player,} \end{cases}$$

where x is the set of moves from the root to the current leaf. Thus, after the game-tree of a QIP instance has been evaluated with the Minimax-Algorithm, the instance is feasible if the corresponding minimax value at the root is less than $+\infty$, which in this case also is the optimal objective function value. The corresponding outgoing edge encodes the first stage variable allocation. (Note that if the quantifiers in the QIP instance are not strictly alternating, each outgoing edge encodes a set of moves, which correspond to a specific variable block.)

Algorithm 5 AlphaBeta(Node v,α,β)

```
1: if (v is a leaf then
2:    /* Calculate the Value of this node with the weighting function f(v) */
3:    h^v := f(v);
4:    return h^v;
5: end if
6: if (v is a MAX node) then
7:    for all v' ∈ Γ(v) do
8:       α := max {α, AlphaBeta(v', α, β)}
9:       if (α ≥ β) then
10:         return β;
11:      end if
12:   end for
13:   return α;
14: end if
15: if (v is a MIN node) then
16:   for all v' ∈ Γ(v) do
17:      β := min {β, AlphaBeta(v', α, β)}
18:      if (β ≤ α) then
19:         return α;
20:      end if
21:   end for
22:   return β;
23: end if
24:
```

A drawback of this solution approach is that the entire game-tree must be evaluated. In current state-of-the-art MIP-solvers, the most important components are branch-and-cut procedures and LP-relaxations with the task to prune the search-tree as much as possible. The counterpart of such a branch-and-bound procedure for QIPs is the *Alpha-Beta Algorithm* [171, 135] as described in Algorithm 5. While the Minimax-Algorithm must evaluate the entire game-tree to compute the root value, the Alpha-Beta Algorithm prunes away branches of the tree that cannot influence the final result. It therefore applies a branch-and-bound technique and finds the minimax equivalent solution without visiting every node in general.

Therefore, it maintains two values α and β, which are updated while the tree is traversed. The semantics of α and β are as follows:

- The value of α at a given node in the game-tree is equal to the maximum value of all predecessor nodes of type MAX that were already visited.
- The value of β at a given node in the game-tree is equal to the minimum value of all predecessor nodes of type MIN that were already visited.
- During the runtime of the Alpha-Beta Algorithm the values of α and β may change, but the value for α will not decrease and the value for β will not increase.
- The minimax value of a node cannot fall below α and cannot exceed β.

Thus, at a given node in the game-tree, these values represent the minimum score that the MAX player is sure to gain at least until that point in the tree, and the maximum score of the MIN player respectively. If the evaluation of a position where the MIN player has to move becomes less than α, the remaining moves at this node need not to be further explored, since a better move has already been found.

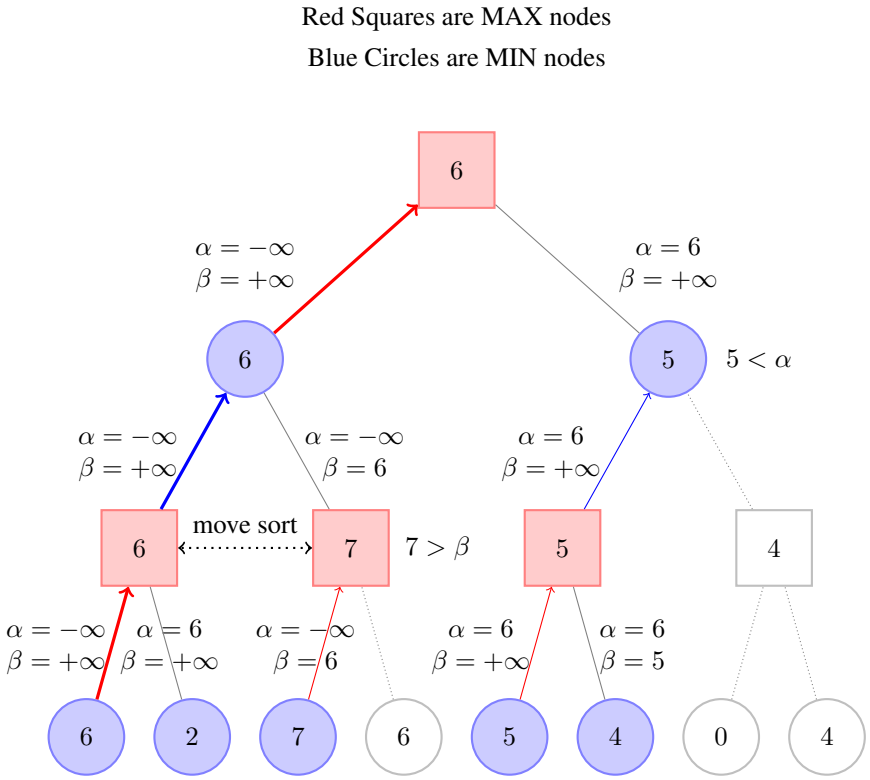


Figure 4.2.: Minimax with Alpha-Beta-Pruning and Move-Sort.

While the order in which the nodes of the tree are evaluated does not matter for the Minimax-Algorithm, it is essential for the performance of the Alpha-Beta Algorithm. The

best moves need to be evaluated first in order to find strong α and β values as soon as possible. Figure 4.2 illustrates this, without swapping the subtrees under the first successor of the root on the left side, the β-cutoff would not have occurred. If the best moves are searched first, the runtime of the Alpha-Beta Algorithm is only $O(\sqrt{b^d})$ where d is the depth of the tree and b is the number of possible moves at each node [171]. The Minimax-Algorithm has a runtime of $O(b^d)$.

The question how information from the solution of the corresponding QLP-relaxation (e.g. lower bounds or cutting planes) can be used to speedup the solution process of QIPs is unknown. However, it is subject of current research, e.g. in the DFG project LO 1396/2–1. In Section 5.2.3 we show how the concept of $\alpha\beta$-pruning and move-ordering can be used to speedup the solution process of multi-stage QLPs.

4.1.3. The Deterministic Equivalent Linear Program

In the following we show how a QLP can be converted into a deterministic equivalent linear program (DEP). As a preparation, let's recall the notion of deterministic equivalence known from stochastic programming (see Section 2.2.2), where many solution methods start by formulating a DEP (see e.g. [47]). The basic assumption therefore is that the realizations of the random parameters can be specified in form of a set of finitely many scenarios. Therefore, when considering a multi-stage stochastic linear program (MSSLP), it is usually assumed that the stochastic elements are defined over a discrete probability space and that the time horizon is finite. This allows to represent a MSSLP in terms of a scenario-tree that can be encoded into a deterministic equivalent linear program (DEP) by replicating the underlying deterministic part for each possible scenario.

In the following we show that in a very similar way we can construct the DEP of a QLP. To fix our notation for the algorithm development and as an introduction for the following sections, we start by a node-based recursive definition for quantified linear programs, similar as it was done for MSSLPs in Section 2.2.2 and use the notation as presented in [47]. A two-stage QLP with an existential variable block at the beginning and at the end can be defined as follows:

$$\min_{x^1_\exists}\left((c^1_\exists)^T x^1_\exists + \max_{x^1_\forall}\left((c^1_\forall)^T x^1_\forall + \min_{x^2_\exists}\left((c^2_\exists)^T x^2_\exists\right)\right)\right)$$

$$\exists\ \circ x^1_\exists\ \forall \circ x^1_\forall \in [l^1_\forall, u^1_\forall]\ \exists \circ x^2_\exists :$$

$$s.t.\ \ W^1_\exists x^1_\exists \le h^1 \tag{4.1}$$

$$T^1_\exists x^1_\exists + T^1_\forall x^1_\forall + W^2_\exists x^2_\exists \le h^2$$

$$l^1_\exists \le x^1_\exists \le u^1_\exists,$$

$$l^2_\exists \le x^2_\exists \le u^2_\exists, \tag{4.2}$$

where the constraint system $Ax \le b$ is splitted into variable blocks by stage and quantifier. $W^1_\exists$ is the deterministic first-stage part of the constraint system that is not affected by

universal variables in the second stage. This can be rewritten as

$$\begin{aligned} \min_{x^1_\exists} \ & (c^1_\exists)^T x^1_\exists + \mathcal{Q}^1(x^1_\exists) \\ s.t. \ & W^1_\exists x^1_\exists \leq h^1 \\ & l^1_\exists \leq x^1_\exists \leq u^1_\exists, \end{aligned}$$

where $\mathcal{Q}^2(x^1_\exists)$ is the worst-case second-stage value function, defined as

$$\mathcal{Q}^1(x^1_\exists) = \max_{\forall x^1_\forall \in E} [Q^2(x^1_\exists, x^1_\forall)],$$

which is the maximum loss that can result from the universal variable values from the set $E = [l^1_{\forall 1}, u^1_{\forall 1}] \times [l^1_{\forall 2}, u^1_{\forall 2}] \times \cdots \times [l^1_{\forall k}, u^1_{\forall k}]$, which is the k-dimensional hyperrectangle with $k = |x_\forall|$. Given a first-stage $x^1_\exists$ and $x^1_\forall$, the second stage can be written as

$$\begin{aligned} Q^2(x^1_\exists, x^1_\forall) = (c^1_\forall)^T x^1_\forall + \min_{x^2_\exists} \ & (c^2_\exists)^2 x^2_\exists \\ s.t. \ & W^2_\exists x^2_\exists \leq \underbrace{h^2 - T^1_\exists x^1_\exists - T^1_\forall x^1_\forall}_{:= h^2(x^1_\exists, x^1_\forall)} \\ & l^2_\exists \leq x^2_\exists \leq u^2_\exists. \end{aligned}$$

To expand this formulation recursively to multi-stage QLPs, the prior formulation of the second-stage value function must be extended for arbitrary stages as follows

$$\mathcal{Q}^{t+1}(x^t_\exists, x^{t-1}_\forall) = \max_{\forall x^t_\forall \in E^t} [\, Q^{t+1}(x^t_\exists, (x^{t-1}_\forall, x^t_\forall))] \text{ for } t = 2, \ldots, H-1.$$

The t-stage value function can be written as

$$\begin{aligned} Q^t(x^{t-1}_\exists, x^{t-1}_\forall) = (c^t_\forall)^T x^t_\forall + \min_{x^t_\exists} \ & (c^t_\exists)^T x^t_\exists + \mathcal{Q}^{t+1}((x^{t-1}_\exists, x^t_\exists), x^{t-1}_\forall) \\ s.t. \ & W^t_\exists x^t_\exists \leq h^t(x^{t-1}_\exists, x^{t-1}_\forall) \\ & l^t_\exists \leq x^t_\exists \leq u^t_\exists, \end{aligned}$$

where the input parameters $x^{t-1}_\exists$ and $x^{t-1}_\forall$ indicate the current state of the system at stage $t-1$, e.g. the history of all existential and universal moves from the stage 1 up to stage t. At stage $t = 1$ the second parameter of function $\mathcal{Q}^1(x^1_\exists, x^1_\forall)$ is an empty vector of universal variables. Finally, at the terminal nodes, the value function can be written as

$$\begin{aligned} Q^H(x^{H-1}_\exists, x^{H-1}_\forall) = (c^H_\forall)^T x^H_\forall + \min_{x^H_\exists} \ & c^H_\exists x^H_\exists \\ s.t. \ & W^H_\exists x^H_\exists \leq h^H(x^{H-1}_\exists, x^{H-1}_\forall) \\ & l^H_\exists \leq x^H_\exists \leq u^H_\exists. \end{aligned}$$

Using the recursive node-based definition, it is easy to see that a multi-stage QLP has very strong similarities to a MSSLP with fixed recourse and variable right-hand side. An important difference is that while a MSSLP minimizes the expected value of an objective function with respect to a set of scenarios, a QLP minimizes an objective function with respect to the possible worst case (maximum loss) that can result from a set of scenarios.

Whereas in a MSSLP the assumption of a finite time horizon and a discrete finite probability distribution can be used to create a scenario-tree, which defines the possible sequence of realizations over all time-stages, we have a similar situation with QLPs. A QLP instance has only a finite number of quantifier changes and by Corally 3.5 we need only check a finite set of scenarios, which result from the $2^{|x_\forall|}$ vertices of the hyperrectangle E, the set of all upper and lower bound combinations of the universal variables. Because the situation is sufficiently similar, we can create the equivalent of a scenario-tree for a QLPs and use the term *decision-tree* of the universal player or *QLP decision-tree* as an analogue for the scenario-tree of a MSSLP.

Nodes in the decision-tree at stage t are decision points where variable fixations for the t-th block of existential variables must be determined by the existential player, respectively a linear program solved, with respect to all previous existential and universal moves $(x^1, \ldots, x^{t-1})$. Arcs of the tree represent moves of the universal player when he fixes his variables to the corresponding lower and upper bounds (in stochastic programming: ... realizations of the random variables). The root node is associated with the first existential move whereas the leaves represent all possible outcomes (scenarios) of the game. Without loss of generality, the first and the last quantifier are assumed to be existential and the quantifiers are strictly alternating in the following example, thus each variable block contains exactly one variable. Figure 4.3 shows the decision-tree for the quantification sequence $\exists x_1 \, \forall x_2 \in \{l_2, u_2\} \, \exists x_3 \, \forall x_4 \in \{l_4, u_4\} \, \exists x_5$. The upper index (j) denotes the j-th copy of the variable and the equations at non-leaf nodes satisfy the *nonanticipativity property* (see their definition and introduction in [229]), which states that decisions taken at any stage must not depend on future realizations of universal variables or on future decisions.

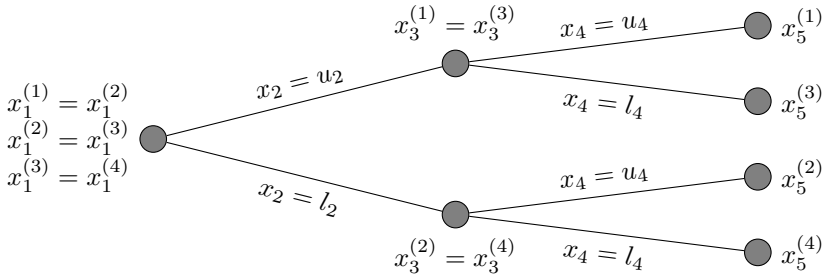


Figure 4.3.: QLP decision-tree for an $\exists \forall \exists \forall \exists$ quantifier sequence.

The scenario-tree of a MSSLP can be encoded into a deterministic equivalent linear program (DEP) by replicating the underlying deterministic linear program (also called core model) for each possible scenario (possible path of events from the root to the leafs of the scenario-tree) and in a similar way, we can construct a deterministic equivalent for a QLP

instance. However, instead of encoding the scenario-tree of randomly arising scenarios, we encode the decision-tree of the universal player into the deterministic equivalent by replicating the underlying existential LP

$$\exists \circ x_\exists : A_\exists x_\exists \leq b(x_\forall), \forall x_\forall \in E$$

of the QLP instance for each possible move sequence of the universal player, where we bring the columns that correspond to decisions of the universal player to the right-hand side $b(x_\forall)$. Given the decision-tree of the universal player as depicted in Figure 4.3, the underlying deterministic model can be parsed into a deterministic equivalent linear program based on the *nodes* of the tree or based on the resulting *scenarios* (paths from the root to the leafs).

Node-based The *compact-view* formulation as described in [181] is node-based and directly exploits the tree structure by defining a reduced set of decision variables. It implicitly satisfies the nonanticipativity property because all nodes in the tree that share a common history up to a stage t also share the same set of decision variables up to that point. It can be seen as a direct mapping on the decision-tree of the universal player, and the underlying LP has so-called (nested) block-ladder structure as depicted in Figure 4.4a for the above example with the quantifier sequence $\exists\forall\exists\forall\exists$. $A_1, \ldots, A_t$ are those blocks of columns of A that belong to the first t variables. The universal player is allowed to make two moves resulting in four scenarios in block ladder matrix form. Figure 4.4b shows, how the block structure of an instance changes, when the position of the last universal quantifier is changed. Note that the horizontal line between the $A_{\exists 3}$ blocks has no structural relevance in this matrix and is only kept for clarification purposes. Figure 4.4c shows the block structure of an $\exists\exists\exists\forall\forall$ QLP. It can be observed that the dimension of the deterministic equivalent in Figure 4.4a is higher, because the different choices of the existential player after a move of the universal player have to be considered.

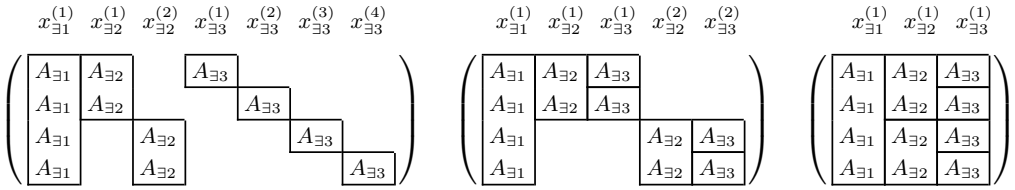


Figure 4.4.: Compact-view DEP block structure for various quantifier sequences.

A game between the universal and the existential player can be traced in the DEP as follows. Firstly, the existential player chooses assignments for his first-stage variables. Thereafter, the universal player decides whether the game is continued in the upper or in the lower part of the DEP. The second move of the existential player consists of choosing assignments for his second-stage variables etc. After as many rounds as the QLP possesses quantifier blocks, the game reaches a single scenario and ends.

Scenario-based In the scenario-based approach, also called *split-view formulation* or *split-variable formulation*, the underlying existential LP is entirely replicated for each path from the root to a leaf of the universal player decision-tree as depicted in Figure 4.5.

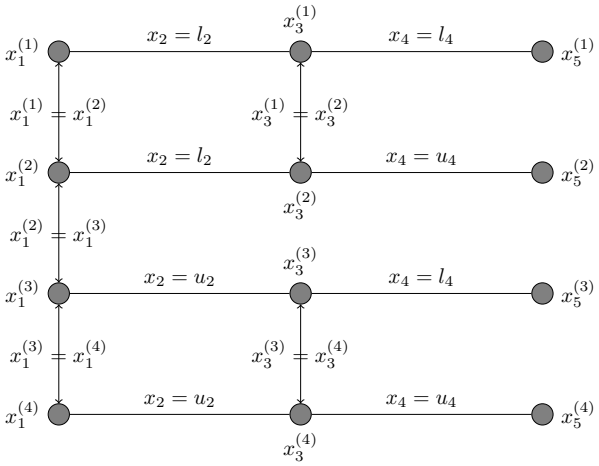


Figure 4.5.: Split-variable universal player decision-tree for an $\exists\forall\exists\forall\exists$ quantifier sequence.

In contrast to the compact-view formulation, which implicitly ensures the nonanticipativity property, since all scenarios that share a common history up to stage t, also share the same decision variables in the DEP up to that point, the same does not hold for the split-variable formulation when the model is parsed according to single scenarios.

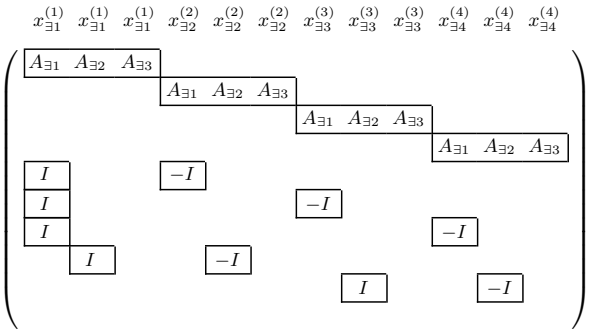


Figure 4.6.: DEP block structure quantifier sequence $\exists\forall\exists\forall\exists$.

Instead, this property must be explicitly modeled by a set of a set of constraints to reflect the underlying model structure induced by the decision-tree of the universal player (the vertical lines in Figure 4.5). This is done by the addition of so called *nonanticipativity constraints (NAC)*, which are of the form

$$x_{ts} = x_{ts'} \, \forall n \in \{1, \ldots, N_t\}, \, t \in \{1, \ldots, H-1\}, s \neq s' \in \Lambda_{tn},$$

where Λ_{tn} is the set of scenarios passing through node n at stage t. Consider for example variable $x_3^{(1)}$ and variable $x_3^{(2)}$ in Figure 4.5. They correspond to two distinct scenarios, but both of them share the same history (path of moves) up to stage $t = 2$ and thus must also have the same value up to that time. Note that this is a central point - scenarios with a common history must have the same set of decisions - and this must always be taken into account when formulating a scenario-based DEP. Figure 4.6 shows the matrix structure for the example with the quantifier sequence $\exists\forall\exists\forall\exists$ already mentioned above.

Figure 4.7 shows, how the block structure of an instance changes, when the position of the last universal quantifier in the quantification vector is changed. Figure 4.8 shows the block structure of an $\exists\exists\exists\forall\forall$ QLP. It can be observed that the dimension of the determin-

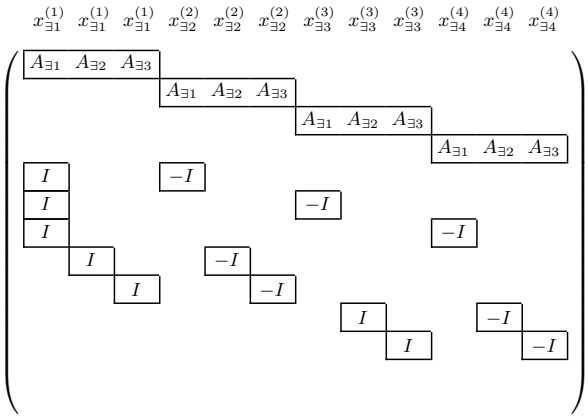


Figure 4.7.: DEP block structure quantifier sequence $\exists\forall\exists\exists\forall$.

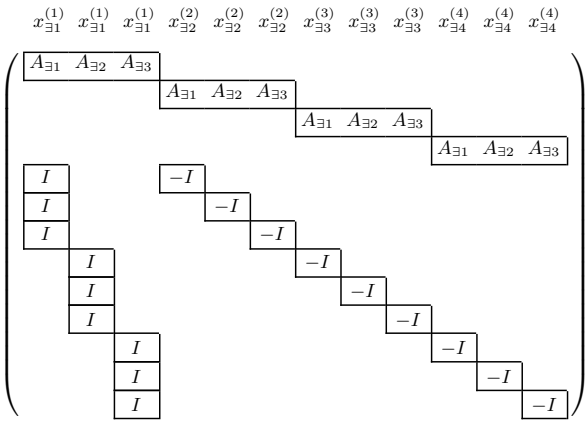


Figure 4.8.: DEP block structure quantifier sequence $\exists\exists\exists\forall\forall$.

istic equivalent in Figure 4.8 is the highest, because the fact that all existential variables must hold for all possible scenarios simultaneously must be explicitly encoded via NACs. The different choices of the existential player after a move of the universal player have to be considered. However, since the entire existential variable block is replicated for each scenario, only the number of NACs change.

Example: DEP of a QLP

Example 9. *To convert the QLP*

$$\min -x_0 + 2x_1 - x_2$$

$$\exists\, x_1 \in [0,1]\ \forall\, x_2 \in [0,1]\ \exists\, x_3 \in [0,1] :$$
$$\begin{pmatrix} 0 & -1 & -1 \\ -1 & 1 & 1 \\ 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} -1 \\ 1 \\ 3 \end{pmatrix}$$

into a DEP, we rewrite the QLP to the effect that the objective function is pushed into the matrix and by Lemma 3.6 being surrogated by an additional variable k. The resulting QLP looks as follows

$$\min k$$

$$\exists\, k \in [-\infty, +\infty]\ \exists\, x_1 \in [0,1]\ \forall\, x_2 \in [0,1]\ \exists\, x_3 \in [0,1] :$$
$$\begin{pmatrix} -1 & -1 & 2 & -1 \\ 0 & 0 & -1 & -1 \\ 0 & -1 & 1 & 1 \\ 0 & 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} k \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 3 \end{pmatrix},$$

where k is unbounded (alternatively we can choose large constants like e.g. -10^{50} and $+10^{50}$ or precompute possible bounds).

Using the compact-view formulation the DEP has four existential variables $k, x_1, x_3^{(1)}, x_3^{(2)}$ and 8 constraints. The corresponding LP looks as follows:

$$\min k$$

$$\begin{pmatrix} -1 & -1 & -1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 2 & 0 & 0 \\ -1 & -1 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} k \\ x_1 \\ x_3^{(1)} \\ x_3^{(2)} \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 3 \\ -2 \\ 0 \\ 0 \\ 1 \end{pmatrix}, 0 \leq x \leq 1.$$

Using the split-variable formulation the DEP has six existential variables $k^{(1)}, k^{(2)}, x_1^{(1)}$, $x_1^{(2)}, x_3^{(1)}, x_3^{(2)}$ and 10 constraints.

The corresponding LP looks as follows:

$$\min k$$

$$\begin{pmatrix} -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} k^{(1)} \\ k^{(2)} \\ x_1^{(1)} \\ x_1^{(2)} \\ x_3^{(1)} \\ x_3^{(2)} \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 3 \\ -2 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, 0 \leq x \leq 1.$$

The first four constraints correspond to scenario 1 ($x_2 = 0$), constraints $5 - 8$ correspond to scenario 2 $x_2 = 1$. The last two constraints ($9 - 10$) are the nonanticipativity constraints and ensure that the duplicated first stage variables ($k^{(1)}, k^{(2)}, x_1^{(1)}, x_1^{(2)}$) coincide.

The split-variable formulation leads to an unnecessary replication of decision variables and constraints and thus may contain a high level of redundancy. However, also the compact-view formulation, which does not contain redundant constraints, can become very large. In general, the resulting DEP grows exponentially with the number of universally quantified variables of the corresponding QLP for both split-variable and compact-view formulation. However, the resulting matrix structure can be exploited by decomposition techniques (see Section 5.1).

Note that we can also create a DEP for QIPs and universally-discrete QMIPs. For the latter, we can simply relax the integrality property of the universal variables and create the DEP for the resulting QLP as described above. In the case of a QIP, we must replicate the integral existential part of the QIP for each possible combination from the ranges of the universal variables, not only the bounds. The result is a large-scale deterministic integer linear program. However, for QMIPs in general, the construction of a DEP is not possible.

The widespread availability of modern LP solvers makes this solution approach available without resorting to any special purpose algorithm. However, for instances with many scenarios this seems not to be a suitable approach and methods that exploit the special structure of the resulting DEPs should be considered as presented in Section 5.1.

4.2. Complexity of Quantified Linear and Integer Programs

In the context of traditional linear and integer programming we are working on problems in the complexity classes **P** or **NP**. However, when considering problems that include uncertainty one is often faced with problems of higher complexity. In the context of quantified linear and integer programming **PSPACE** is the class of interest, albeit there are

certain interesting subclasses of lower complexity. We start with a brief review of the basic concepts of computational complexity as described in the book [9] (or see e.g. [165] for another excellent reference). After the introduction of the basic complexity classes and some details on their interplay in Section 4.2.1, we present an overview of existing results for quantified integer programs and proceeds with a detailed complexity analysis of quantified linear programs in 4.2.2. As a result we provide a comprehensive complexity taxonomy based on several structural properties for the continuous and integer case.

4.2.1. Basic Complexity Classes

Computational complexity theory focuses on issues of computational efficiency – quantifying the amount of computational resources required to solve a given task. It turned out that there is a simple theoretical computational model of computation that suffices for studying many questions about computation and its efficiency – the *Turing machine* (as e.g. formally defined in [9]). It can informally defined as described in [96]:

A Turing machine (TM) consists of a fixed number of tapes each of which contains an infinite number of tape cells. The contents of a tape cell comes from a finite set of symbols called the tape alphabet. All the tapes initially contain only a special blank character except for the finite input written at the beginning of the first tape. Each tape has a tape head sitting on the first character on each tape. The Turing machine also has a finite state memory to control its operations. In each step, it can move each tape independently one character left or right, read and possibly change the characters under each head, change its current state and decide whether to halt and accept or reject the input. Time is measured by the number of steps before halting as a function of the length of the input. Space is measured as the number of different character locations touched by the various heads.

The above model has been developed for so-called *decision*, *recognition*, or *feasibility problems*, which just ask for a *yes* or a *no* answer, however, it is well known that at least in linear programming the optimization problem and the feasibility problem are polynomially equivalent [197, 161].

The running time of an algorithm measures the *time* and *space* used by a TM to solve a class of problems with respect to the size of the input, called the *encoding length* (denoted by n in the following). In the standard encoding scheme the number of bits necessary to save the input of a problem is measured. A *complexity class* is a set of problems that can be computed within some given resource bounds.

Time Complexity: The classes P, NP and coNP

One possible means of describing the complexity of a problem is the amount of time a TM may use to solve a *worst-case* instance with respect to the encoding length. To do this we define *time-bounded computation*, which places a limit on the number of steps a TM can move during its computation, with respect to the input size n.

Definition 4.1 (The class **DTIME**)**.**
Let $T : N \to N$ be some function. A language L is in $\mathbf{DTIME}(T(n))$ *if there is a TM that runs in time $c \cdot T(n)$ for constant $c > 0$ and decides L.*

The complexity class **P** contains all problems that can be solved by a TM in polynomial time, which is at most n^c for some constant $c > 0$. It arises when forming the union over all polynomially time bounded problems.

Definition 4.2 (The class **P**)**.**
$\mathbf{P} = \cup_{c \geq 1} \mathbf{DTIME}(n^c)$.

The class **P** is felt to capture the decision problems that are efficiently solvable by algorithms and therefore is a natural definition of *efficient computation* in the field of theoretical computer science. Linear programming for example is known be solvable in weakly polynomial time [133, 130].

Whereas the complexity class **P** contains problems that can be efficiently *solved* by a TM, a natural extension is to classify problems whose solutions can be efficiently *verified*. The class **NP** contains the set of all problems whose solutions can be verified in polynomial time by a TM, where a polynomial algorithm to find the solution is unknown (a common assumption is to *guess* a solution by a *non-deterministic* Turing machine (NTM) (as e.g. formally defined in [9]).

Definition 4.3 (The class **NP**)**.**
A language $L \subseteq \{0,1\}^*$ is in **NP** *if there exists a polynomial $p : N \to N$ and a polynomial-time TM M (called* verifier *for L) such that for every $x \in \{0,1\}^*$,*

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1 \tag{4.3}$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy M(x,u)=1, then u is called a certificate[1] *for x (with respect to the Language L and the machine M).*

Because a TM can only read one bit in a step, the solution must not be too long – at most polynomial in the length of the input. Thus, clearly $\mathbf{P} \subseteq \mathbf{NP}$, however, if also $\mathbf{P} = \mathbf{NP}$ holds is unknown and one of the most important questions in theoretical computer science (e.f. it is one of the six remaining *millenium problems* [82]). The essence of the **P** vs **NP** question is as follows: *"If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem?"* (e.g. is decryption as easy as encryption?).

To relate the computational complexity of two different problems in **NP** we use the following definitions:

Definition 4.4 (Reductions, **NP**-hardness and **NP**-completeness)**.**
A language $L \subseteq \{0,1\}^*$ is polynomial-time reducible *to a language $L' \subseteq \{0,1\}^*$, denoted by $L \leq_{\mathbf{P}} L'$, if there is a function $f : \{0,1\}^* \to \{0,1\}^*$, which can be computed in polynomial time such that for every $x \in \{0,1\}^*$, $x \in L$ if and only if $f(x) \in L'$.*

[1] A certificate needs not to be the solution but can also have another form, e.g. a tree.

We say that L' is $\mathbf{NP}$*-hard if $L \leq_{\mathbf{P}} L'$ for every $L \in \mathbf{NP}$. We say that L' is* $\mathbf{NP}$*-complete if L' is* $\mathbf{NP}$*-hard and $L' \in \mathbf{NP}$. Note that in order to show that a problem is in* $\mathbf{NP}$*, it suffices to show that a solution can be verified in polynomial time.*

The $\mathbf{NP}$-completeness tells that a problem is as hard as any other $\mathbf{NP}$-complete problem, $\mathbf{NP}$-hardness tells that a problem is at least as hard as any other problem in $\mathbf{NP}$. Those problems turn up in many applications and if $\mathbf{P} \neq \mathbf{NP}$, then efficient algorithms for *every* input do not exist. In 1971, Cook published his seminal paper defining the notion of $\mathbf{NP}$-completeness showing that SAT is $\mathbf{NP}$-complete [70]. One year later Karp showed that many other important problems of practical interest are $\mathbf{NP}$-complete [131], among them problems as e.g. the $0/1$-integer programming feasibility problem. Many of those problems have a very large number of solutions and often there is in some cases no other exact way to find an optimal solution than a brute force search. Today, thousands of computational problems from a wide variety of disciplines have been shown to be $\mathbf{NP}$-complete, among them CSP feasibility [53] and the general integer programming feasibility problem [197]. The essence of $\mathbf{NP}$-completeness is that problems from this class have the ability to represent any other problem from the whole complexity class $\mathbf{NP}$ (and the transformation does not take more than polynomial time with respect to the size of the original problem). Take for example problems from SAT, CSP, and IP feasibility. Each of them can be transformed among themselves in polynomial time and solved with the corresponding solution techniques [225, 142, 238].

An interesting additional complexity class strongly related to $\mathbf{NP}$ is $\mathbf{coNP}$, which can be defined as follows

Definition 4.5 (The class $\mathbf{coNP}$)**.**
A language $L \subseteq \{0,1\}^*$ is in $\mathbf{coNP}$ *if there exists a polynomial $p : N \to N$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$,*

$$x \notin L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 0. \tag{4.4}$$

Just as $\mathbf{NP}$ can be considered to be the set of problems with short "yes"-certificates, $\mathbf{coNP}$ can be considered to be the set of problems with short "no"-certificates and it is widely believed that $\mathbf{coNP} \neq \mathbf{NP}$. An alternative definition results if the $\exists$ quantifier in Definition 4.3 is replaced by a $\forall$ quantifier as e.g. presented in [9].

Definition 4.6 (The class $\mathbf{coNP}$, alternative definition)**.**
A language $L \subseteq \{0,1\}^*$ is in $\mathbf{coNP}$ *if there exists a polynomial $p : N \to N$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$,*

$$x \in L \Leftrightarrow \forall u \in \{0,1\}^{p(|x|)},\ M(x,u) = 1. \tag{4.5}$$

An example for a $\mathbf{coNP}$-complete problem is TAUTOLOGY – the question whether a boolean formula is true under all possible assignments.

Space Complexity: The class PSPACE

The classes **P**, **NP** and **coNP** are characterized by time boundedness and required to be solvable in polynomial time or there must be at least a polynomial time certificate for a "yes" or "no" answer respectively. Another possibility to describe the complexity of a problem is to limit the amount of space used by an algorithm. To do this we define *space-bounded computation*, which places a limit on the number of tape cells a TM can use during its computation, with respect to the input size n.

Definition 4.7 (The class **DSPACE**)**.**
Let $S : N \to N$ be some function and $L \subseteq \{0,1\}^*$. A language L is in $\mathbf{DSPACE}(S(n))$ *if there is a constant c and a TM M deciding L such that at most $c \cdot S(n)$ locations on M's work tapes (excluding the input tape) are ever visited by M's head during its computation on every input of length n.*

When limiting the amount of space an algorithm can use, there is no longer a polynomial limitation for the computation time, since tape cells of a TM can be overwritten and reused. The complexity class **PSPACE** arises when forming the union over all polynomially space-bounded problems.

Definition 4.8 (The class **PSPACE**)**.**
$\mathbf{PSPACE} = \cup_{c \geq 0} \mathbf{DSPACE}(n^c)$.

Likewise to the case of the class **NP**, we can define a notion of *complete* problems for the class **PSPACE**:

Definition 4.9 (**PSPACE**-hardness, **PSPACE**-completeness)**.**
A language L' is **PSPACE***-hard if $L \leq_{\mathbf{P}} L'$ for every $L \in$* **PSPACE***. We say that L' is* **PSPACE***-complete if L' is* **PSPACE***-hard and $L' \in$* **PSPACE***.*

As is the case of **NP**-complete problems, **PSPACE**-complete problems are quite common and often arise naturally. Typical problems from this class are QSAT and QCSP but they also arise from problems reducible to some polynomial-depth two-person games [205, 196], with alternation of existential and universal quantifiers being an interesting variant of this point of view [63]. In the seminal Paper *Games against nature* [164] Papadimitriou showed that the complexity class **PSPACE** includes games with a random, disinterested adversary. There he claimed that the class **PSPACE** can be interpreted as an alternative characterization of *optimization under uncertainty*.

PSPACE includes many other classes as shown in Figure 4.9[2]. For instance, **NP** and hence **P** are included in **PSPACE**, since checking any certificate for a problem in **NP** can not use more than polynomial space because it runs in polynomial time. However, note that some of the areas might be empty since the only result known for sure is that $\mathbf{P} \subsetneq$ EXPTIME.

Whereas the central feature of **NP**-completeness is the existence of a short certificate, which can be checked in polynomial time, the analogous concept for **PSPACE**-completeness is the existence of a *winning-strategy* for a two-person game [9]. However,

[2] See [9] for a more deatiled delimitation to other compelxity classes

describing such a strategy in tree form as in Definition 2.14 can require exponential space. This lack of a *short certificate (with polynomially many bits)* is the crucial difference, and, although this has not yet been proved, it is widely believed that **PSPACE**-complete problems are inherently more difficult than any problem in **NP** or **coNP**.

From a more practical viewpoint the real issue separating problems that are **NP**-complete and those which are **PSPACE**-complete lies in the size of the problem representation, i.e., the size of the input. For example, any problem in **PSPACE** can be encoded as a polynomially sized QCSP. CSPs, on the other hand, are only **NP**-complete and except $\mathbf{PSPACE} = \mathbf{NP}$, there do exist problems for which a CSP formalization is always exponentially larger than the equivalent QCSP representation, even though they both have the same exponential worst case time complexity. The same can be observed when the DEP of a QLP or a QIP is constructed (see also Section 4.1.3).

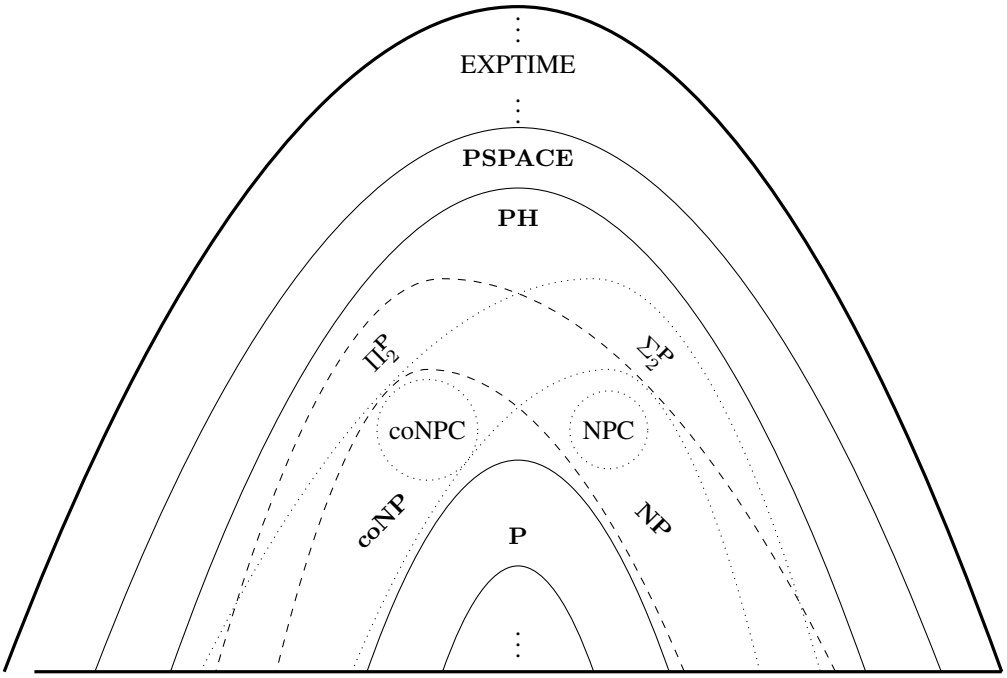


Figure 4.9.: Complexity Classes [9].

The Polynomial Hierarchy and Alternations

The *polynomial-time hierarchy* (PH), which is a generalization of **P**, **NP**, and **coNP** for which no short certificates of membership do exist, was first proposed by Stockmeyer [205]. It consists of a number of subclasses, which are conjectured to be distinct. It can be stated as the set of languages defined via polynomial time predicates combined with a constant number of alternating ($\forall$) and ($\exists$) quantifiers, generalizing the definitions of **NP** and **coNP**. Whereas problems in **NP** ask whether there *exists* a string from a language L

satisfying certain properties, problems in **coNP** ask whether *all* strings from a language L satisfy certain properties. A natural extension is to consider problems that combine existential and universal quantifiers. We distinguish the case when the quantification starts with an existential quantifier and when it starts with an universal quantifier respectively and follow the notation presented in [9].

Definition 4.10 (The class $\Sigma_2^{\mathbf{P}}$)**.** *The class $\Sigma_2^{\mathbf{P}}$ is the set of all languages L for which there exists a polynomial-time TM M and a polynomial q such that*

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)}\, \forall v \in \{0,1\}^{q(|x|)},\ M(x,u,v) = 1 \tag{4.6}$$

for every $x \in \{0,1\}^*$.

Definition 4.11 (The class $\Pi_2^{\mathbf{P}}$)**.** *The class $\Pi_2^{\mathbf{P}}$ is the set of all languages L for which there exists a polynomial-time TM M and a polynomial q such that*

$$x \in L \Leftrightarrow \forall u \in \{0,1\}^{q(|x|)}\, \exists v \in \{0,1\}^{q(|x|)},\ M(x,u,v) = 1 \tag{4.7}$$

for every $x \in \{0,1\}^*$.

Note that $\Sigma_0^{\mathbf{P}} = \Pi_0^{\mathbf{P}} = \mathbf{P}$, $\Sigma_1^{\mathbf{P}} = \mathbf{NP}$ and $\Pi_1^{\mathbf{P}} = \mathbf{coNP}$. The complexity classes that can be defined via a combination of a polynomial-time computable predicate and a *constant* number of $\forall$ and $\exists$ quantifiers make up the polynomial hierarchy.

Definition 4.12 (The class $\Sigma_i^{\mathbf{P}}$)**.** *For $i \geq 1$, a language L is in $\Sigma_i^{\mathbf{P}}$ if there exists a polynomial-time TM M and a polynomial q such that*

$$\begin{aligned} x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots Q_i u_i \in \{0,1\}^{q(|x|)} \\ \textit{s.t. } M(x, u_1, u_2, \ldots, u_i) = 1 \end{aligned} \tag{4.8}$$

for every $x \in \{0,1\}^*$, where Q_i denotes $\forall$ or $\exists$ depending on whether i is even ($\forall$) or odd ($\exists$).

Definition 4.13 (The class $\Pi_i^{\mathbf{P}}$)**.** *For $i \geq 1$, a language L is in $\Pi_i^{\mathbf{P}}$ if there exists a polynomial-time TM M and a polynomial q such that*

$$\begin{aligned} x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \ldots Q_i u_i \in \{0,1\}^{q(|x|)} \\ \textit{s.t. } M(x, u_1, u_2, \ldots, u_i) = 1 \end{aligned} \tag{4.9}$$

for every $x \in \{0,1\}^*$, where Q_i denotes $\forall$ or $\exists$ depending on whether i is even ($\exists$) or odd ($\forall$).

Definition 4.14 (The polynomial hierarchy)**.** *The polynomial hierarchy is the set $\mathbf{PH} = \cup_i \Sigma_i^{\mathbf{P}} = \cup_i \Pi_i^{\mathbf{P}}$ [9].*

In [195] a list of problems known to be complete for the second and higher levels of the polynomial-time hierarchy is presented, among them $\sum_i$-SAT and Presburger Arithmetic. When being restricted to a conjunction of linear inequalities, Presburger Arithmetic is in

term quantified integer programming [211]. For every $i \geq 1$ the classes $\Sigma_i^{\mathbf{P}}$ and $\Pi_i^{\mathbf{P}}$ have a complete problem involving quantified boolean expressions of the following type with a limited number of alternations

$$\sum_i SAT = Q_1 u_1 \, Q_2 u_2 \, \ldots \, Q_i u_i \phi(u_1, u_2, \ldots, u_i) = 1, \tag{4.10}$$

where ϕ is a boolean formula (not necessarily in CNF form)[165].

Each u_i is a vector of boolean variables, and Q_i is either $\forall$ or $\exists$ depending on whether i is even or odd and whether the first quantifier is universal $Q_1 = \forall$ or existential $Q_1 = \exists$. The $\sum_i$ SAT problem is a special case of the *true quantified boolean formula* (TQBF) problem (which is known to be **PSPACE**-complete [9]), to a constant number of alternations. Note that there is neither a restriction on the number of boolean variables, nor on the number of clauses. It is known that $\mathbf{PH} \subseteq \mathbf{PSPACE}$ and widely believed that unless the polynomial hierarchy collapses $\mathbf{PH} \neq \mathbf{PSPACE}$ [9].

The polynomial hierarchy can alternatively be defined using *Alternating Turing machines* (ATMs) (as e.g. formally defined in [9]). An alternating Turing machine is a NTM in which every state is either $\exists$ or $\forall$ and the definition of acceptance of an ATM alternates between these modes [63], whereas a NTM has an existential acceptance criterion for problems from the class **NP** and a universal acceptance criterion for problems from the class **coNP**. Consider for example a quantified boolean formula starting with existential variables and with a bounded number of quantifier alternations. The ATM starts branching existentially to try all possible values of the existential variables in the first block and then universally to try all possible values the universal variable in the next block, and so on, in the left-to-right order the variables are bound to the quantifiers. After deciding (guessing) a value for all quantified variables, the machine accepts if the resulting boolean formula evaluates to true, and rejects if it evaluates to false. At an existential variable the ATM accepts if an assignment exists such that the remaining problem stays satisfiable. At a universal variable the ATM accepts if for any possible variable assignment the remaining problem stays satisfiable. Thus, ATMs generalize the rules used in the definition of the complexity classes **NP** and **coNP** and it is easy to see that finitely many alternations results in the finite levels of the polynomial time hierarchy. The class of problems ATMs can solve in polynomial time is called **AP** (alternating polynomial time [9]) and for unbounded alternation holds:

Proposition 4.1. $\mathbf{AP} = \mathbf{PSPACE}$.

Proof. See [9]. □

Efficient Computation vs. Expressiveness

In comparison, due to the the negative results from complexity theory, it seems discouraging to search for efficient (i.e., polynomial time) algorithms to solve optimization problems that are **NP**-hard, **coNP**-hard, **PSPACE**-hard or somewhere between.

As a consequence, algorithmics often tries to find restricted subclasses of problems that are hard in general, which then allow finding a polynomial time algorithm [129], or at least

approximations [124, 92]. Nevertheless, many real-world problems are **NP**-hard problems (e.g. mixed-integer linear programs) and **PSPACE**-hard problems (e.g. computer games like Chess) that cannot be ϵ-approximated in polynomial time. However, an ordinary point of criticism is that the definition of time and space complexity relies on *worst-case exact computation*, which is too strict in many cases and does not reflect the fact that the most astonishing and admired computational feats are based on formulations of **NP**-hard problems and **PSPACE**-hard problems.

Indeed, many hard real-world problems can be solved efficiently in practice, e.g., we are today able to solve very large mixed-integer linear programs of practical size with up to millions of variables, even though they might not appear to belong to a restricted subclass. Furthermore note that even though polynomial time algorithms do exist, they are not necessarily used in practical applications. The simplex algorithm has nonpolynomial (worst-case) complexity, but it is very successfully applied to LPs – which, as mentioned above, are in **P** – quite commonly.

4.2.2. Complexity of Quantified Linear and Integer Programming

It is clear that in a quantified linear or integer program complexity arises from two issues, the number of alternations in the quantifier string and the structure of the constraint matrix. For instance, even in the presence of unbounded alternation Q2SAT – the problem where each clause is limited to at most two literals – is solvable in polynomial time [10], whereas Q3SAT is already **PSPACE**-complete. We first review existing complexity results for the integer case and clarify the proof ideas. For the continuous case, complexity results in the presence of unbounded alternation only exist for problems where the constraint matrix satisfies certain structural properties. We therefore focus on results for arbitrary matrix structures but bounded alternation. However, the complexity for QLPs in general stays unknown.

Complexity of QIPs: Literature Review

The complexity results for several classes of QIP feasibility problems as presented in the following are primary based on the two papers by Subramani [209, 211].

Proposition 4.2. *The $\forall$-QIP decision problem can be solved in polynomial time.*

Proof. In [211] it was shown that the $\forall$-QIP problem can be relaxed to the $\forall$-QLP problem without altering the solution space. The resulting $\forall$-QLP can be solved in polynomial time using function `ELIM-UNIV-VARIABLE()` (see Algorithm 2 in Section 3.1.1). □

This is an interesting result, because at a first glance this problem seems to be in **coNP** because of the universal quantifier. Because of its structural resemblance to the **coNP**-complete problem TAUTOLOGY one might even conclude that the similar holds for the $\forall$-QIP problem.The fact that tailing universal variables in constraints, including the special case when the last variable block consists of universal variables, can be simply eliminated in polynomial time, is also known as *universal reduction* in the context of QSAT problems [59].

Proposition 4.3. *The $\exists\forall$-QIP decision problems is* $\mathbf{NP}$*-complete.*

Proof. To show that the $\exists\forall$-QIP problem is in $\mathbf{NP}$, we guess a solution $\overline{x}_\exists$ for the existential player. The solution can be verified by solving the $\forall$-QIP

$$\forall \circ x_\forall \in [l_\forall, u_\forall] : A_\forall \cdot x_\forall \leq \underbrace{b - A_\exists \cdot \overline{x}_\exists}_{=:b(\overline{x}_\exists)},\ x_\forall \in \mathbb{Z}^{|x_\forall|},$$

which by Proposition 4.2 can be solved in polynomial time. The $\mathbf{NP}$-hardness can be established by a reduction from the general IP problem, which is known to be $\mathbf{NP}$-complete, to a $\exists\forall$-QIP where the $\forall$-block consists of dummy variables fixed to zero. □

Note that without the knowledge about universal reduction the $\exists\forall$-problem seem to be $\Sigma_2^\mathbf{P}$-complete.

Proposition 4.4. *The $\forall\exists$-QIP decision problems is $\Pi_2^\mathbf{P}$-complete.*

Proof. In [211] the concept of Alternating Turing-Machines (ATM) was used to show that the problem to decide arbitrary $\forall\exists$-QIPs belongs to the class $\Pi_2^\mathbf{P}$. The ATM starts in state $\forall$ and guesses the values $x_\forall$, it then switches to state $\exists$ and guesses the values $x_\exists$. If the constraint system $Ax \leq b$ holds, the ATM accepts, otherwise it rejects. Since the variables $x_\exists$ and $x_\forall$ are integral, the guessed vector x only has polynomial size with respect to the input. The $\Pi_2^\mathbf{P}$-hardness can be established by a reduction from the QSAT problem

$$\forall x_1\ \ldots \forall x_m\ \exists x_{m+1}\ \ldots \exists x_n\ \ \mathcal{C},$$

which is known to be $\Pi_2^\mathbf{P}$-complete [58], to the 0/1-QIP problem as shown in Example 6 in Section 2.3. □

Proposition 4.5. *The QIP decision problem is* $\mathbf{PSPACE}$*-complete.*

Proof. To show that the QIP problem is in $\mathbf{PSPACE}$, we solve it with the Minimax-Algorithm (see Algorithm 4 in Section 4.1.2) or the Alpha-Beta Algorithm (see Algorithm 5 in in Section 4.1.2). Both utilize a depth-first search and need not hold the entire game-tree in the memory but only the current path. Thus they can evaluate the exponentially sized game-tree with only polynomial space [135]. The $\mathbf{PSPACE}$-hardness can be established by a reduction from the QSAT problem (see Definition 2.24), which ins known to be $\mathbf{PSPACE}$-complete, to the 0/1-QIP problem as shown in Example 6 in Section 2.3. □

Complexity QLPs: Literature Review

The complexity of quantified linear programming feasibility problems for arbitrary constraint matrices and unbounded quantifier alternation is unknown. The only known results for unbounded quantifier alternation exist for the case where the constraint matrix is totally unimodular (TUM), in this case the QLP decision problem is in $\mathbf{P}$ as shown in [108, 212].

In the succeeding complexity analysis we concentrate on quantified linear programs restricted to a constant number of quantifier alternations. In this context, it has been shown that the QLP feasibility problem with only one quantifier change is either in $\mathbf{P}$ (when the quantification begins with existential quantifiers and ends with universal ones) or $\mathbf{coNP}$-complete (when the quantification begins with universal quantifiers and ends with existential ones) [212].

Proposition 4.6. *The $\exists\forall$-QLP decision problems is in $\mathbf{P}$.*

Proof. We use function `ELIM-UNIV-VARIABLE()` (see Algorithm 2 in Section

From the perspective of algorithm development this is an interesting result, since as we know from Section 3.3, the feasible solution of the relaxed QLP that results when all universally quantified variables are pushed to the end of the quantification vector, yields a certificate of feasibility and an upper bound for the original QLP. The complexity result establishes that computing this bound is a very cheap operation.

Proposition 4.7. *The $\forall\exists$-QLP decision problem is $\mathbf{coNP}$-complete.*

Proof. To show that the $\forall\exists$-QLP is in $\mathbf{coNP}$ we use the fact that a problem is in $\mathbf{coNP}$ if its negation is in $\mathbf{NP}$. The negation of a $\forall\exists$-QLP can be written as

$$\underbrace{\exists x_1 \in [l_1, u_1] \ \ldots \exists x_m \in [l_m, u_m]}_{=: \neg(\forall \circ x_\forall \in [l_\forall, u_\forall])} \ \neg(\exists \circ x_\exists : (A_\forall, A_\exists)^T \cdot (x_\forall, x_\exists) \leq b), x \in \mathbb{Q}^n.$$

To show that this problem is in $\mathbf{NP}$, we guess values for the variables $\overline{x}_\forall = \overline{x}_1, \ldots, \overline{x}_m$ with $x_{\forall i} \in \{l_{\forall i}, u_{\forall i}\}$, and check in polynomial time if the resulting polyhedral system

$$\forall \circ x_\exists : \ A_\exists \cdot x_\exists \leq \underbrace{b - A_\forall \cdot \overline{x}_\forall}_{=: b(\overline{x}_\forall)},$$

is empty, and thus have a short certificate for a possible "no" answer. As shown in [212], the $\mathbf{coNP}$-hardness can be established by an *inverse reduction* from the *Maximum 2-satisfiability problem (MAX2SAT)*, which is known to be $\mathbf{NP}$-complete [165]. A problem P_1 is said to be inverse reducible to a problem P_2, if there exists a polynomial time TM computable function such that $x \in P_1 \Leftrightarrow f(x) \notin P_2$.

Definition 4.15 (Maximum 2-satisfiability (MAX2SAT))**.**
Given a SAT problem as in Definition 2.22 where each clause in $\mathcal{C} = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ consists of exactly two literals from $\{y_1, \overline{y}_1, y_2, \overline{y}_2, \ldots, y_n, \overline{y}_n\}$, and a number k. The task of the Maximum 2-satisfiability problem (MAX2SAT) is to determine whether there is an assignment $x \in \{0,1\}^n$ such that the number of satisfied clauses in the formula $\mathcal{C}$ is greater than or equal to k.

Let P_1 be the MAX2SAT problem, then the corresponding P_2 problem, which is a $\forall\exists$-QLP is constructed as described in the following. Each clause C_i is replaced by a pair of linear constraints as follows:

1. If $C_i = (y_j, y_k)$, add the two constraints: $(1 - y_j) + x_i \geq 1$ and $(1 - y_k) + x_i \geq 1$.
2. If $C_i = (\overline{y}_j, y_k)$, add the two constraints: $y_j + x_i \geq 1$ and $(1 - y_k) + x_i \geq 1$.
3. If $C_i = (y_j, \overline{y}_k)$, add the two constraints: $(1 - y_j) + x_i \geq 1$ and $y_k + x_i \geq 1$.
4. If $C_i = (\overline{y}_j, \overline{y}_k)$, add the two constraints: $y_j + x_i \geq 1$ and $y_k + x_i \geq 1$.

Thus, $\mathcal{C}$ has been replaced by a set of $2 \cdot m$ constraints, this set is called $\mathcal{Z}_1$. Additionally, a second set of constraints $\mathcal{Z}_2$ is added, which contains the bounds for the x-variables $0 \leq x_i \leq 1,$ with $i = 1, \ldots, m$. Finally we add an aggregate constraint

$$\sum_{i=1}^{k} x_i \leq k + 1, \tag{4.11}$$

called $\mathcal{Z}_3$, and write the $\forall\exists$-QLP as follows:

$$\forall\, y_1 \in [0,1]\, \forall\, y_2 \in [0,1] \ldots \forall\, y_n \in [0,1]\, \exists x_1\, \exists x_2 \ldots \exists x_m \ : \mathcal{Z}_1 \wedge \mathcal{Z}_2 \wedge \mathcal{Z}_3.$$

Now consider an assignment $\overline{y}$ for problem P_1 that satisfies all clauses in $\mathcal{C}$ and assume w.l.o.g. the i-th clause to be $C_i = (y_j, \overline{y}_k)$ (the other cases can be argued in a similar way). Thus, the corresponding constraints in P_2 are $(1 - y_j) + x_i \geq 1$ and $y_k + x_i \geq 1$. Furthermore, assume w.l.o.g. that $y_j =$ **true** in P_1, thus the setting $y_j = 1$ in P_2 enforces x_i to be at least one in order to satisfy $(1 - y_j) + x_i \geq 1$. Thus, whenever a clause C_i in problem P_1 is satisfied by an assignment y, the corresponding variable x_i in problem P_2 is set to 1. If an assignment satisfies k or more clauses in P_1, at least k of the x_i variables are set to 1, violating the aggregate constraint, making the $\forall\exists$-QLP infeasible.

Likewise, if clause $C_i = (y_j, \overline{y}_k)$ is not satisfied by an assignment y, $y_j =$ **false** and $y_k =$ **true** must hold. Hence, the corresponding constraints in P_2 can be satisfied only using the y variables, allowing variable x_i being set to 0. As a result, if all possible assignments to $\mathcal{C}$ satisfy at most $k - 1$ clauses C_i in problem P_1, then at most $k - 1$ variables x_i must be set to 1 in problem P_2 in each assignment, and thus the $\forall\exists$-QLP is true. □

This is an interesting result because it is another example that shows that also continuous mathematical programs can encode hard discrete problem. For instance, from the **coNP**-completeness we can conclude that it is possible to transform negated instances of a $\forall\exists$-QLP into a mixed integer program (MIP) in polynomial time (see, e.g. [148] for an example using Farkas' Lemma).

Complexity of QLPs: Own Contributions

Up to this point, we presented known complexity results for the simple case of one single quantifier change or the special case when the constraint matrix is TUM. In the following we extend the complexity results to an arbitrary but fixed constant number of quantifier changes as described in Definition 2.6. In our proof we use a polyhedral approach based on the investigations of the polyhedral and algorithmic properties of QLPs elaborated in

the previous sections. As a preparation, let us first recall some basic results about encoding lengths, where we follow the concepts of [118].

Definition 4.16 (Row and Vertex Complexity)**.** *Let $P \subseteq \mathbb{R}^n$ be a polytope and ϕ and ν be positive integers.*

- *P has* row complexity *(also called facet complexity) of at most ϕ, if there is a system of inequalities with rational coefficients $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m$ for some $m \in \mathbb{N}$ such that P is described with the help of a set of inequalities and the encoding length (i.e., the number of bits used) of each row is at most ϕ. Each entry in A and b requires at least one bit.*

- *P has* vertex complexity *of at most ν, if there is a finite set $V \subseteq \mathbb{R}^n$ such that $P = \mathrm{conv}(V)$ and the encoding length of each vector in V is at most ν. We assume that $\nu \geq n$.*

Lemma 4.8. *A polytope P with row complexity of at most ϕ has vertex complexity of at most $4\phi n^2$ and a polytope with vertex complexity of at most ν has row complexity of at most $3\nu n^2$.*

A proof and further details can be found in [118].

Theorem 4.9. *Consider some instance of the c-QLP problem. The vertices of the solution polytope whose universal variables attain one of their bounds each, has an encoding length which is polynomially bounded in the encoding length of this instance.*

Proof. In order to prove the theorem, we inspect the DEP of a QLP instance and present an algorithm which constructs such a vertex in $\mathbb{Q}^n$. The algorithm itself has no limitations regarding time or space complexity. We only guarantee that it terminates in finite time at a vertex polynomially bounded in the encoding length of the instance.

Let k be the number of quantifier changes. The variables of the last block of existential variables occur replicated in the DEP as often as there are scenarios. With $N = 2^{|x_\forall|}$, let $S = \{s_1, ..., s_N\}$ be the set of scenarios, each represented by a certain subset of rows R_i. The FME-method shows that the elimination of a variable x which only occurs in e.g. R_i can be performed without considering other rows than those in R_i. That is, we are allowed to decouple the scenarios as long as we are only interested in the elimination of intra-scenario variables.

Instead of using FME-method for the elimination of the variables of the last existential block, the decomposed single systems can be compressed to a representation without superfluous zeros and with facet complexity ϕ. Each polytope belonging to such a single system is then transformed from facet into vertex representation, and the desired projection is performed by simply leaving off the last dimension of all vertices.

In the next step, the projected vertex representation is re-transformed into facet form by constructing the convex hull. Let this new system have the smallest possible encoding length. If the former system had facet complexity ϕ the new system has a facet complexity of $O(\phi n^4)$, following Lemma 4.8. Last but not least, the necessary extra-zeros are added

to the system again such that the decomposed single systems can be merged to one large system of inequalities again.

After k such elimination steps only the variables of the first stage remain, so no extra zeros need to be incorporated. The facet complexity of this final system then is $O(\phi\, n^{4k})$. Applying Lemma 4.8 once more shows that the number of bits, which we need in order to encode the first move of the existential player, is polynomially bounded in the size of the input description.

The assignment of the first-stage variables matches the first move of the existential player. After the universal player's move, this first-stage assignment can be brought to the right-hand side and the elimination of the next block of variables in the DEP can be performed. The whole procedure need not be repeated more than k times such that the constructed vertex of the QLP solution space has polynomial size (bit-complexity). □

Lemma 4.10. *Given a c-QLP instance. If there is a winning strategy for the existential player, there is one whose games can be polynomially encoded.*

Proof. Suppose there is a winning strategy for the existential player, then the solution polytope is not empty. Therefore, we can construct a new winning strategy for the existential player as follows: The first existential variable is chosen maximal in its range such that the remaining solution polytope after this move is not empty (c.f. Figure 3.5). Then, for both universal moves, the third existential variable is chosen maximal in its range such that the remaining solution polytope after this move is not empty. Repeating this argument leads to a winning strategy whose games are vertices of the solution polytope. Additionally, all universal moves of these games attain their bounds, as in Proposition 4.9. Thus, all games of this strategy can be polynomially encoded. □

Proposition 4.11. *The c-QLP decision problems with $k-1$ quantifier changes (k variable blocks, n variables) is in $\Sigma^{\mathbf{P}}_{k-1}$ if the first quantifier is existential ($Q_1 = \exists$) and the last quantifier is existential ($Q_n = \exists$). If the first quantifier is universal ($Q_1 = \forall$) and the last quantifier is existential ($Q_n = \exists$) the problem is in $\Pi^{\mathbf{P}}_{k-1}$.*

Proof. We will construct an algorithm that (though in general being highly inefficient) can decide c-QLP in polynomial space. That is, for a given c-QLP instance with encoding length I_{enc}, we can determine from Theorem 4.9 a polynomial $S_{\text{enc}}(I_{\text{enc}}) \in O(\phi\, n^{4k})$ which states an upper limit on the space requirement of the algorithm depending on the c-QLP input encoding length. By Lemma 4.10, either there is an existential winning strategy whose games can be polynomially encoded, or there is no winning strategy for the existential player at all. The second case would also imply that there is no existential winning policy for this c-QLP instance by Corollary 3.5. Therefore, an algorithm which examines all existential winning strategies with encoding length of at most $S_{\text{enc}}(I_{\text{enc}})$ of the existential moves can decide the c-QLP problem.

We use an ATM that guesses values for each existentially quantified variable using an $\exists$ state and for the universally quantified variables using a $\forall$ state. However, if the last quantifier block consists of existential variables we only do this for the first $k-1$ blocks and do a deterministic polynomial time computation at the end to check if the resulting

$\exists$-QLP has a solution. This can be done in polynomial time with standard LP algorithms [133, 130]. Thus, depending on whether the first variable block is existentially quantified or universally quantified, the problem is in $\Sigma^{\mathbf{P}}_{k-1}$ or $\Pi^{\mathbf{P}}_{k-1}$, respectively. □

Corollary 4.12. *The c-QLP problem is in* **PSPACE**.

Proof. $\mathbf{PH} \subseteq \mathbf{PSPACE}$ [9]. □

The result of Corollary 4.12 is highly related to [14](chapters 13/14), where it could be shown that in the field of real numbers even polynomial programs with a constant number of quantifier changes can be decided with polynomial space using algebraic representations for numbers. Thus, the pure question whether the QLP with a constant number of quantifier alternations can be decided with polynomial space is known and answered. However, our results add – to the best of our knowledge – the following aspects.

- We guarantee that solvable c-QLPs do not only possess real solutions of polynomial length, which are implicitly given by an algebraic term, but they also possess rational solutions of polynomial length for each vector component, explicitly given as a binary encoding of the nominator and the denominator of a rational number.
- Our analysis does not only attack the decision problem, but also gives a guide for a solution procedure for the first stage variables as described in [88]. In this way, we can generate the certificate in polynomial space (although we cannot keep the complete certificate in memory at one point in time).

However, although a formal proof for the case of unbounded alternation has proven elusive, we suspect that quantified linear programming is **PSPACE**-hard in general. It has seemingly been shown that the QLP decision problem with an arbitrary (unbounded) number of quantifier changes is in **PSPACE** (see, e.g. [212, 147]), but both proofs have significant shortcomings. To show that a problem is in **PSPACE** it must be shown that an algorithm runs on a TM using space which is polynomial in the number of bits required to represent the input, and each arithmetic operation needs to be done in the grade-school fashion, bit by bit. Therefore, it does not suffice to show that the result is rational. It must be also verified that all *intermediate numbers* involved in the computation can be represented by polynomially many bits and also the result. A similar problem occurs in the context of Gaussian elimination where intermediate computation steps may suffer from exponential growth, while the final answer is of polynomial size [197, 100, 182].

The approach proposed in [212] uses an ATM to guess the values $x_\exists$ and $x_\forall$ and argues by Proposition 4.1 that for unbounded alternation the QLP problem is in **PSPACE**. However, they neither mention whether it suffices to inspect only polynomial sized rational values of $x_\exists$ where the size of the denominator is bounded in the size of the input, nor if the solution values they compute have a polynomial encoding and if they can be computed with at most polynomial space.

In [147] the authors propose an approach where they use the FME-method (see Algorithm 3 in Section 3.1) to compute vertices of the QLP solution polytope which are polynomially bounded in the encoding length of this instance. However, they neglect the fact that

the vertex complexity in one of the intermediate elimination steps needs not to be polynomially bounded, e.g. the rational values describing a vertex resulting from the intersection of new constraints generated by the FME-method might not be bounded polynomially in the case of unbounded alternation.

5. The Alpha-Beta Nested Benders Decomposition Algorithm

Prior to this work, the only known approach to solve the QLP decision problem in general involved sequential variable elimination techniques as described by Subramani [212], which lead to double exponential time and space consumption. In Section 4.1.3 we showed how an equivalent deterministic linear program for the QLP optimization problem can be formulated, albeit the resulting formulation is still single exponential in size. In this chapter, we develop an algorithm that relies on decomposition techniques and that solves the QLP optimization problem. In Section 5.1 we introduce the concept of decomposition techniques for linear programs as a preparation for the main algorithm, which is then presented in Section 5.2. Finally, in Section 5.3 we describe the details of our implementation.

5.1. Decomposition Techniques

Large-scale linear programs typically have special structures that can be exploited to gain computational advantage in the solution process, like e.g. the *sparsity* of non-zero coefficients. When the DEP of a QLP or a MSSLP is constructed, the corresponding linear program grows exponentially with the number of random or universal variables, possibly reaching ten or hundred million decision variables and constraints. Thus, methods that ignore the special structure of the resulting system may become quite inefficient already for small instances. In order to tackle such problems, decomposition methods have been developed. These methods exploit the resulting special matrix block structure, and therefore decompose a problem into a master problem and one or more smaller subproblems that can be solved independently of each other. Decomposition methods work alternating on the master problem and the subproblem(s) and change information among them, until the original problem is solved to optimality. In this section we focus on structures that arise when the DEP of a QLP instance is constructed, as described in Section 4.1.3.

Therefore, consider the following linear program

$$\begin{array}{rlll} z = & \min & c^T x & \\ & s.t. & Ax \leq b & \\ & & x \in \mathbb{R}^n, & \end{array} \tag{5.1}$$

where A has a special structure.

The so-called *primal block-angular* structure looks as follows

$$Ax = \begin{pmatrix} A_0 & A_1 & A_2 & \dots & A_K \\ & B_1 & & & \\ & & B_2 & & \\ & & & \ddots & \\ & & & & B_K \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix} \leq \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_K, \end{pmatrix} \tag{5.2}$$

and is one of the most widely known and recognized structures. It typically results when the system being modeled naturally splits into a set of subsystems, e.g. a set of facilities or time periods, independent from a number of global constraints [214]. This structure also results if a two-stage or multi-stage QLP is transformed into a DEP using the scenario-based split-variable formulation (see Section 4.1.3). Note that in this case alls matrices $B_1, \dots, B_K$ have identical coefficients but correspond to different variables. However, the B_k matrix blocks need not to be identical in general in order to apply decomposition techniques.

Using the node-based compact-view representation to construct the DEP of a two-stage QLP (see Section 4.1.3), the resulting matrix structure is called *dual block-angular* and looks as follows

$$Ax = \begin{pmatrix} A_0 & & & & \\ A_1 & B_1 & & & \\ A_2 & & B_2 & & \\ \vdots & & & \ddots & \\ A_K & & & & B_K \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix} \leq \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_K \end{pmatrix}. \tag{5.3}$$

Note that in case of the DEP of a QLP the matrix blocks $A_1, \dots, A_k$ are equal but correspond to different matrices $B_1, \dots, B_K$, which have identical coefficients but correspond to different variables. As above, the matrix blocks need not to be identical in general in order to apply decomposition. It is easy to see that this structure is simply the transpose of the primal block angular structure. In the case of a DEP resulting from a multi-stage QLP, this structure appears nested (see Section 4.1.3).

As a preparation for the algorithm development we review two famous methods to solve such problems in the following – the Dantzig-Wolfe Decomposition, which can be used to solve system (5.2), and the Benders Decomposition, which can be used to solve system (5.3).

5.1.1. Dantzig-Wolfe Decomposition

The *Dantzig-Wolfe Decomosition Principle*, called *DW-method* in the following, was developed in the late 1950s [71] and is a classical solution approach for large-scale structured linear programming problems that could not be solved using standard simplex based solvers, since these problems exceeded their capacities. However, due to the capability of modern hardware and the current generation of modern simplex or interior point based

solvers, the DW-method has become less popular to solve LPs.

It can be applied to problems with a block-angular matrix structure as depicted in system (5.2), with $k = 1, \ldots, K$ independent subproblem blocks

$$\begin{aligned} B_k x_k &\leq b_k \\ x_k &\in \mathbb{R}^{n_k}, \end{aligned} \tag{5.4}$$

and a set of *coupling constraints*

$$\sum_{k=0}^{K} A_k x_k \leq b_0, \tag{5.5}$$

which can e.g. result from the nonanticipativity constraints of a split-view DEP formulation of a QLP or QIP.

The DW-method suggests to solve the K independent blocks individually and then adjusts the solution to take the interconnections imposed by the coupling constraints into account in order to solve the original problem (5.2). Its principal components are the transformation of the original problem into a so-called *DW-master* problem using *Minkowski's Representation Theorem*, the partitioning into several subproblems called *DW-subproblems*, and the solution of the DW-master using a *delayed column generation* algorithm. The latter starts with a valid solution of the DW-master and computes new columns by solving the DW-subproblem.

Lemma 5.1 (Minkowski's Representation Theorem)**.** *Given a feasible LP problem with solution space*

$$\mathcal{P}_{LP} = P(A, b). \tag{5.6}$$

Any point $x \in \mathcal{P}_{LP}$ can be written as a linear combination of its extreme points (see Definition 1.8) and extreme rays (see Definition 1.9), if the feasible region cannot be assumed to be bounded [1]. The expression

$$\begin{aligned} x = \sum_{l=1}^{L} \lambda_l p^l + \sum_{m=1}^{M} \mu_m r^m \\ \sum_{l=1}^{L} \lambda_l = 1 \\ \lambda_1, \ldots, \lambda_L \geq 0 \\ \mu_1, \ldots, \mu_M \geq 0 \end{aligned} \tag{5.7}$$

where p^l denotes the l-th extreme point and r^m denotes the m-th extreme ray of $\mathcal{P}_{LP}$ is called *Minkowski's Representation Theorem* [161].

Figure 5.1 shows a two-dimensional example with two extreme points $A = (2.0, 0.5)$ and $B = (0.5, 2.0)$ as well as two extreme rays r_1, r_2, with $r_1 = (2.0, 1.0)$ being anchored

[1] Note that by Definition 2.1 QLPs are bounded

at extreme point A and $r_2 = (1.0, 1.5)$ being anchored at extreme point B.

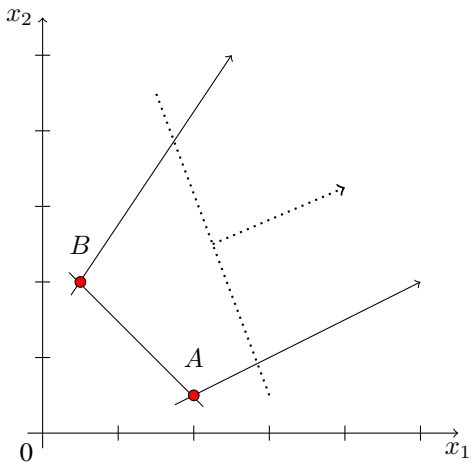


Figure 5.1.: Solution space of an unbounded LP in the two-dimensional space.

The DW-master problem is characterized by

$$\begin{aligned} \min \quad & c_0^T x_0 + \sum_{k=1}^{K} c_k^T x_k \\ s.t. \quad & A_0 x_0 + \sum_{k=1}^{K} A_k x_k \leq b_0 \\ & x \in \mathbb{R}^n, \end{aligned} \tag{5.8}$$

and by substituting equation (5.7) in system (5.8), we can reformulate the DW-master-problem to the effect that the original variables x_k that correspond to the K subproblems are replaced by the variables λ and μ

$$\begin{aligned} \min \quad & c_0^T x_0 + \sum_{k=1}^{K} (\sum_{l=1}^{L} (c_k^T p_k^l) \lambda_{k,l} + \sum_{m=1}^{M} (c_k^T r_k^m) \mu_{k,m}) \\ s.t. \quad & A_0 x_0 + \sum_{k=1}^{K} (\sum_{l=1}^{L} (A_k p_k^l) \lambda_{k,l} + \sum_{m=1}^{M} (A_k r_k^m) \mu_{k,m})) \leq b_0 \\ & \sum_{k=1}^{K} \sum_{l=1}^{L} \lambda_{k,l} = 1 \text{ for all } k = 1, \ldots, K \\ & x_0 \in \mathbb{R}^{n_0},\ \lambda_{k,l} \geq 0,\ \mu_{k,m} \geq 0. \end{aligned} \tag{5.9}$$

Compared to the original system (5.2) the number of constraints is reduced, however, since each of the subproblems can have exponentially many extreme points p_k^l and extreme rows r_k^m, system (5.9) can have a very large number of variables $\lambda_{k,l}$ and $\mu_{k,m}$. The basic property exploited by the DW-method is that most of these variables will be *non-basic* and set to zero in the optimal solution of system (5.9).

Therefore, the DW-method uses a special variant of the simplex algorithm that starts

with only a small number of variables and the corresponding LP is called *restricted master problem* (RMP). The RMP is solved for the current active set of variables and promising new variables (new columns) which price out some variables from the current active set are added during the execution of the algorithm. In the traditional simplex method, the pricing step is the only operation where all columns of the matrix are used. In the DW-method the pricing is done by solving the K subproblems (5.4), also called *pricing problems*. Their solution yields so-called *proposals* for the RMP and the attractiveness of a specific proposal $\lambda_{k,l}$ or $\mu_{k,m}$ can be quantified by its *reduced costs* σ_k. For the k-th subproblem this can be done by solving the following LP

$$\begin{aligned} SP_k(\pi_1): \ \sigma_k = \min_{x_k} \quad & (c^T - \pi_1^T A_k)x_k \\ s.t. \quad & B_k x_k \leq b_k \\ & x_k \in \mathbb{R}^{n_k}, \end{aligned} \tag{5.10}$$

which computes the most attractive basic feasible solution x_k to enter the RMP given the dual information π_1, π_2^k from the current solution of the RMP. π_1 are the dual variables that correspond to the coupling constraints and π_2^k the dual variables that correspond to the convexity constraint of the k-th pricing problem.

Let $\overline{\pi}_1, \overline{\pi}_2^k$ be the dual information from the optimal solution of the current RMP. Given an optimal solution $\overline{x}_k$ for $\text{SP}_k(\overline{\pi}_1)$ for the k-th pricing problem, then the following two cases are distinguished:

1. If system system (5.10) is bounded $(c^T - \pi_1^T A_k)\overline{x}_k < \overline{\pi}_2^k$, then the corresponding column $\overline{\lambda}_{k,i}$ can be introduced into the RMP, with a cost coefficient $c_k^T \overline{x}_k$.

2. If system system (5.10) is unbounded and $(c^T - \pi_1^T A_k)\overline{x}_k < 0$, then the corresponding column $\overline{\mu}_{k,j}$ can be introduced into the RMP, with a cost coefficient $c_k^T \overline{x}_k$.

After all pricing problems have been solved and at least one new column was added to the RMP, it is solved again and the process continues until an optimal solution is found when $\overline{\sigma}_k \geq \overline{\pi}_2^k$ holds for all $k = 1, \ldots, K$. Algorithm 6 illustrates the DW-method for an instance with K subproblems. Note that in the first step an initialization phase is passed, where an initial set of proposals and a solution that satisfies the coupling constraints (5.5) is computed. The latter is achieved by solving a phase I problem similar as it is also done in the simplex algorithm.

First efficient implementations of the algorithm were described in the 1980s [122, 120, 121]. For more details on many aspects of the DW-method we refer the reader to [214], where also a very detailed computational study can be found. Attempts to use the DW-method to solve stochastic (mixed-)integer programming problems in a scenario-based decomposition approach where reported in [123, 202]. Recently a computational study where strong dual bounds where obtained on general MIPs was proposed in [34], also the efficient parallelization of the solution process is subject of current research [180].

However, column generation does not seem to be practical for QLPs, because the number of rows in system (5.5), which result from the nonanticipativity constraints of the split-view DEP and determine the number of constraints in the DW-master problem, is almost

the same as the number of variables in the corresponding stages $t = 1, \ldots, H$ in the entire DEP problem. This fact has also been reported in the context of stochastic programming [71, 188], and it is one principal reason why other decomposition methods are used to solve these problems.

Algorithm 6 The Dantzig-Wolfe Decomposition Algorithm

Initialization of the restricted master problem (RMP):
  Choose initial subset of variables $\lambda_{k,i}$ and $\mu_{k,j}$ for the RMP;
  Find feasible solution to RMP which satisfies the coupling constraints;
if RMP is infeasible **then**
  return false: the problem instance is infeasible;
else
  while true **do**
    Solve the RMP to obtain dual solution;
    Let $\overline{\pi}_1$ and $\overline{\pi}_2^k$ be the duals of the coupling constraints and the k-th convexity constraint;
    for $k = 1$ to K **do**
      Solve subproblem SP$_k(\overline{\pi}_1)$;
      Let $\overline{x}_k$ be the optimal solution;
      if SP$_k(\overline{\pi}_1)$ is bounded and $(c^T - \pi_1^T A_k)\overline{x}_k < \overline{\pi}_2^k$ **then**
        Add the proposal $\overline{\lambda}_{k,l}$ with cost coefficient $c_k^T \overline{x}_k$ to the RMP;
      else if SP$_k(\overline{\pi}_1)$ is unbounded **then**
        Add the proposal $\overline{\mu}_{k,m}$ with cost coefficient $c_k^T \overline{x}_k$ to the RMP;
      end if
    end for
    if No new proposal $\lambda_{k,i}$ or $\mu_{k,j}$ generated **then**
      return true: optimal solution found
    end if
  end while
end if

5.1.2. Benders Decomposition

The *Benders Decomposition Priciple*, called *BD-method* in the following, was also developed in the late 1950s [25], and was originally proposed to solve mixed-integer linear programs. As in the case of the DW-method, the BD-method eliminates a part of the constraint system. However, whereas the former eliminates constraints, which are then re-introduced via a delayed column generation algorithm, the BD-method eliminates a set of variables, which are then re-introduced with a *delayed constraint generation* approach. From this viewpoint, the BD-method can be considered as the equivalent of the DW-method applied to the dual of system (5.2), which results in system (5.3).

To illustrate how the BD-method works, we first consider the special case $K = 1$ such that A and c of system (5.1) decompose into two parts

$$\begin{aligned} z = \min \quad & c_0^T x_0 + c_1^T x_1 \\ s.t. \quad & A_0 x_0 \qquad\quad \leq b_0 \\ & A_1 x_0 + B_1 x_1 \leq b_1 \\ & x_0 \in \mathbb{R}^{n_0},\ x_1 \in \mathbb{R}^{n_1}, \end{aligned} \tag{5.11}$$

with $A_0 \in \mathbb{R}^{m_0 \times n_0}$, $A_1 \in \mathbb{R}^{m_1 \times n_1}$, $B_1 \in \mathbb{R}^{m_1 \times n_1}$, $c_0 \in \mathbb{R}^{n_0}$, $c_1 \in \mathbb{R}^{n_1}$ with $n_0 + n_1 = n$, $m_0 + m_1 = m$.

Applying Benders Decomposition, the decision variables of system (5.11) are decomposed into a *restricted master problem (RMP)* and one *subproblem (SP)*, which are both w.l.o.g. assumed to be implicitly bounded in the constraint system (e.g. by setting artificial lower and upper bounds of suitable dimension). The RMP consists of the first stage variables x_0 and an approximation variable θ for the objective function part of the SP and looks as follows:

$$\begin{aligned} FMP: \min \quad & c_0^T x_0 + \theta \\ s.t. \quad & A_0 x_0 \qquad \leq b_0 \\ & x_0 \in \mathbb{R}^{n_0}. \end{aligned} \tag{5.12}$$

The SP, which contains the second-stage variables x_1 is written as:

$$\begin{aligned} SP(x_0): \; z^{SP} = \min \quad & c_1^T x_1 \\ s.t. \quad & B_1 x_1 \leq b - A_1 x_0 \\ & x_1 \in \mathbb{R}^{n_1}. \end{aligned} \tag{5.13}$$

The corresponding *dual SP (DSP)* has the property that the solution space no longer depends on the values x_0, regardless whether they are valid or invalid for the SP, only the objective function changes depending on the values x_0. The DSP has the following form:

$$\begin{aligned} DSP(x_0): \; z^{DSP} = \max \quad & \pi^T (b - A_1 x_0) \\ s.t. \quad & B_1^T \pi = c_1 \\ & \pi \leq 0. \end{aligned} \tag{5.14}$$

The solution of the current RMP yields a so-called *proposal* $(\overline{x}_0, \overline{\theta})$ for the SP and after substituting $\overline{x}_0$ into system (5.13) the following two cases can happen. If system (5.13) is feasible and bounded, then system (5.14) is also feasible and bounded and the solution π is located at the l-th extreme point for some $l \in L$, where L is the finite set of extreme points of the DSP. If system (5.13) is infeasible, then its dual (5.14) must be unbounded[2], and the solution corresponds to the m-th extreme ray for some $m \in M$, where M is the finite set of extreme rays of the DSP. Using this dual information, two different types of cutting planes – called *benders cuts* – can be added to the RMP to cutoff the current proposal $(\overline{x}_0, \overline{\theta})$ when the RMP is solved the next time. The standard BD-method distinguishes the following two types of cuts:

- A *feasibility cut* is a linear constraint which ensures that a first stage decision becomes second stage feasible.
- An *optimality cut* is a linear approximation of the second-stage objective function and provides a lower bound on θ for the current proposal.

Given a proposal $(\overline{x}_0, \overline{\theta})$, obtained from the solution of the RMP, these cuts can be derived as follows. As mentioned above, the role of the feasibility cut is to ensure that

[2]It cannot be infeasible, since we assumed that the variables of system (5.13) are bounded.

a first-stage decision becomes second-stage feasible. If a proposal $\overline{x}_0$ is not valid for the SP($\overline{x}_0$), than the DSP($\overline{x}_0$) is unbounded and for the corresponding extreme ray $\overline{r}^m$ holds

$$\overline{z}^{DSP} = (\overline{r}^m)^T(b - A_1\overline{x}_0) > 0. \tag{5.15}$$

Therefore, in order to ensure the feasibility of the second stage, the feasibility cut

$$(\overline{r}^m)^T(b - A_1x_0) \leq 0, \tag{5.16}$$

which cuts off the invalid solution, must be added to the RMP to avoid the current proposal $\overline{x}_0$ in the next iteration.

On the other hand, if the SP($\overline{x}_0$) has an optimal solution with objective function value $\overline{z}^{SP}$, then also the DSP($\overline{x}_0$) has an optimal solution $\overline{z}^{DSP}$ that corresponds to the l-th extreme point, which is denoted by $\overline{p}^l$ in the following. By strong duality $\overline{z}^{DSP} = \overline{z}^{SP}$ holds, and if $\overline{z}^{DSP} > \overline{\theta}$, then

$$(\overline{p}^l)^T(b - A_1\overline{x}_0) > \overline{\theta}. \tag{5.17}$$

In this case, the proposal $(\overline{x}_0, \overline{\theta})$ has violated the constraint

$$(\overline{p}^l)^T(b - A_1x_0) \leq \theta, \tag{5.18}$$

which is the corresponding optimality cut that must be added to the RMP in order to avoid the current proposal $\overline{x}_0$ in the next iteration.

Since the DSP can only have finitely many extreme points and extreme rays [161], the *full master problem (FMP)* can be written as follows

$$\begin{array}{rlll} FMP: \min & c_0^Tx_0 + \theta & & \\ s.t. & A_0x_0 & \leq b_0 & \\ & (r^m)^T(b - A_1x_0) & \leq 0 \; \forall m \in M & \\ & (p^l)^T(b - A_1x_0) & \leq \theta \; \forall l \in L & \\ & x_0 \in \mathbb{R}^{n_0}, & & \end{array} \tag{5.19}$$

where θ is the auxiliary variable used to represent the objective function value of the SP. This formulation is equivalent to the original problem (5.11), however, there can be exponentially many extreme rays and extreme points and not all of them might be needed to find the optimal solution. Therefore, the BD-method starts solving a restricted master problem RMP

$$\begin{array}{rlll} RMP: \min & c_0^Tx_0 + \theta & & \\ s.t. & A_0x_0 & \leq b_0 & \\ & (r^m)^T(b - A_1x_0) & \leq 0 \;\text{ for some } m \in M & \\ & (p^l)^T(b - A_1x_0) & \leq \theta \;\text{ for some } l \in L & \\ & x_0 \in \mathbb{R}^{n_0}, & & \end{array} \tag{5.20}$$

with no (or only a few) cuts from L and M being used at the beginning. It computes cuts if

the current proposal violates any of the non-included constraints in an iterative process until an optimal solution is found or infeasibility is detected. In the latter case also system (5.11) is infeasible. The optimal solution is found, if for a given proposal $(\overline{x}_0, \overline{\theta})$ also the SP($\overline{x}_0$) has an optimal solution with value $\overline{z}^{SP}$ and the optimality condition $\overline{\theta} = \overline{z}^{SP}$ is satisfied. If this is the case, the BD-method stops. Otherwise, a feasibility or optimality cut is added to the RMP, which is then re-solved again to obtain a new proposal. In each iteration where the SP is feasible, $c_0^T\overline{x}_0 + \overline{\theta}$ yields a lower bound for the initial problem, while $c_0^T\overline{x}_0 + \overline{z}^{SP}$ yields an upper bound. The difference between these bounds gets smaller, and if it becomes less than a predefined ε, the BD-method terminates and an optimal solution $(\overline{x}_0, \overline{x}_1)$ for the original system (5.11) is found.

Thus far, we have only considered the special case for $K = 1$, however, the BD-method can be also applied when there are multiple independent blocks A_k as in system (5.3). In this case there is one approximation variable θ_k for each block $k = 1, \ldots, K$, and the RMP can be formulated as follows

$$\begin{array}{rlll} RMP: \min & c_0^T x_0 + \sum\limits_{k=1}^{K} \theta_k & & \\ s.t. & A_0 x_0 & \leq b_0 & \\ & (r_k^m)^T(b_k - A_k x_0) & \leq 0 \text{ for some } m \in M_k \text{ and } k \in K & \\ & (p_k^l)^T(b_k - A_k x_0) & \leq \theta_k \text{ for some } l \in L_k \text{ and } k \in K & \\ & x_0 \in \mathbb{R}^{n_0}, & & \end{array} \tag{5.21}$$

whereas the k-th subproblem can be written as

$$\begin{array}{rlll} SP_k(x_0): \; z_k^{SP} = \min & c_k^T x_k & & \\ s.t. & B_k x_k \leq b_k - A_k x_0 & & \\ & x_k \in \mathbb{R}^{n_k}. & & \end{array} \tag{5.22}$$

In each iteration up to K cuts are added to the RMP. The feasibility cuts from the subproblems all restrict the same variable x_0, but each subproblem SP_k adds optimality cuts $(p_k^l)^T(b_k - A_1 x_0) \leq \theta_k$ for some $l \in L_k$ only for the approximation variable θ_k that corresponds to SP_k. The BD-method terminates when the optimality condition is satisfied for all subproblems. As it is well known, the Benders Decomposition method converges to an optimal solution in a finite number of iterations or proves the infeasibility of the original problem, if the number of subproblems is limited because each of them only has a finite number of extreme points and extreme rays [161]. Algorithm 7 illustrates the BD-method for an instance with K subproblems.

In this example the BD-method was applied to a simple linear program, however, it can be applied to a much broader class of problems. In the two-stage case we only add linear constraints to the RMP and use its solution as a proposal for the second stage. Thus, the RMP needs not be a linear program, but can also be integer program (see e.g. [25]), a nonlinear program (see e.g. [62]), or a constraint programming problem (see e.g. [93, 33]). Furthermore, since the subproblem is only used to obtain dual information to generate cuts for the RMP, it needs not to be a linear program at all, but can be an arbitrary convex program [107]. Benders' Decomposition is to date the basis of many algorithms that have

been developed in the stochastic programming community over the past decades. For the two-stage case most proposed methods trace back to the L-shaped method of Van Slyke and Wets [219] and this method has also been successfully extended to the multi-stage case by Birge [45]. Today, some of the most efficient solution approaches for stochastic programming problems rely on these basics. Over the past decades many improvements have been suggested in the literature (see also Section 2.2.2). The vast amount of theoretical basics and computational experiences motivated us to adopt the Benders Decomposition principle and some of its extensions to solve two-stage and multi-stage QLPs.

Algorithm 7 The Benders Decomposition Algorithm

while true **do**
  Solve the RMP;
  if RMP is infeasible **then**
    return false: the problem instance is infeasible;
  else
    Let $(\overline{x}_0, \overline{\theta}_1, \ldots, \overline{\theta}_K)$ be the optimal solution;
  end if
  for $k = 1$ to K **do**
    Solve subproblem $\text{SP}_k(\overline{x}_0)$;
    if $\text{SP}_k(\overline{x}_0)$ is infeasible **then**
      Let $\overline{r}_k^m$ be the corresponding dual extreme ray;
      Add the feasibility cut $(\overline{r}_k^m)^T(b_k - A_k x_0) \leq 0$ to the RMP;
    else if $\text{SP}_k(\overline{x}_0)$ has an optimal solution **then**
      Let $\overline{z}_k^{SP}$ be the objective function value and $\overline{p}_k^l$ the corresponding dual extreme point;
      if $\overline{z}_k^{SP} > \overline{\theta}_k$ **then**
        Add the optimality cut $(\overline{p}_k^l)^T(b_k - A_k x_0) \leq \theta_k$ to the RMP;
      end if
    end if
  end for
  if No feasibility cuts were added **and** $\overline{z}_k^{SP} \leq \overline{\theta}_k \; \forall k = 1, \ldots, K$ **then**
    return true: optimal solution found
  end if
end while

5.2. Decomposition Algorithms for QLPs

In this section, we extend the BD-method to solve QLPs. We start with the algorithmic description of an extension to solve two-stage QLP optimization problems. Afterwards, we extend the proposed algorithm to work in a nested application in order to solve multi-stage QLP optimization problems. Finally, we integrate techniques from game-tree search, exploiting the minimax property of the universal player decision-tree, as well as a bounding scheme based on the solution of QLP-relaxations.

5.2.1. Benders Decomposition for Two-Stage QLPs

In Section 2.2.2 we mentioned that the Benders Decomposition principle as described in the preceding section was also adapted in the context of stochastic programming to solve two-stage stochastic linear programming problems (see e.g. [219]). In the following we show how the BD-method can be applied to solve two-stage QLPs and therefore resort to the recursive definition of two-stage QLPs as proposed in Section 4.1.3. Recall that a two-stage QLP can be written as follows

$$\begin{aligned} z = \min_{x^1_\exists} \ & (c^1_\exists)^T x^1_\exists + \mathcal{Q}^1(x^1_\exists) \\ s.t. \ & W^1_\exists x^1_\exists \leq h^1 \\ & l^1_\exists \leq x^1_\exists \leq u^1_\exists, \end{aligned}$$

where the worst-case second-stage value function $\mathcal{Q}^1(x^1_\exists)$ is defined as

$$\mathcal{Q}^1(x^1_\exists) = \max_{\forall x^1_\forall \in E} [Q^2(x^1_\exists, x^1_\forall)],$$

which is the maximum loss obtained from the second-stage scenarios resulting from the universal variable vectors in the discrete set E. Given a first-stage proposal $x^1_\exists$ and any realization of the universal variables $x^1_\forall$, the resulting second stage subproblem can be written as

$$\begin{aligned} Q^2(x^1_\exists, x^1_\forall) = (c^1_\forall)^T x^1_\forall + \min_{x^2_\exists} \ & (c^2_\exists)^T x^2_\exists \\ s.t. \ & W^2_\exists x^2_\exists \leq \underbrace{h^2 - T^1_\forall x^1_\forall}_{:= h^2(x^1_\forall)} - T^1_\exists x^1_\exists \\ & l^2_\exists \leq x^2_\exists \leq u^2_\exists, \end{aligned}$$

where $(c^1_\forall)^T x^1_\forall$ and $h^2(x^1_\forall)$ are the scenario dependent objective function offset and the right-hand side respectively that result from the corresponding realization of the universal variables $x^1_\forall$.

As an introduction for the algorithm development, we first provide some more notation and then describe a basic version of the BD-method for two-stage QLPs. We can w.l.o.g. assume that a QLP instance starts and ends with an existential variable block. Solving a two-stage QLP can be interpreted as solving the universal player decision-tree (see Section 4.1.3) with depth $H = 2$. Therefore, the tree is traversed multiple times solving nodal LPs and with information being passed between the root node at stage $t = 1$ an the leaf nodes at stage $t = 2$.

If there are $|x_\forall|$ universally quantified variables, then there are $k = 1, \ldots, K$ subproblems with $K = 2^{|x_\forall|}$ that correspond to the leaf nodes of the universal player decision-tree and that must be solved in in each iteration. The k-th subproblem corresponds to the k-th

vector of universal variables $x_{\forall}^{1,k}$ in the discrete set

$$E = \prod_{i=1}^{|x_{\forall}|} \{l_{\forall i}, u_{\forall i}\} = \{l_{\forall 1}, u_{\forall 1}\} \times \{l_{\forall 2}, u_{\forall 2}\} \times \cdots \times \{l_{\forall k}, u_{\forall k}\}. \tag{5.23}$$

The resulting subproblems only differ in their right-hand sides $h^2(x_{\forall}^{1,k}) = h^2 - T_{\forall}^1 x_{\forall}^{1,k}$ and in the constant universal variable objective function offset $(c_{\forall}^1)^T x_{\forall}^1$. The recourse matrix $W_{\exists}^2$ and the first-stage matrix $T_{\exists}^1$ are fixed and identical for each subproblem. Thus, the k-th subproblem can be formulated as follows

$$\begin{aligned} \mathcal{Q}^2(x_{\exists}^1, x_{\forall}^{1,k}) : \; z_{\mathcal{Q}}^{2,k} = (c_{\forall}^1)^T x_{\forall}^{1,k} + \min \quad & (c_{\exists}^2)^T x_{\exists}^{2,k} \\ s.t. \quad & W_{\exists}^2 x_{\exists}^{2,k} \leq h^2(x_{\forall}^{1,k}) - T_{\exists}^1 x_{\exists}^1 \\ & l_{\exists}^2 \leq x_{\exists}^{2,k} \leq u_{\exists}^2, \end{aligned} \tag{5.24}$$

The dual information of $\mathcal{Q}^2(x_{\exists}^1, x_{\forall}^{1,k})$ is needed to generate feasibility and optimality cuts and can be formulated as follows

$$\begin{aligned} \mathcal{Q}_D^2(x_{\exists}^1, x_{\forall}^{1,k}) : \; z_{\mathcal{Q}}^{2,k} = (c_{\forall}^1)^T x_{\forall}^{1,k} + \max \quad & \pi_k^T (h^2(x_{\forall}^{1,k}) - T_{\exists}^1 x_{\exists}^1) + \lambda_k^T l_{\exists}^2 + \mu_k^T u_{\exists}^2 \\ s.t. \quad & (W_{\exists}^2)^T \pi_k + \lambda_k + \mu_k = c_{\exists}^2 \\ & \pi_k \leq 0, \lambda_k \geq 0, \mu_k \leq 0, \end{aligned} \tag{5.25}$$

where the dual variables that correspond to the bounds of the variables in the primal subproblem are explicitly specified since we need this level of detail to explain the characteristic features of our implementation later (we assumed these bounds to be implicitly encoded into the constraint system in Section 5.1.2). Whereas all variables were assumed to be non-negative in the original L-shaped description this formulation is used to allow arbitrary (finite) variable bounds. After solving the k-th subproblem we obtain a set of dual values $(\pi_k, \lambda_k, \mu_k)$, where π_k corresponds to the original rows of the primal subproblem, λ_k corresponds to the lower bounds of the variables and μ_k corresponds to the upper bounds respectively. Given a fixed set of dual values $\overline{r}_k^m = (\overline{\pi}_k^m, \overline{\lambda}_k^m, \overline{\mu}_k^m)$ representing the m-th extreme ray of an an unbounded subproblem k, the corresponding feasibility cut can be computed as follows

$$\begin{aligned} & \underbrace{\left((\overline{\pi}_k^m)^T h^2(x_{\forall}^{1,k}) + (\overline{\lambda}_k^m)^T l_{\exists}^2 + (\overline{\mu}_k^m)^T u_{\exists}^2 \right)}_{=:(\overline{r}_k^m)^T h_k^2} - (\overline{\pi}_k^m)^T (T_{\exists}^1 x_{\exists}^1) \leq 0 \\ \Leftrightarrow \quad & (\overline{\pi}_k^m)^T (T_{\exists}^1 x_{\exists}^1) \geq (\overline{r}_k^m)^T h_k^2. \end{aligned} \tag{5.26}$$

Accordingly, given a fixed set of dual values $\overline{p}_k^l = (\overline{\phi}_k^l, \overline{\lambda}_k^l, \overline{\mu}_k^l)$ representing the l-th extreme point of a bounded subproblem k, the corresponding optimality cut can be computed

as follows

$$\begin{aligned} & \underbrace{\left((\overline{\pi}_k^l)^T h^2(x_\forall^{1,k}) + (\overline{\lambda}_k^l)^T l_\exists^2 + (\overline{\mu}_k^l)^T u_\exists^2\right)}_{=:(\overline{p}_k^l)^T h_k^2} - (\overline{\pi}_k^l)^T (T_\exists^1 x_\exists^1) \quad \leq \theta_k \\ \Leftrightarrow \qquad & (\overline{\pi}_k^l)^T (T_\exists^1 x_\exists^1) + \theta_k \quad \geq (\overline{p}_k^l)^T h_k^2. \end{aligned} \tag{5.27}$$

The biggest difference to the formulation solved by the original BD-method as illustrated in Section 5.1.2 is that if the entire problem is feasible, the second-stage value function is neither a simple sum of the objective function parts of the K subproblems, nor a weighted-sum as in the case of an TSSLP, but the maximum of them

$$\max_{\forall x_\forall^1 \in E} [Q^2(x_\exists^1, x_\forall^1)]. \tag{5.28}$$

In order to achieve the *min-max-min* property of the objective function in the two-stage case, we modify Algorithm 7 as follows. If a subproblem is infeasible, a feasibility cut is computed and added to the RMP as before. But if a subproblem is feasible, it yields an optimality cut that provides a lower bound on the corresponding approximation variable θ_k. Since we are looking for the worst-case θ_k, which is the one with the maximum value among all θ_k for $k = 1 \ldots, K$, we can simply share one single approximation variable θ that is restricted by all optimality cuts that result from the K subproblems simultaneously. Thus, the corresponding RMP can be written as follows

$$\begin{aligned} RMP: \; z_\mathcal{Q}^1 = \min \quad & (c_\exists^1)^T x_\exists^1 + \theta & & \\ s.t. \quad & W_\exists^1 x_\exists^1 & \leq h_1 & \\ & (\overline{\pi}_k^m)^T (T_\exists^1 x_\exists^1) & \geq (\overline{r}_k^m)^T h_k^2, & \text{for some } m \in M_k \text{ and } k \in K \\ & (\overline{\pi}_k^l)^T (T_\exists^1 x_\exists^1) + \theta & \geq (\overline{p}_k^l)^T h_k^2, & \text{for some } l \in L_k \text{ and } k \in K \\ & l_\exists^1 \leq x_\exists^1 \leq u_\exists^1. & & \end{aligned} \tag{5.29}$$

Given a QLP instance, the solution of the RMP generates a proposal $(\overline{x}_\exists^1, \overline{\theta})$ in each iteration, and the corresponding objective function value

$$Z_{LB} := \overline{z}_\mathcal{Q}^1 = (c_\exists^1)^T \overline{x}_\exists^1 + \overline{\theta} \tag{5.30}$$

yields a lower bound for the original QLP. While there are infeasible subproblems, the corresponding feasibility cuts are added to the RMP, which is then solved again. After all subproblems have been solved to optimality for a given proposal $(\overline{x}_\exists^1, \overline{\theta})$,

$$Z_{UB} = (c_\exists^1)^T \overline{x}_\exists^1 + \max\{\overline{z}_\mathcal{Q}^{2,1}, \overline{z}_\mathcal{Q}^{2,2}, \ldots, \overline{z}_\mathcal{Q}^{2,K}\} \tag{5.31}$$

yields an upper bound and it holds

$$Z_{LB} \leq \overline{z} \leq Z_{UB}, \tag{5.32}$$

where $\overline{z}$ is the optimal objective function value of the original QLP instance.

Algorithm 8 describes the BD-method for a two-stage QLP instance. Until the difference between the upper and lower bound does not fall below a predefined ε, the algorithm iterates and first the RMP is solved in line 3. If it becomes infeasible also the original QLP instance is declared infeasible in line 4 – 5. Otherwise, a new proposal is obtained in line 7 and the objective function value is used to update the lower bound in line 8. Afterwards, all subproblems are solved with respect to the current proposal as described in lines 10 – 22. If a subproblem is infeasible, the corresponding feasibility cut is added in lines 12 – 14. If a subproblem is feasible and the corresponding objective function value $\overline{z}_{\mathcal{Q}}^{2,k}$ is larger than the approximation variable $\overline{\theta}$ from the current proposal of the RMP, then the corresponding optimality cut is added to the RMP as shown in lines 15 –21. If all subproblems are feasible, the upper bound is updated in line 24 and the algorithm proceeds with the next iteration until the algorithm terminates eventually when the bounds coincide or the RMP becomes infeasible. When the algorithm terminates with an optimal solution in line 27, the current proposal $(\overline{x}_{\exists}^1)$ yields the optimal first-stage variable assignment for the original QLP instance, and $\overline{z}_{\mathcal{Q}}^1$ is the corresponding optimal objective function value.

Algorithm 8 Benders Decomposition Algorithm for Two-Stage QLPs.

1: $Z_{LB} = -\infty$, $Z_{UB} = +\infty$
2: **while** $|Z_{UB} - Z_{LB}| \geq \epsilon$ **do**
3: Solve the RMP;
4: **if** RMP is infeasible **then**
5: **return false**: The problem instance is infeasible;
6: **else**
7: Let $(\overline{x}_{\exists}^1, \overline{\theta})$ be the proposal and $\overline{z}_{\mathcal{Q}}^1$ the corresponding objective function value;
8: Update the lower bound: $Z_{LB} = \overline{z}_{\mathcal{Q}}^1$
9: **end if**
10: **for** $k = 1$ to K **do**
11: Solve subproblem $\mathcal{Q}^2(\overline{x}_{\exists}^1, x_{\forall}^{1,k})$
12: **if** $\mathcal{Q}^2(\overline{x}_{\exists}^1, x_{\forall}^{1,k})$ is infeasible **then**
13: Let $\overline{r}_k^m = (\overline{\pi}_k^m, \overline{\lambda}_k^m, \overline{\mu}_k^m)$ be the corresponding dual extreme ray ;
14: Add the feasibility cut $(\overline{\pi}_k^m)^T(T_{\exists}^1 x_{\exists}^1) \geq (\overline{r}_k^m)^T h_k^2$ to the RMP;
15: **else if** $\mathcal{Q}^2(\overline{x}_{\exists}^1, x_{\forall}^{1,k})$ has an optimal solution **then**
16: Let $\overline{z}_{\mathcal{Q}}^{2,k}$ be the corresponding objective function value;
17: Let $\overline{p}_k^l = (\overline{\phi}_k^l, \overline{\lambda}_k^l, \overline{\mu}_k^l)$ be the corresponding dual extreme point;
18: **if** $\overline{z}_{\mathcal{Q}}^{2,k} > \overline{\theta}$ **then**
19: Add the optimality cut $(\overline{\pi}_k^l)^T(T_{\exists}^1 x_{\exists}^1) + \theta \geq (\overline{p}_k^l)^T h_k^2$ to the RMP;
20: **end if**
21: **end if**
22: **end for**
23: **if** $\mathcal{Q}^2(\overline{x}_{\exists}^1, x_{\forall}^{1,k})$ is feasible for all $k = 1, \ldots, K$ **then**
24: Update the upper bound: $Z_{UB} = (c_{\exists}^1)^T \overline{x}_{\exists}^1 + \max\{\overline{z}_{\mathcal{Q}}^{2,1}, \overline{z}_{\mathcal{Q}}^{2,2}, \ldots, \overline{z}_{\mathcal{Q}}^{2,K}\}$;
25: **end if**
26: **end while**
27: **return true**: Objective function value $\overline{z}_{\mathcal{Q}}^1$ and first-stage variable assignment $\overline{x}_{\exists}^1$;

5.2.2. Nested Benders Decomposition for Multi-Stage QLPs

Similar as in the case of the the L-shaped method, which was extended to the multi-stage case by Birge [45], we can extend Algorithm 8 to handle QLP optimization problems with more than two stages ($1 < t < H$). The key idea of the Nested Benders Decomposition is the observation that a problem with Hstages has the same form as the corresponding two-stage problem where the first $H - 1$ stages are considered as one problem. Thus, solving a multi-stage QLP can be achieved by a recursive application of the BD-method forming a chain of consecutive subproblems. This can be interpreted as solving the universal player decision-tree (see Section 4.1.3), which is therefore traversed multiple times solving nodal LPs and with information being passed between adjoined nodes of the tree. The information that are passed down between a node at stage t and its successors at stage $t + 1$ are proposals for the first t variable blocks of the existential variables $(x^1_\exists, x^2_\exists, \ldots, x^t_\exists)$. Dual information from the node's successors at stage stage $t + 1$ are passed up the tree to compute feasibility and optimality cuts, which are then added to the nodal LP at stage t.

In the following, we extend Algorithm 8 to the multi-stage case and as a preparation we first provide some additional notation, similar as it is used in [45]. We can again w.l.o.g. assume that a QLP instance starts with an existential variable block and ends with an existential variable block. When solving a multi-stage QLP the corresponding universal player decision-tree has $t = 1, \ldots, H$ stages, where H is the number of existential variable blocks $x^t_\exists$. A tree node v^t_i is denoted by its stage t and an index $i = 1, \ldots, K^t$, where K^t denotes the number of nodes at stage t. A node v^t_i at stage t is implicitly specified by the set of fixed universal variable vectors

$$x^{t-1,i}_\forall = (\overline{x}^1_\forall, \overline{x}^2_\forall, \ldots, \overline{x}^{t-1}_\forall), \tag{5.33}$$

which correspond to a specific path from the root to a particular node i at stage t. Further input parameters are the set of existential variable assignments from the root at stage $t = 0$ to the current node v^t_i at stage t

$$x^{t-1,i}_\exists = (\overline{x}^1_\exists, \overline{x}^2_\exists, \ldots, \overline{x}^{t-1}_\exists), \tag{5.34}$$

and each nodal LP is optimized with respect to these values. At stage $t > 1$ we use the notation of $a(i, t)$ to denote the ancestor of a node v^t_i and at stage $t < H$ we use the notation $d(i, t)$ for the set of its descendants in order to refer to the respective parent and child nodes of v^t_i. The number of universal variables $|x^t_\forall|$ at stage $t < H$ determines the number of outgoing edges for all nodes i at stage t, where each edge corresponds to one of the lower and upper bound combinations from the discrete set

$$E^t = \prod_{i=1}^{|x^t_\forall|} \{l^t_{\forall i}, u^t_{\forall i}\}. \tag{5.35}$$

Each node v^t_i has a corresponding LP attached that is used to determine the variable assignment of the existential player at this node (in other words the next move in the game).

The root node at stage $t = 1$ corresponds to a restricted master problem as described by system (5.29). The leaf nodes at stage $t = H$ correspond to LP subproblems as described by system (5.24) or its dual system (5.25) respectively as known from the two-stage case. For all other stages $t = 2, \ldots, H-1$ the linear program attached to a specific node v_i^t, denoted by RMP_i^t, acts as a restricted master problem for the current node and is therefore modified by the cuts from the set of descendants $v_j^{t+1} \in d(i,t)$. At the same time it acts as a subproblem for the ancestor $a(i,t)$ of node v_i^t and the dual information obtained from the solution of the RMP_i^t are used to generate benders cuts for the restricted master problem $\text{RMP}_{a(i,t)}^{t-1}$. The RMP_i^t can be formulated as follows

$$
\begin{array}{lllll}
\mathcal{Q}^{t,i}(x_\exists^{t-1,a(i,t)}, x_\forall^{t-1,i}) & : & & & \\
z_\mathcal{Q}^{t,i} = (c_\forall^{t-1})^T x_\forall^{t-1,i} & + & \min\ (c_\exists^t)^T x_\exists^{t,i} & + & \theta^{t,i} \\
s.t. \quad T_\exists^{t-1} x_\exists^{t-1,a(i,t)} & + & W_\exists^t x_\exists^{t,i} & \leq & h^t(x_\forall^{t-1,i}) \\
 & & D_m^{t,i} x_\exists^{t,i} & \geq & d_m^{t,i},\ \text{for some } m \in M_j \\
 & & & & \text{and } v_j^{t+1} \in d(i,t) \\
 & & E_l^{t,i} x_\exists^{t,i} + \theta^{t,i} & \geq & e_l^{t,i},\ \text{for some } l \in L_j \\
 & & & & \text{and } v_j^{t+1} \in d(i,t) \\
 & & l_\exists^t \leq x_\exists^t \leq u_\exists^t, & &
\end{array}
\tag{5.36}
$$

where the corresponding vector of existential decision variables is denoted by $x_\exists^{t,i} \in \mathbb{Q}^{n_t}$ and the input parameters $x_\exists^{t-1,a(i,t)}$ and $x_\forall^{t-1,i}$ indicate the current state of the system, e.g. the history of all existential and universal moves from stage 1 up to stage $t-1$. The constant objective function offset $(c_\forall^{t-1})^T x_\forall^{t-1,i}$ and the right-hand side vector $h^t(x_\forall^{t-1,i})$ depend on the universal decision variables $x_\forall^{t-1,i}$ of the previous stages. The matrix $T^{t-1} \in \mathbb{Q}^{m_t - 1 \times n_{t-1}}$ belongs to the existential decision variables $x_\exists^{t-1,a(i,t)}$ of the previous stages $1, \ldots, t-1$, whereas the recourse matrix $W^t \in \mathbb{Q}^{m_t \times n_t}$ belongs to the existential decision variables $x_\exists^{t,i}$ of stage t. Note that since those matrices and the objective coefficients are identical for all nodes i at a given stage t, we can drop the i indices for sake of simplicity in the following.

Note that the nodal LP at an inner node is at the same the restricted master problem for the node's direct successors and also the subproblem for the node's direct ancestor. Thus, the problem can contain feasibility and optimality cuts that must also be considered in the cut generation process. The current set of feasibility cuts obtained from the successors $v_j^{t+1} \in d(i,t)$ of a node v_i^t are denoted by

$$
D_m^{t,i} x_\exists^{t,i} \geq d_m^{t,i},\ \text{for some } m \in M_j \text{ and } v_j^{t+1} \in d(i,t), \tag{5.37}
$$

whereas

$$
E_l^{t,i} x_\exists^{t,i} + \theta^{t,i} \geq e_l^{t,i},\ \text{for some } l \in L_j \text{ and } v_j^{t+1} \in d(i,t), \tag{5.38}
$$

denote the current optimality cuts. Given the solution of a subproblem obtained from a node $v_j^{t+1} \in d(i,t)$, these cuts are computed as follows. If the corresponding RMP_j^{t+1} has an optimal primal solution, we obtain the corresponding dual extreme point $\overline{p}_j^{t+1} =$

$(\overline{\pi}_j^{t+1}, \overline{\rho}_j^{t+1}, \overline{\sigma}_j^{t+1}, \overline{\lambda}_j^{t+1}, \overline{\mu}_j^{t+1})$, where the values $\overline{\pi}_j^{t+1}$ correspond to the original rows of the problem, $\overline{\rho}_j^{t+1}$ to the feasibility cuts and $\overline{\sigma}_j^{t+1}$ to the optimality cuts. The values $\overline{\lambda}_j^{t+1}$ correspond to the lower bound variables and $\overline{\mu}_j^{t+1}$) to the upper bound variables respectively. We can compute the optimality cut $E_l^{t,i} x_\exists^{t,i} + \theta^{t,i} \geq e_l^{t,i}$ as follows

$$\begin{aligned} E_l^{t,i} =& (\overline{\pi}_j^{t+1})^T T_\exists^t \\ e_l^{t,i} =& (\overline{\pi}_j^{t+1})^T h^{t+1}(x_\forall^{t,j}) + (\overline{\rho}_j^{t+1})^T d^{t+1,j} + \\ & (\overline{\sigma}_j^{t+1})^T e^{t+1,j} + (\overline{\lambda}_j^{t+1})^T l_\exists^{t+1} + (\overline{\mu}_j^{t+1})^T u_\exists^{t+1}. \end{aligned}$$

If the subproblem is infeasible we compute the feasibility cut $D_m^{t,i} x_\exists^{t,i} \geq d_m^{t,i}$ with the corresponding dual ray $\overline{r}_j^{t+1} = (\overline{\pi}_j^{t+1}, \overline{\rho}_j^{t+1}, \overline{\sigma}_j^{t+1}, \overline{\lambda}_j^{t+1} \overline{\mu}_j^{t+1})$ as follows

$$\begin{aligned} D_m^{t,i} =& (\overline{\pi}_j^{t+1})^T T_\exists^t \\ d_m^{t,i} =& (\overline{\pi}_j^{t+1})^T h^{t+1}(x_\forall^{t,j}) + (\overline{\rho}_j^{t+1})^T d^{t+1,j} + \\ & (\overline{\sigma}_j^{t+1})^T e^{t+1,j} + (\overline{\lambda}_j^{t+1})^T l_\exists^{t+1} + (\overline{\mu}_j^{t+1})^T u_\exists^{t+1}. \end{aligned}$$

Note that even though both formulas look identical except the approximation variable $\theta^{t,i}$, in case of an unbounded subproblem the dual information is an extreme ray (a direction settled at a point int the solution space of the dual subproblem), while in case of an optimal solution it is an extreme point in the solution space of the dual subproblem.

Function $\mathcal{Q}^{t,i}(x_\exists^{t-1,a(i,t)}, x_\forall^{t-1,i})$ as formally described in Algorithm 9 illustrates the recursive application of the Nested Benders Decomposition Algorithm for multi-stage QLPs. When solving a QLP instance the algorithm proceeds as follows. At the beginning for $t = 1$, the function $\mathcal{Q}^{0,1}(x_\exists^{0,a(1,1)}, x_\forall^{0,1})$ is initialized with two empty vectors. When a node v_i^t is visited, first the upper and lower bound are initialized in line 1. Afterwards, the restricted master problem RMP_i^t is solved in line 3. When the RMP_i^t is infeasible, three possible cases must be checked as described in lines 5 – 11: v_i^t is the root node ($t = 1$), v_i^t is a leaf node ($t = T$), or v_i^t is an inner node ($1 < t < H$). If v_i^t is the root node, then the original QLP instance is declared infeasible in line 6. Otherwise, dual information (depending on whether the node is an inner node or a leaf node) are passed back to the ancestor node $a(i,t)$ as described in lines 7 – 11. If the RMP_i^t is feasible, the lower bound is updated with the corresponding objective function value $\overline{z}_\mathcal{Q}^{t,i}$ in line 14 and afterwards the current proposal $\overline{x}_\exists^{t,i}$ is passed to the successors $v_j^{t+1} \in d(i,t)$ of node v_i^t. If the nodal LP of a successor v_j^{t+1} turns out to be infeasible for the current proposal, dual information are used to compute a feasibility cut, which is then added to the RMP_i^t in lines 17 – 20. Otherwise, if the nodal LP has an optimal solution and the corresponding objective function value $\overline{z}_\mathcal{Q}^{t+1,j}$ is greater than the current objective function approximation variable $\overline{\theta}^{t,i}$, dual information are used to add an optimality cut to the RMP_i^t in lines 21 – 27. After all subproblems are solved and if all of them are feasible, the upper bound is updated in line 31. Until the difference between the upper and lower bound at the current node does

Algorithm 9 Function $\mathcal{Q}^{t,i}(x_{\exists}^{t-1,a(i,t)}, x_{\forall}^{t-1,i})$

$Z_{LB}^{t,i} = -\infty$, $Z_{UB}^{t,i} = +\infty$
while $|Z_{UB}^{t,i} - Z_{LB}^{t,i}| \geq \epsilon$ **do**
  Solve relaxed master problem RMP$_i^t$;
  if RMP$_i^t$ is infeasible **then**
    if t==1 **then**
      return false: The problem instance is infeasible;
    else if t==H **then**
      return Dual extreme ray $\overline{r}_H^i = (\overline{\pi}_H^i, \overline{\lambda}_H^i, \overline{\mu}_H^i)$;
    else
      return Dual extreme ray $\overline{r}_t^i = (\overline{\pi}_t^i, \overline{\rho}_t^i, \overline{\sigma}_t^i, \overline{\lambda}_t^i, \overline{\mu}_t^i)$;
    end if
  else
    Let $(\overline{x}_{\exists}^{t,i}, \overline{\theta}^{t,i})$ be the current proposal and $\overline{z}_{\mathcal{Q}}^{t,i}$ the objective function value;
    Update lower bound: $Z_{LB}^{t,i} = \overline{z}_{\mathcal{Q}}^{t,i}$;
    # Solve all successor subproblems of node v_i^t
    for $v_j^{t+1} \in d(i,t)$ **do**
      if $\mathcal{Q}^{t+1,j}(\overline{x}_{\exists}^{t,i}, x_{\forall}^{t,j})$ is infeasible **then**
        Let $\overline{r}_{t+1}^j = (\overline{\pi}_{t+1}^j, \overline{\rho}_{t+1}^j, \overline{\sigma}_{t+1}^j, \overline{\lambda}_{t+1}^j, \overline{\mu}_{t+1}^j)$ be the corresponding dual extreme ray;
        add the feasibility cut $D_m^{t,i} x_{\exists}^{t,i} \geq d_m^{t,i}$ to RMP$_i^t$;
      end if
      if $\mathcal{Q}^{t+1,j}(\overline{x}_{\exists}^{t,i}, x_{\forall}^{t,j})$ has an optimal solution **then**
        Let $\overline{p}_{t+1}^j = (\overline{\pi}_{t+1}^j, \overline{\rho}_{t+1}^j, \overline{\sigma}_{t+1}^j, \overline{\lambda}_{t+1}^j, \overline{\mu}_{t+1}^j)$ be the corresponding dual extreme point;
        Let $\overline{z}_{\mathcal{Q}}^{t+1,j}$ the be the corresponding objective function value;
        if $\overline{z}_{\mathcal{Q}}^{t+1,j} > \overline{\theta}^{t,i}$ **then**
          add the optimality cut $E_l^{t,i} x_{\exists}^{t,i} + \theta^{t,i} \geq e_l^{t,i}$ to RMP$_i^t$;
        end if
      end if
    end for
  end if
  if $\mathcal{Q}^{t+1,j}(\overline{x}_{\exists}^{t,i}, x_{\forall}^{t,j})$ has an optimal solution for all $j \in d(i,t)$ **then**
    Update upper bound: $Z_{UB}^{t,i} = (c_{\forall}^{t-1})^T x_{\forall}^{t-1,i} + (c_{\exists}^t)^T \overline{x}_{\exists}^{t,i} + \max_{v_j^{t+1} \in d(i,t)} \{\overline{z}_{\mathcal{Q}}^{t+1,j}\}$;
  end if
end while
# Optimal solution found at node v_i^t for $\mathcal{Q}^{t,i}(x_{\exists}^{t-1,a(i,t)}, x_{\forall}^{t-1,i})$
if t==1 **then**
  return true: Objective function value $\overline{z}_{\mathcal{Q}}^{1,1}$ and first-stage variable assignment $\overline{x}_{\exists}^{1,1}$;
else if t==H **then**
  return Objective function value $\overline{z}_{\mathcal{Q}}^{H,i}$ and extreme point $\overline{p}_H^i = (\overline{\pi}_H^i, \overline{\lambda}_H^i, \overline{\mu}_H^i)$;
else
  return Objective function value $\overline{z}_{\mathcal{Q}}^{t,i}$ and extreme point $\overline{p}_t^i = (\overline{\pi}_t^i, \overline{\rho}_t^i, \overline{\sigma}_t^i, \overline{\lambda}_t^i, \overline{\mu}_t^i)$;
end if

not fall below an predefined ε as depicted in line 2, the next iteration proceeds in line 4, where the relaxed master problem RMP_i^t is solved again to obtain a new proposal. When the bounds match, the current node v_i^t is solved to optimality and dual information as well as the optimal objective function value are passed back to the ancestor $a(i,t)$ of the current node.

Algorithm 9, called NBD-algorithm in the following, has been implemented and tested in a detailed computational study with instances that were generated from existing LP and IP test sets [88]. Note that in this description of the algorithm the tree is traversed with a *depth-first-search (DFS)* and each node of the tree is solved to optimality, or until it becomes infeasible, before dual information are passed up the tree. For more details on the implementation and the different algorithmic degrees of freedom the user can choose in our implementation, see Section 5.3.

5.2.3. Enhancemant of the Nested Benders Decomposition

In the following we describe how the NBD-algorithm can be extended by game-tree search techniques and a technique very similar to the bounding operation in modern branch-and-bound algorithms. As already mentioned in the previous section, solving a QLP instance with the NBD-algorithm can be illustrated as solving the universal player decision-tree, where each node v_i^t in the tree has a corresponding linear program RMP_i^t attached, whose solution determines the move of the existential player in the current situation. Using Algorithm 9 to solve a QLP instance, the tree is traversed multiple times with information in the form of proposals and cuts being passed between nodes of the tree until each node is solved to global optimality with respect to the *current* proposal of its ancestor. This is very similar as in the case of the Nested L-shaped method, even though the optimality of each node in the tree is not necessary for QLPs as we will argue in the following.

When solving a MSSLP with the Nested L-shaped method the goal is to optimize an *expected value* for a set of scenarios, whereas in a multi-stage QLP we optimize the *worst-case value* that can result from a set of scenarios. Thus, the Nested L-shaped method must solve the entire scenario-tree to optimality, and then the expected value at the root node results from the weighted sum of the optimal solutions of all nodal LPs in the scenario tree. The universal player decision-tree in contrast has the so-called minimax property, which means that we look for the worst-case path of the best winning strategy for the existential player. Nevertheless, Algorithm 9 solves each node to optimality, although most of these solutions do not influence the final worst-case result. This is very similar to the situation when a minimax-tree is solved with the Alpha-Beta Algorithm (as described in Section 4.1.2), and indeed, there is a technique very similar to the $\alpha\beta$-heuristic that can be applied within the NBD-algorithm in order to reduce the number of nodes that must be solved while Algorithm 9 traverses the universal player decision-tree. Note that while in the Alpha-Beta Algorithm each node is visited at most once, the NBD-algorithm traverses the universal player decision-tree multiple times.

The $\alpha\beta$-heuristic

The main difference between computing a weighted sum and the maximum of the objective function values of all successors of a node comes algorithmically into play when a proposal is determined to be feasible for all successors and the corresponding upper bound is computed. When solving a MSSLP with the Nested L-shaped method, all nodal LPs of the successors of a node must be solved to optimality to ensure termination of the algorithm. However, in the case of a multi-stage QLP, the optimality of all subproblems is not a mandatory requirement as we will see in the following.

If an inner node v_i^t receives a new proposal $\overline{x}_\exists^{t-1,a(i,t)}$ from its ancestor $a(i,t)$ at stage $t-1$, the subtree rooted at node v_i^t is solved to optimality with respect to this proposal, or until the nodal linear program attached to v_i^t becomes infeasible. After the feasibility of the subtree is established, the upper and lower bounds of v_i^t move towards each other and for the optimal objective function value $\overline{z}_\mathcal{Q}^{t,i}$ holds $Z_{LB}^{t,i} \leq \overline{z}_\mathcal{Q}^{t,i} \leq Z_{UB}^{t,i}$ until these values coincide eventually after a finite number of iterations. Then node v_i^t passes its optimal solution $\overline{z}_\mathcal{Q}^{t,i}$ and the corresponding dual information to its ancestor $a(i,t)$ at stage $t-1$.

When a node v_i^t determines that its current proposal $\overline{x}_\exists^{t,i}, \overline{\theta}^{t,i}$ is valid for all of its descendants $v_j^{t+1} \in d(i,t)$ their corresponding optimal objective function values $\overline{z}_\mathcal{Q}^{t+1,j}$ can be arranged as exemplarily denoted in the following for an example with 8 subproblems

$$\underbrace{\overline{z}_\mathcal{Q}^{t+1,2} \leq \overline{z}_\mathcal{Q}^{t+1,4} \leq \overline{z}_\mathcal{Q}^{t+1,6}}_{\text{no new optimality cuts}} \leq \overline{\theta}^{t,i} \underbrace{\leq \overline{z}_\mathcal{Q}^{t+1,7} \leq \overline{z}_\mathcal{Q}^{t+1,8} \leq \overline{z}_\mathcal{Q}^{t+1,3} \leq \overline{z}_\mathcal{Q}^{t+1,1} \leq \overbrace{\overline{z}_\mathcal{Q}^{t+1,5}}^{\text{worst-case}}}_{\text{new optimality cuts}}, \quad (5.39)$$

where some optimal solutions lie below the approximation variable $\overline{\theta}^{t,i}$ and thus do not yield new optimality cuts. Some subproblems yield larger values than $\overline{\theta}^{t,i}$ and are used to compute optimality cuts for the nodal LP attached to node v_i^t . One ore more subproblems yield the current worst-case that is used to compute the upper bound

$$Z_{UB}^{t,i} = (c_\exists^t)^T \overline{x}_\exists^{t,i} + \max\{\overline{z}_\mathcal{Q}^{t+1,1}, \overline{z}_\mathcal{Q}^{t+1,2}, \ldots, \overline{z}_\mathcal{Q}^{t+1,8}\}. \quad (5.40)$$

This is equal to the minimax principle as mentioned in Section 4.1.2. The existential player tries to minimize the value of a nodal linear program, with respect to the worst-case move of the universal player, which is the corresponding subproblem with the *maximal* objective function value. When using a depth first-search to traverse the tree as it is done in the Alpha-Beta Algorithm (see Section 4.1.2), the knowledge of the current maximal value of some $v_j^{t+1} \in J' \subset J$, with $J = d(i,t)$, can be used in a similar way as α is used in the Alpha-Beta Algorithm. In the current iteration, it denotes the minimum value, the maximizing player (the universal one) will at least obtain at node v_i^t. Let therefore $\alpha_i^t = (c_\forall^t)^T \overline{x}_\forall^{t,j} + \overline{z}_\mathcal{Q}^{t+1,j}$ denote the current maximal value after *some* of v_i^t's successors have already been solved. When α_i^t is passed to the remaining nodes from the set of successors $v_j^{t+1} \in J \backslash J'$, which are yet unsolved in the current iteration, each node v_j^{t+1} from this set can stop computing its exact optimal objective function value $\overline{z}_\mathcal{Q}^{t+1,j}$ after it

determines feasibility with respect to the current proposal $\overline{x}_{\exists}^{t,i}$, and detects that

$$Z_{LB}^{t+1,j} \leq \overline{z}_{\mathcal{Q}}^{t+1,j} \leq Z_{UB}^{t+1,j} \leq \alpha_i^t. \tag{5.41}$$

In this case the eventual optimal objective function value $\overline{z}_{\mathcal{Q}}^{t+1,j}$ will not affect the computation of the upper at the ancestor node v_i^t of node v_j^{t+1} . Depending on the size of the subtree rooted at node v_j^{t+1}, this can avoid the need to compute a very large amount of LPs.

We can also integrate a value analogously to β, which depicts the maximum value the minimizing player will yield for sure at a specific node v_i^t at stage t. In terms of the NBD-algorithm this is the upper bound $Z_{UB}^{t,i}$ from the previous iteration. In the next iteration, the value

$$\beta_i^t = Z_{UB}^{t,i} - (c_{\forall}^{t-1})^T x_{\forall}^{t-1,i} - (c_{\exists}^t)^T \overline{x}_{\exists}^{t,i} \tag{5.42}$$

results from the upper bound obtained in the last iteration less the current objective function part of node v_i^t and the static offset that results from the universal variable objective coefficients of the t-th stage. β_i^t can be passed to the successors $v_j^{t+1} \in J$ at stage $t+1$ together with the new proposal $\overline{x}_{\exists}^{t,i}$. If a successor v_j^{t+1} determines feasibility with respect to the current proposal and furthermore detects that

$$\beta_i^t \leq Z_{LB}^{t+1,j} \leq \overline{z}_{\mathcal{Q}}^{t+1,j} \leq Z_{UB}^{t+1,j}, \tag{5.43}$$

it can stop computing its exact optimal objective function value because a better solution has already been found in the previous iteration. In the following we denote these two values as α and β and if a node stops its computation because one of the above mentioned breaking conditions are met, we speak of an α-*cutoff* and a β-*cutoff* respectively, reflecting the strong similarities to the original Alpha-Beta Algorithm.

Stronger cuts by QLP-relaxations

Up to now, if a proposal $\overline{x}_{\exists}^{t,i}$ and an α_i^t or β_i^t is passed from a node v_i^t to one of its direct successors $v_j^{t+1} \in d(i,t)$, the entire subtree rooted at v_j^{t+1} must at least be traversed once to determine feasibility and thus to obtain a valid upper bound $Z_{UB}^{t+1,j}$, which then might lead to an α-cutoff. However, depending on the size of the subtree this can become a computationally expensive operation. In a modern branch-and-bound implementation the effectiveness of partial enumeration is closely related to the bounding operation, where the LP-relaxation of an IP or a MIP is solved to generate a lower bound (in case of a minimization problem) that can be used to prune the search space, and which is one of the most crucial operations in IP and MIP solving.

In Section 3.3 we introduced several types of QLP-relaxations and in order to strengthen the effect of the $\alpha\beta$-heuristic, the $\exists\forall$-relaxation is of particular interest. As we already know, if the $\exists\forall$-relaxation QLP$_{\exists\forall}$ of a QLP instance is feasible with $\overline{z}_{\exists\forall}$ being the corresponding optimal objective function value, then also the original QLP instance is feasible and for its optimal objective function value $\overline{z}$ holds $\overline{z} \leq \overline{z}_{\exists\forall}$.

Thus, if a proposal $\overline{x}_{\exists}^{t,i}$ and an α_i^t are passed from a node v_i^t to one of its direct successors

$v_j^{t+1} \in d(i,t)$, we may not need to solve the entire subtree rooted at v_j^{t+1}, but it might suffice to solve the $\exists\forall$-relaxation $\text{QLP}_{\exists\forall}^{t+1,j}$ of the $\text{QLP}^{t+1,j}$ that corresponds to the subtree rooted at node v_j^{t+1} and which results from the original QLP when all variables are fixed according to the current moves of both players up to node v_j^{t+1}. Thus, if we solve $\text{QLP}_{\exists\forall}^{t+1,j}$ and this is feasible with optimal objective function value $\overline{z}_{\exists\forall}^{t+1,j}$, then also the subtree rooted at node v_j^{t+1} is feasible with respect to the current proposal and for its optimal objective function value holds

$$Z_{LB}^{t+1,j} \leq \overline{z}_{\mathcal{Q}}^{t+1,j} \leq \overline{z}_{\exists\forall}^{t+1,j}. \tag{5.44}$$

Thus $\overline{z}_{\exists\forall}^{t+1,j}$ is an upper bound for the optimal objective function value of node v_j^{t+1} with respect to the current proposal $\overline{x}_{\exists}^{t,i}$ and it can be used to check if an α-cutoff occurred. However, if $\text{QLP}_{\exists\forall}^{t+1,j}$ is infeasible, we might have to check the entire subtree in order to obtain a valid upper bound. Note that it can of course happen that a subtree that need not be inspected in the current iteration, must be inspected in later iterations and vice versa. Furthermore note that while the $\alpha\beta$-heuristic makes only sense if applied when solving a multi-stage QLP optimization problem, the cut obtained by the solution of the $\exists\forall$-relaxation can be also used when a QLP feasibility problem is solved.

Move-Ordering

In the classical Alpha-Beta Algorithm as shown in Section 4.1.2, the $\alpha\beta$-effect can be furthermore improved with the help of a heuristic that determines the order in which the nodes of the tree are visited. As in the case of the Alpha-Beta Algorithm, this becomes also an important issue in the NBD-algorithm when the $\alpha\beta$-heuristic and QLP-relaxations are used in order to prune some subtrees during a tree traversal. The Alpha-Beta Algorithm, which traverses the minimax-tree only once in order to compute the solution at the root, uses heuristics to determine the sequence of possible moves. In the NBD-algorithm, which traverses the universal player decision-tree multiple times, the order in which nodes are visited can be organized based on information from previous iterations.

Until a proposal is not feasible for all successors of a node, infeasible successors from the previous iteration are visited first in the current version of our algorithm. Then, to obtain strong bounds as soon as possible after feasibility has been detected, the successor of a node that provided the worst-case subsolution in the previous iteration, is visited first in the next iteration, speculating that it will again provide a strong α-bound. Also the other successors are arranged in descending order by their solution values from the last iteration. However, many other sorting criterions are possible, e.g. the number of non-redundant optimality cuts so far or the total number a node was visited so far.

Algorithmic Description of the Alpha-Beta Nested Benders Decomposition

Function $\mathcal{Q}^{t,i}(x_{\exists}^{t-1,a(i,t)}, x_{\forall}^{t-1,i}, \alpha^{t-1,a(i,t)}, \beta^{t-1,a(i,t)})$ as described in Algorithm 10 illustrates the recursive application of the $\alpha\beta$-NBD algorithm to solve multi-stage QLP optimization problems.

Algorithm 10 Function $\mathcal{Q}^{t,i}(x_{\exists}^{t-1,a(i,t)}, x_{\forall}^{t-1,i}, \alpha^{t-1,a(i,t)}, \beta^{t-1,a(i,t)})$:

$Z_{LB}^{t,i} = -\infty$, $Z_{UB}^{t,i} = +\infty$, $\alpha_i^t = -\infty$, $\beta_i^t = +\infty$;
if $1 < t < T$ **then**
  Solve the $\exists\forall$-relaxation $\text{QLP}_{\exists\forall}^{t,i}$ of $\text{QLP}^{t,i}$
  if $\text{QLP}_{\exists\forall}^{t+1,j}$ is feasible and $\overline{z}_{\exists\forall}^{t,i}$ being the objective function value **then**
    if $\overline{z}_{\exists\forall}^{t,i} \leq \alpha^{t-1,a(i,t)}$ **then**
      break: α-cutoff
    end if
  end if
end if
while $|Z_{UB}^{t,i} - Z_{LB}^{t,i}| \geq \epsilon$ **do**
  Update move-sequence for next iteration and solve relaxed master problem RMP_i^t;
  if RMP_i^t is infeasible **then**
    if t==1 **then**
      return false: The problem instance is infeasible;
    end if
    return Dual extreme ray $\overline{r}_t^i$;
  else
    Let $(\overline{x}_{\exists}^{t,i}, \overline{\theta}^{t,i})$ be the current proposal and $\overline{z}_{\mathcal{Q}}^{t,i}$ the objective function value;
    Update lower bound: $Z_{LB}^{t,i} = \overline{z}_{\mathcal{Q}}^{t,i}$;
    if $Z_{LB}^{t,i} \geq \beta^{t-1,a(i,t)}$ **then**
      break: β-cutoff
    end if
    Update β_i^t: $\beta_i^t = Z_{UB}^{t,i} - \overline{x}_{\exists}^{t,i}$ and solve subproblems in move order;
    for $j \in d(i,t)$ **do**
      if $\mathcal{Q}^{t+1,j}(\overline{x}_{\exists}^{t,i}, x_{\forall}^{t,j}, \alpha_i^t, \beta_i^t)$ is infeasible **then**
        Add the feasibility cut $D_m^{t,i} x_{\exists}^{t,i} \geq d_m^{t,i}$ to RMP_i^t;
      end if
      if $\mathcal{Q}^{t+1,j}(\overline{x}_{\exists}^{t,i}, x_{\forall}^{t,j}, \alpha_i^t, \beta_i^t)$ has an optimal solution with $\overline{z}_{\mathcal{Q}}^{t+1,j}$ **then**
        if $\overline{z}_{\mathcal{Q}}^{t+1,j} > \overline{\theta}^{t,i}$ **then**
          Add the optimality cut $E_l^{t,i} x_{\exists}^{t,i} + \theta^{t,i} \geq e_l^{t,i}$ to RMP_i^t;
        end if
        if $\overline{z}_{\mathcal{Q}}^{t+1,j} > \alpha^{t,i}$ **then**
          Update $\alpha^{t,i}$: $\alpha^{t,i} = \overline{z}_{\mathcal{Q}}^{t+1,j}$;
        end if
      end if
    end for
  end if
  if $\mathcal{Q}^{t+1,j}(\overline{x}_{\exists}^{t,i}, x_{\forall}^{t,j}, \alpha_i^t, \beta_i^t)$ has an optimal solution for all $j \in d(i,t)$ **then**
    Update upper bound: $Z_{UB}^{t,i} = (c_{\exists}^t)^T \overline{x}_{\exists}^{t,i} + \max_{v_j^{t+1} \in d(i,t)} (\overline{z}_{\mathcal{Q}}^{t+1,j})$;
    if $Z_{UB}^{t,i} \leq \alpha^{t-1,a(i,t)}$ **then**
      break: α-cutoff
    end if
  end if
end while
if t==1 **then**
  return true: Objective function value $\overline{z}_{\mathcal{Q}}^{1,1}$ and first-stage variable assignment $\overline{x}_{\exists}^{1,1}$;
end if
return Upper bound $Z_{UB}^{t,i}$ and extreme point $\overline{p}_t^i$ from the dual solution;

Note that apart from algorithmic differences to Algorithm 9 some simplifications were made in order to preserve the readability, we e.g. assume that the vectors with the dual information have suitable dimension. In contrast to the NBD-algorithm the proposed extension has the following main differences.

Before the RMP$_i^t$ at an inner node v_i^t is solved, the algorithm first solves the $\exists\forall$-relaxation in order to check whether the subtree rooted at node v_i^t can be pruned as described in lines 2 – 9. This can of course only be done when the α value passed down from the ancestor $a(i,t)$ is different from its initial value $-\infty$, otherwise a valid α value does not yet exists and this check is bypassed. The same holds if the β value passed down from the ancestor equals its initial value $+\infty$.

If no cutoff occurs and the subtree must be solved, the algorithm continues as in the traditional case. However, before the subproblems are solved, the algorithm computes the corresponding value β_i^t if an upper bound does exist in line 23 and after each solution of a subproblem the current worst-case value $\alpha^{t,i}$ is potentially updated in line 33. Both values are passed to the successors of node v_i^t. After all subproblems are solved to feasibility the upper bound is updated and the algorithms checks for an α-cutoff (if the ancestor passed down a valid α value). If this is not the case the algorithm proceeds with the next iteration at line 11. After each solution of the RMP$_i^t$ the algorithm furthermore checks whether a β-cutoff occurs in line 20 (if the ancestor passed down a valid α value). When the algorithm decides to pass back dual information to its ancestor because a cutoff occurs or the bounds match, the upper bound is passed back instead of the objective function value $\overline{z}_{\mathcal{Q}}^{t,i}$ that results from the solution of the current RMP$_i^t$. The reason is that until the bounds match, $\overline{z}_{\mathcal{Q}}^{t,i}$ is a lower bound for the optimal objective function value at node v_i^t and if this value is passed back the algorithm would underestimate the real value, which can lead to problems if more complex tree traversal strategies and cutting schemes are used as described in the following section. Overestimating this value by passing back the upper bound causes no harm as long as the optimal objective function value of the worst-case descendant is guaranteed to be computed exactly.

The integration of $\alpha\beta$-cuts and the information obtained from the QLP-relaxation with the NBD-method can be interpreted as a combination of a branch-and-cut and branch-and-bound approach with the Alpha-Beta Algorithm. Benders optimality and feasibility cuts are used to prune the search space of existential variables, while the solution of the $\exists\forall$-QLP and the $\alpha\beta$-cuts are used to prune parts of the universal branching-tree in each iteration. In the following section we describe details of our algorithmic implementations and other techniques that were used to speedup the algorithm.

5.3. Implementation Details

Any implementation of the Nested Benders Decomposition approach and its variants has to deal with a number of details and it is well-known that often minor implementation issues can have a large impact on the practical performance of this method. In the context of stochastic programming many extensions of the standard nested L-shaped method have been proposed to improve its performance and we adapted some of the techniques to our

algorithm. In this section we sketch the most important design decisions of our implementation and the different choices our algorithm provides for the user. However, the focus of this section lies on techniques that find direct application (with some modifications) in the current version of our algorithm or which might be interesting extension for future work, not to give an overall overview of all work done in the field of stochastic programming in this context. A detailed experimental study of the effects and a comparison of their strength and weaknesses is provided in Section 6.2.

5.3.1. Information Passing

As already mentioned, solving a QLP with the $\alpha\beta$-NBD algorithm can be interpreted as solving the universal player decision tree, which is therefore traversed multiple times and information are passed among nodes of the tree. Starting from the root problem at stage $t = 1$, all subproblems in the tree are examined in sequence and solved either to optimality or until they have been detected to be infeasible. Information are passed along the tree in two directions: primal information (proposals for the existential variables) are passed down the tree used to create the right-hand sides in the subproblems, dual information from subproblems are passed back up the tree in order to generate feasibility and optimality cuts.

At each node of the tree an auxiliary LP that represents the current scenario must be solved. Before it can be solved, it must be loaded into a LP-solver and because re-optimization is usually much faster than solving from scratch, having one LP-solver instance loaded into the memory for each node of the tree might significantly lower the computational time required. However, due to memory constraints, this is not possible for problems with a large number of scenarios. On the other hand, if only one LP-solver is shared among all nodes of the tree, the memory requirements are negligible, but the corresponding nodal LP must be reloaded into the LP-solver each time a node is solved. The time might be reduced by reusing components already loaded, e.g. from a neighboring problem and it might be beneficial to use this data instead of recreating it from scratch. However, the anticipated benefit largely depends on how similar those problems are. When a nodal LPs that correspond to different stages must be solved in sequential iterations, then the entire problem must be created from scratch, since LPs from different stages do not share any data that can be reused. Another complicating factor is the presence of scenario-dependent optimality and feasibility cuts after some iterations at inner nodes of the tree. These need to be restored when a problem is reloaded into the LP-solver in a later iteration when the node is visited again. A compromise approach is to hold one LP-solver instance in memory for each stage $t = 1, \ldots, H$. If a nodal LPs at stage $t < H$ is loaded, it is adapted from data of the neighboring problem being loaded before. Since LPs at the same stage only differ in their right-hand sides and benders cuts, much of the data currently loaded can be reused. At stage $t = H$ only the right-hand sides need to be adapted in the corresponding LP-solver instance. When an inner node at stage t is loaded, cuts that were generated in the previous iteration and that exclusively correspond to the nodal LP being currently loaded into the LP-solver, must be stored (e.g. in the corresponding tree node). Afterwards the cuts of the current nodal LP that were generated in previous iterations are added to the LP-solver instance, followed by an adaption of the right-hand side to

the current proposal.

According to the results in [5], the decision which approach to use depends on the strategy used to traverse the tree. In the classical Nested L-shaped algorithm the scenario-tree is traversed using a *breadth-first-search* (BFS), one stage at a time. A *depth-first-search* (DFS) approach was proposed in [5], where it is argued that this allows memory saving techniques that are otherwise not possible. Both variants are depicted in Figure 5.2 and in the absence of feasibility cuts (e.g. when considering complete recourse problems) both variants are equivalent when we only consider which nodes are solved and in which relative order, but they differ when an infeasible subproblem is detected [5]. The classical version results in an immediate backward pass, whereas the recursive DFS approach will continue down the tree into subtrees which are not descendants of the infeasible node.

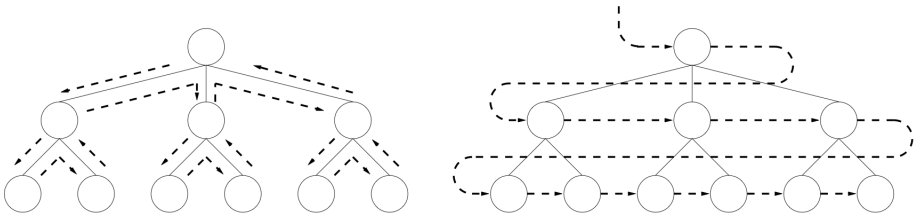


Figure 5.2.: Depth-First-Search vs. Breath-First-Search [5].

According to [5], BFS is preferred when one single LP-solver instance is used, which is re-created from scratch when passing between neighboring stages of the tree and which is adapted from data of neighboring LPs already loaded into memory. When a DFS is used, it is highly preferable to hold one LP per stage, since each feasible inner node is solved twice each time it is loaded. One time when the current proposal for the successors is generated and another time when dual information to pass to its ancestors are generated after the cuts from the subproblems were added. Using DFS with one single LP would cause a recreation from scratch every time a nodal LP is solved because consecutive solutions always stem from neighbored stages. In [5] a variant is proposed where LP-solver instances are only shared in a portion of the tree, whereas in other parts up to a so-called *block-stage*, each node has its own LP-solver instance. Below the block-stage subtrees are partitioned into several blocks, and all nodes within an block share the same LP-solver instance.

We decided to apply a DFS approach and share one LP-solver instances for all nodes at the same stage of the tree, since these problems only differ in their right-hand sides, apart from cuts being generated during the solution process. This can be illustrated as holding one LP-solver instances for each node of the current active path from the root to a leaf in memory. This saves a significant amount of memory and has the advantage that nodes at a stage can benefit from sharing information among each other. Moreover, DFS is the most successful approach to traverse a game-tree in two-person zero sum games like chess, othello etc. Furthermore, since the DFS approach reaches leafs of the universal player decision-tree much earlier than the classical variant and these leaves are lower bounds for the entire QLP and also for neighboring subtrees, these bounds can be used to integrate

pruning-techniques also used in the Alpha-Beta Algorithm as described in Section 4.1.2.

5.3.2. Sequencing Protocols

There is one LP problem attached to each node of the universal player decision tree and starting from the root at stage $t = 1$, all problems in the decision tree are examined in sequence using a depth-first-search and solved either to optimality or until they have been detected to be infeasible. Information is passed along the decision tree in two ways: dual information are passed back up the tree to generate new benders cuts, primal information are passed down the tree to generate new right-hand sides for the problems.

In Algorithm 9 a node in the tree only receives optimality cuts from its descendants after each descendant's subproblem is solved to optimality with respect to its descendants in turn, and so on. Optimality means that the upper and lower bounds on the optimal worst-case objective function value coincide. The *tree traversing theorem* [158] states that an optimality cut that is passed back from a subproblem while there is still a gap between the lower and upper bound is also a valid cut.

Thus, there is a great degree of freedom with respect to the order and direction in which subproblems in the decision tree are solved. At every stage t except the first ($t = 1$) and the last ($t = H$), the NBD-algorithm can decide in which direction it should push information after a nodal LP has been detected to be feasible. This is done with the help of a so-called *sequencing protocol* and several studies showed that the sequencing protocol has a strong impact on the solution time of the NBD-algorithm [97, 46, 158, 5, 232].

Very early in the development of variants of the Nested Benders Decomposition algorithm three basic tree traversing strategies – *fast-back* (FB), *fast-forward* (FF), and *fast-forward-fast-back* (FFFB) – were proposed in [231]. The fast-forward protocol is the variant mentioned above, where all subproblems are solved to optimality before passing back dual information to the ancestor node. Fast-back in contrast generates dual information whenever possible and only passes primal information to the next stage when the dual information provide no new cuts for the ancestor node. The fast-forward-fast-back protocol alternates between these two protocols moving in one direction (forward or backward) as long as possible, and reversing direction only when no further progress can be made in the current direction. Several computational evaluations showed that the fast-forward-fast-back protocol is the best among them [97, 158].

Recently hybrid mechanisms were proposed and combined with other approaches [5, 232]. The so-called *bouncing strategy* was introduced in [5] and solves a problem up to a stage $t < H - 1$ using the FFFB protocol and then returns to the first stage. The stage t where the algorithm changes its direction is called *block-stage* and the rationale behind this approach is that the algorithm spends most of the time in larger stages. Iterations are subdivided into *major iterations* and *minor iterations*. Major iterations are normal iterations that go down up to stage H, minor iterations stop at the block stage. Minor iterations are repeated until there are no new cuts in the subtree from the root up to stage t, then a major iteration is performed once. The goal is to compute better proposals for later stages and thereby to reduce the number of iterations and thus the overall number of nodes that must be solved. However, it is not clear which stage is the best block-stage a priori, an

enhanced version proposed in [232], were the block-stage is dynamically determined.

In our implementation the three sequencing protocols FB, FF, and FFFB are governed by a user-defined switch. In the consideration that the FFFB protocol has shown to be very efficient in practice we propose a slight modification in order to enhance the effect of the $\alpha\beta$-heuristic and the bounding by the solution of the QLP-relaxation at nodal LPs. Both techniques rely on the existence of an upper bound for the subtree recently traversed. The traditional FFFB protocol also initiates a backward pass before all subproblems were at least solved to feasibility and thus an upper bound does not exist in early iterations. We therefore propose a combination of the FF protocol and the FFFB protocol that stays in forward-mode until all subproblems of a current node become feasible and an upper bound on the subtree can be computed, or the node becomes infeasible itself. This variant is called *fast-forward-fast-feasible-back* (FFFFB) protocol in the following.

5.3.3. Solving nodal LP-problems

To generate benders cuts, we need dual information from subproblems. While it is possible for both a primal linear program and its dual to be infeasible, this condition is rare in practice. The more common case is that an infeasible LP has an unbounded dual. Especially for the case of a QLP, where all variables are bounded by Definition 2.1, it is guaranteed that each dual subproblem is unbounded when the corresponding primal subproblem is infeasible.

If a linear program is unbounded, then the feasible region is unbounded in a direction that improves the objective function. The direction in question is an extreme ray anchored at an extreme point of the feasible region (as e.g. visualized in Figure 5.1). For the computation of feasibility cuts it is important to use this ray in case of an unbounded subproblem, instead of simply using the dual values which are sometimes passed back in some non-commercial solvers as we noticed during our algorithmic implementations (even though the explicit function to get a ray was called). This can lead to invalid feasibility cuts, e.g. if the corresponding anchor (extreme point) is passed back and if it is located at the zero point. The result would be a feasibility cut with an empty left-hand side.

The state-of-the-art commercial solvers CPLEX and Gurobi provide the functionality to pass back an extreme ray, e.g. in the C++ API there is the corresponding function call `getRay` and also in the Gurobi C++ API the corresponding functions are available. The availability of advanced functions is solver dependent and not every implementation supports them. However, instead of explicitly solving the *dual* subproblem with the *primal* simplex algorithm, we also can solve the *primal* subproblem with the *dual* simplex.

Nowadays, modern LP and MIP solvers provide the opportunity to solve LP problems either by an interior point, primal simplex or dual simplex algorithm or a combination of them. Several computational studies indicate that the overall performance of the dual simplex may be superior to that of the primal simplex algorithm [49]. Also when solving a MIP with a state-of-the-art MIP-solver based on a branch-and-bound approach, where dual bounds on the objective function value are computed by successive re-optimization of LP subproblems, the dual simplex method is typically used because of its ability to take advantage of a given nearly optimal starting solution from previous iterations, similar as it

is the case in the (Nested) Benders Decomposition approach.

When a primal subproblem is solved with the dual simplex method and it is feasible, the dual values can be directly accessed, e.g. in the CPLEX C++ API with a function named `getDuals`. When the LP subproblem is infeasible, it is not longer possible to use the `getRay` function to get the dual ray, instead we need a Farkas' certificate (see Remark 6), which is a recession direction (a vector of constraint multipliers) for the dual problem and proves the infeasibility of the primal LP. Most modern state-of-the-art solvers provide the functionality to access these data, e.g. using CPLEX by the call `getFarkas`.

If the variable bounds are implicitly added to the constraint system of the primal LP, in which case they will have dual variables associated with them, the approach mentioned above works out of the box. However, the more efficient way is to specifiy them directly as bounds of the model and let the LP-solver software handle them. This leads to an increase in performance, since the number of constraints and thus the size of the basis matrices is reduced. However, in this case there will be no associated dual variables, which raises the question how to get the corresponding λ and μ values (called dual slack variables) for the lower and upper bounds.

If the primal subproblem is feasible, the dual slack variables λ and μ correspond to the vector of *reduced costs* (rc) as e.g. described in [136]. For a given rc_i one can simply check whether it corresponds to the i-th entry of λ_i or μ_i by checking the actual variable bounds and the current primal solution as described in [136, 183]. If the primal subproblem is infeasible and primal variable bounds are present, the corresponding dual slack variables can not be determined via the reduced costs, as they are not available. In this case we can extend the Farkas' certificate by the corresponding dual multipliers to a dual ray as described in [184].

5.3.4. Cutting Mechanism

It it well-known that the cutting scheme used within the (Nested) Benders Decomposition approach is of particular importance for the overall practical performance and a diverse range of publications deal with this topic [25, 48, 46, 95, 216, 232].

A drawback that can especially occur in early iterations is that as long as there are violated feasibility cuts, no optimality cuts are computed using the traditional approach, although they are important to improve the lower bound because they involve the approximation variable(s) θ explicitly and should be generated as early as possible to that reason. A simple way to mitigate this drawback was proposed by Benders himself in [25], where he argued that in case of an dual unbounded subproblem, where the primal simplex method discovered an unbounded extreme ray, also an vertex of the dual solution space is known, which is the origin of the unbounded ray (see also Figure 5.1). Thus, the corresponding optimality cut can be computed with very little additional computational effort. However, even though the resulting optimality cut is not guaranteed to be violated, its addition can be quite substantial in many cases [95]. In our implementation this type of optimality cut can be enabled by the user.

In the context of stochastic programming many variants of the (nested) L-shaped mode were proposed that dealt with different cutting schemes for the optimality cuts [48, 46,

232]. When solving a MSSLP, there are two common ways to create optimality cuts in order to approximate the objective function values for deeper stages if there are more than one subproblem. One way is summing up the weighted dual variables for each subproblem forming one optimality cut, whereas the other way is to disaggregate optimality cuts by placing one cut for each subproblem in the corresponding master, called *multicut* [48]. The usual procedure places one cut at each iteration, aggregating all the information of a node's successors by summing up the weighted dual variables for each subproblem forming one optimality cut, which then restricts a single approximation variable θ. Birge and Louveaux suggest to disaggregate optimality cuts by placing one cut for each subproblem in the corresponding master [48]. This *multicut* approach replaces a single approximation variable θ by $\sum_{k=1}^{k} p_k \theta_k$ and then places cuts on each θ_k using only information from the corresponding k-th subproblem. Comparisons of the standard aggregated cut and the multicut approach are presented in [46], the experiments presented in [97] showed that multicuts may reduce the solution time for large problems. However, the trade-offs in terms of computational times are problem dependent.

An adaptive multicut method that generalizes the single cut and the multicut methods for the two-stage case is proposed in [216]. The proposed method dynamically adjusts the level of aggregation to somewhere between single cut and pure multicut, since a good level of cut aggregation is usually not known a priori. Their computational investigations showed that an dynamic adaption can lead to a significant improvement in the solution time. In [232] this technique was extended to the multi-stage case.

In each implementation of a variant of the Nested Benders Decomposition scheme, a difficulty that can arise is a large number of cuts that might need to be stored in the master problems. In the algorithmic description of the BD-method and the NBD-methods as presented in Section 5.2 in each iteration one feasibility cut for each infeasible subproblem is added, alternatively the algorithm could break the current iteration after the first appearance of a feasibility cut and directly solve the master problem again with the new cut to obtain an updated proposal for the next iteration. In our implementation both variants are available via a user defined switch. To reduce the number of optimality cuts when solving a TSSLP or a MSSLP one must balance the pros and cons between a single cut and a multicut approach or use a hybrid approach. When solving the a two-stage or multi-stage QLP with the (Nested) Benders Decomposition approach, an aggregated cut makes no sense, since the subproblems do not have probabilities associated with them and we are not interested in a weighted sum, but a worst-case value. Accordingly, using the term multicut means in this case that all optimality cuts, whose corresponding objective function values are greater than the approximation variable θ are added. The QLP equivalent of a single-cut mechanism is to only add the optimality cut that corresponds to the worst-case subsolution of the current iteration. Another reason on the account to only add a single optimality cut in each iteration is that optimality cuts are typically quite bad from a numerical point of view, because they tend to exhibit a higher density and dynamism (ratio between the maximum and minimum absolute value of the cut coefficients) with respect to feasibility cuts [95]. In our implementation we support both cases via a user defined switch and furthermore allow to choose whether optimality cuts are also added in the presence of other infeasible subproblems, or only when all subproblems are feasible.

Another approach to reduce the computational burden due to a large number of cuts that arise during computation is to remove previously added cuts when they become obsolete. Although it was shown that there is no easy way to keep the number of cuts bounded, it was pointed out that inactive cuts can be removed [189]. A problem that might occur is that there is no reliable rule to determine which cut can be safely removed [189], which can lead to an unnecessary re-computation in later iterations. Very recently a variant of the (nested) L-shaped method was proposed where the cut removal strategy does not only remove inactive cuts (when the corresponding dual values are zero) but keeps some of the removed information by computing an aggregated cut from the removed cuts [232]. However, since the removal of cuts only leads to a reduced memory usage and does not increase the solution time considerably as reported in [216] and furthermore the computation of an aggregated cut from the removed ones is not possible for QLPs, we did not implement such techniques in our implementation. However, before each new feasibility or optimality cut is added, we apply a simple check that detects if the cut is redundant or if it makes an existing cut redundant itself. In the former case the new redundant cut is discarded, in the latter case the existing redundant cut is adapted.

5.3.5. Warm-Start Techniques and Advanced Start

Most state-of-the art LP-solvers offer three algorithmic choices for the solution of the subproblems, the primal simplex algorithm, the dual simplex algorithm and an interior point method. During the course of the (Nested) Benders Decomposition algorithm many similar subproblems are solved repeatedly and techniques that take advantage of optimal basis information from previous solutions are critical for an efficient implementation. While the primary computational challenge in the two-stage case is the solution of a large number of similar subproblems that only differ in their right-hand sides in each iteration, the greatest potential for computational savings in the multi-stage case comes from the ability to select good initial start bases whenever a subproblem is solved.

We decided to use the dual simplex algorithm as LP-solver because it can be warm started from a previously stored basis and our algorithmic implementation benefits from these capabilities in the following way. Due to the iterative nature of the NBD-algorithm the subproblems are slightly changed in each iteration by the addition of new cuts or new right-hand side values that result from varying proposals. When a nodal LP is solved again after a cut has been placed in it, the previously optimal solution will still satisfy all but the last constraint and it will in particular still be *dual feasible*. It is therefore possible to use the basis from the previous iteration as the starting basis for the dual simplex method in the next iteration. The same holds if only the right-hand side changes.

In our implementation we usually store the optimal basis of a LP as well as the cuts that were generated up to that point in each node of the tree and reload them into the LP-solver when the node is visited the next time. For cases where we need to reduce memory requirements, our implementation allows a user defined switch to a mode were the optimal basis and/or the cuts are not stored. If the base from the previous solution of an LP is not reused when the LP is solved the next time, the underlying LP-solver might try to automatically perform some sort of warm-start by using solution information from the

solution of the LP solved most recently. If those problems only differ slightly, e.g. only in a few right-hand side coefficients, this might also work well. Note that if also the cuts are discarded, every subtree must be solved to optimality using the fast-forward sequencing protocol in each iteration to avoid cycling and thus to ensure convergence of the algorithm in this case.

One of the well-known drawbacks of the Nested Benders Decomposition algorithm is that it tends to be inefficient in early iterations of the algorithm [186]. To avoid iterations with slow progress at the beginning, we use an advanced start procedure to get a good start solution by solving the LP-relaxation of the QLP and use the assignment of the first-stage existential variables as initial proposal for the first iteration of our algorithm.

5.3.6. Other possible Enhancements

In the following we describe possible enhancements of the general Nested Benders Decomposition algorithm that originated from the stochastic programming community, where various extensions to the standard L-shaped and the nested L-shaped method were proposed in the last decades. Also very recently many new algorithmic ideas were proposed and evaluated in detailed computational studies [216, 239, 7, 232]. In the current version of our implementation the following techniques are not implemented since many of them can only be applied to stochastic problems, others might be implemented in later versions.

Bunching

Note that apart from benders cuts at the inner nodes, all linear programming problems only differ in their right-hand sides. When a large number of rather small, similar problems must be solved, it might be beneficial to try to reduce the computational time for each of these small problems, even though each individual problem only takes a small amount of time. In the absence of cuts, many problems at the same stage have the same constraint matrix and only differ in their right-hand sides, implying that many subproblems at the same stage might share the same bases, both at the optimum and at intermediate steps in the pivot sequence of the simplex method [119]. Sequential solution of similar subproblems can take advantage of this fact by considering all these problems simultaneously. The method, called *trickling* [97] or *bunching* [230], generates a tree of optimal bases for subproblems, which are checked before a subproblem is solved. In some cases this approach can achieve significant efficiencies. However, the current version of our implementation does not support this technique and instead we confine ourself to the warm-start capabilities capabilities of modern dual simplex solvers from a given basis as described above.

Stage Aggregation

When a problem instance consists of many stages with a large number of rather small problems, it might be beneficial to aggregate them, resulting in a tree with fewer nodes but larger nodal LPs, as e.g. described in [79]. Apart from an increased solution time for nodal LPs that could be compensated by the need to solve less subproblems and might

result in a lower overall solution time, aggregation also affects the memory consumed. The size of the subproblems becomes larger, however, the number of cuts, which are used to transfer information between stages are reduced. Stage aggregation might be an interesting enhancement of our implementation.

Regularization

It is well-known that the original (Nested) Benders Decomposition algorithm, even if a good warm-start is used, may deviate significantly from this start point in some cases, and thus, all efficiency obtained from a good warm-start might be lost [186]. Furthermore, as already mentioned above, the number of cuts for master problems may increase substantially, leading to very large nodal LPs. To overcome these difficulties, a regularized version of the L-shaped method was proposed in [186, 190], and extended to the nested L-shaped method in [191, 187]. This method adds a quadratic term to the objective function of each subproblem in order to keep solutions of successive iterations closer together in order to stabilize the progress of the algorithm. It furthermore enables valid deletion schemes for unused cuts. For the two-stage case a detailed computational study was proposed in [239]. There, the regularized L-shaped method was compared to the level decomposition method developed in [94], the box-constrained trust-region method [143], and the original L-shaped method. The authors report that the regularized version outperforms the non-regularized version in many instances of their test set. Regularization might be another interesting enhancement of our implementation, though, to the best of our knowledge, further computational experiences when being applied to multi-stage problems do not exist.

Parallelization

Finally, the NBD-algorithm is well suited for parallelization as all the subproblems that have to be solved in a stage are independent of one another and can thus be solved in parallel. Several approaches have been proposed in the research literature in the context of stochastic programming [191, 80, 46, 79] and also we plan to parallelize our algorithmic implementation.

6. Implementation and Numerical Results

As we have seen in the last chapter, there are many factors that may influence the performance of the (Nested) Benders Decomposition algorithm and a true evaluation can be only based on a thorough practical investigation, which is the subject of this chapter. As a working basis for the algorithm development many other components had to be implemented, e.g. an input format to describe QLPs and its integer variants such that users can read and write instances on hard disk, data structures to store the content of them, conversion methods to create and solve the corresponding DEPs for debugging purposes of our own algorithm, and so on. We decided to collect all components within a software package and make them available via an intuitive application programming interface (API).

In Section 6.1 we give an overview of the resulting quantified linear and integer programming framework and solver called `QlpOpt`, where we describe its main functionalities and components. In Section 6.2 we provide a thorough computational study where we compare the effect of different computational strategies as mentioned in the previous chapter, when being applied within our (Nested) Benders Decomposition algorithm. We furthermore compare our implementation with the approach to directly solve the DEPs of QLP instances using external state-of-the-art LP-solvers. Finally, we check the applicability of quantifier elimination techniques, which are usually used in the context of real-valued quantified constraints when being applied to QLPs.

6.1. The Quantified Linear Programming Framework `QlpOpt`

This thesis is supplemented by the software `QlpOpt`, which is a solver and framework for quantified linear programming problems. It provides different solution techniques and a variety of methods to work with quantified (mixed-integer) linear programs. It has been designed both as black box optimizer accessible via a simple *command-line interface* (CLI), and an *application programming interface* (API). The sources can be made available upon request for research purposes by the author.

6.1.1. Overview

All experiments were carried out using a software designed and implemented following the theoretical framework laid down in this thesis. The algorithms and components were implemented in C++ and embedded into a software package called `QlpOpt`. Apart from

several algorithms to solve QLPs and certain kinds of Q(M)IPs, `QlpOpt` provides methods to read and write instances on disk using a common file format (see Section 6.1.3). It allows the user to create certain kinds of QLP instances via an intuitive interface similar as is it used in conventional (mixed-)integer linear programming APIs like CPLEX or Gurobi (see Section 6.1.4). It furthermore offers various techniques to modify these instances by e.g. adding or removing variables, constraints, and matrix coefficients. To support interoperability, all platform dependent functions are implemented using *Boost (C++ libraries)*[1], which range from general-purpose libraries to operating system abstractions. Boost works on almost any modern operating system, including Apple OS X, Linux and Microsoft Windows variants. Our implementation furthermore supports exact arithmetic using the *GNU multiple precision arithmetic library* (GMP)[2], a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating-point numbers (see Section 6.1.5). In the following we provide some more details.

6.1.2. Components

The software `QlpOpt` is composed of the following main components:

- **Datastructures:** A QLP solver needs certain internal data structures to operate. For the implementation we rely on our own efficient data structures and initialization methods, e.g. sparse data structures like `QpSparseVec` or `QpSparseMatrix` using packed storage that only stores nonzero entries and their indices. The datastructures package also comprises own numerical datatypes `QpNum`, `QpDouble`, and `QpRational` that are used to avoid numerical difficulties (see also Section 6.1.5). These base types are used to implement datatypes for the single components of a QLP like `QpObjectiveFunction`, `QpVar`, `QpRhs`, and also a flexible datastructure `Qlp` that directly supports methods to dynamically modify variables, constraints and other parts of the QLP.

- **Algorithms:** This package is the core of our algorithmic implementation and contains some of the algorithms described in Section 4.1 and Section 5.2. We implemented several variants of the (Nested) Benders Decomposition algorithm, the QEA-method as described in Algorithm 3 and functions to solve the DEP of a QLP and certain Q(M)IP instances.

- **Extern Solvers:** This package contains several external LP and MIP solvers for the computation of the DEP of or to solve nodal LPs within the (Nested) Benders Decomposition algorithm (see also Section 6.1.4).

- **Utilities:** This package contains several components that are not necessarily required to solve QLPs, but that provide various methods that are needed within our solution framework. This contains components to manipulate QLPs, like classes to compute certain kinds of QLP and QIP relaxations and components to create the DEP of a

[1] `http://www.boost.org`
[2] `https://gmplib.org`

QLP or QIP instance. Further utility classes to write to and read from the filesystem, exception handling, and so on are also included.

6.1.3. The QLP File Format

As an input format for our quantified linear programming framework, a new standardized file format was required in order to handle quantifiers and their order in the quantification sequence. We decided to extend the CPLEX LP-file format[3] because it well-known and features also a very good readability for the non-experienced user compared to other formats. Some mandatory modifications between the CPLEX-LP file format and the QLP file format have to be considered and are listed below. The keywords are as follows, where new keywords are marked with $*$:

MAXIMIZE / MINIMIZE
SUBJECT TO
BOUNDS
GENERALS
BINARIES
ALL*
EXISTS*
RANDOM*
ORDER*
END

Every keyword has to be written in capital letters, abbreviations are not allowed. The `BOUNDS` section that follows the constraint section is mandatory. Each bound definition has to begin on a new line and must be of the form $l \leq x \leq u$. Afterwards the variables are typed. To specify any of the variables as general integer variable, a `GENERAL` section has to be added; to specify any of the variables as binary integer variable, a `BINARY` section has to be added. In every section the variables are separated by at least one empty space. Moreover, every variable is marked with one of the new keywords `ALL` or `EXISTS`, and analogously the variables in the these sections are separated by at least one empty space. The order of the variables is specified within the `ORDER` block where they must again be separated by at least one empty space.

[3] `http://lpsolve.sourceforge.net/5.5/lp-format.htm`

The QLP file for the example

$$\exists\, x_1 \in [0,1]\ \forall\, x_2 \in [0,1]\ \exists\, x_3 \in [0,1] \cap \mathbb{Z}:$$
$$\min\ x_1 - 2x_2 - 2x_3$$
$$\begin{pmatrix} 0 & -1 & -1 \\ -1 & 1 & 1 \\ 2 & 2 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} -1 \\ 1 \\ 3 \end{pmatrix},$$

looks as follows:

MINIMIZE
$x1 - 2x2 - 2x3$
SUBJECT TO
$-x2 - x3 <= -1$
$-x1 + x2 + x3 <= 1$
$2x1 + 2x2 <= 3$
BOUNDS
$0 <= x1 <= 1$
$0 <= x2 <= 1$
$0 <= x3 <= 1$
GENERALS
$x1\ x2\ x3$
EXISTS
$x1\ x3$
ALL
$x2$
ORDER
$x1\ x2\ x3$
END.

6.1.4. External LP and MIP Solvers

To solve nodal LPs in our algorithm and to solve the DEPs, we need access to linear and mixed-integer programming solvers and it is obvious that the choice of the LP software is crucial for the overall efficiency of our algorithmic implementation. Common LP and MIP solvers differ in many ways and they come with different licenses and of course different features, e.g., the way in which they can be called by a program as well as the algorithms

that are incorporated to solve (mixed-integer) linear problems. In our implementation we support several common solvers and make them available to our algorithms and users of our framework. This is done via a common interface, whose solver-specific implementation acts as a wrapper class around a specific solver library and provides access to the functionalities in a uniform way. Thus, it it easy to extend our implementation by further external LP or MIP solvers, which can be then used to solve the DEPs or nodal LPs of our (Nested) Benders Decomposition algorithm.

In the current version of our implementation the following commercial and free open source solvers are available:

- **CPLEX:** The CPLEX optimizer[4] is a commercial solver actively developed by IBM. Apart from a modern solver for (mixed-integer) linear programs it can also handle other related mathematical optimization problems (e.g. non-linear and quadratic programs). It is written in C and it is available on several computing platforms and can be accessed from several programming languages.

- **Gurobi:** The Gurobi optimizer[5] is a commercial solver actively developed by Gurobi Optimization. Apart from a modern solver for (mixed-integer) linear programs it can also handle other related mathematical optimization problems (e.g. non-linear and quadratic programs). It is written in C and it is available on several computing platforms and can be accessed from several programming languages.

- **SoPlex:** SoPlex[6] is a non-commercial linear programming solver free for academic research. It is based on the revised simplex algorithm implemented in C++ and was developed by R.Wunderling during his PhD thesis [233]. It can be used used as a standalone solver or it can be embedded into other programs using a C++ class library. We use SoPlex indirectly to solve DEPs of QLPs via the mixed-integer programming solver SCIP.

- **SCIP:** SCIP (Solving Constraint Integer Programs)[7] is a non-commercial framework for solving integer and constraint programs free for academic research. The SCIP core was developed by T. Achterberg during his PhD thesis [2, 4, 3] based on the solver SIP of A. Martin. Today SCIP is developed further by a community from the academic environment and it is currently one of the fastest non-commercial solvers for mixed integer programming (MIP) and mixed integer nonlinear programming (MINLP). The source code is publicly available and can be used via a C callable library or a standalone solver. Similar as in our implementation, SCIP has an open LP-solver support, which means that it is possible to call a range of open source and commercial LP-solvers, we use SCIP together with SoPlex.

- **CLP:** Clp (Coin-or linear programming)[8] is a non-commercial open-source linear

[4] http://www.ibm.com/software/commerce/optimization/cplex-optimizer
[5] http://www.gurobi.com
[6] http://soplex.zib.de
[7] http://scip.zib.de
[8] https://projects.coin-or.org/Clp

programming solver created within the COIN-OR project, which aims at creating open software for the operations research community. Clp is written in C++ and can be accessed via a callable library.

- **CBC:** Cbc (Coin-or branch and cut)[9] is is a non-commercial open-source mixed integer programming solver created within the COIN-OR project. Cbc is written in C++ and can be used as a callable library or using a stand-alone executable.

There are other codes available we did not support yet, e.g. the commercial software *XPRESS*[10], but also non-commercial codes like *GNU Linear Programming Kit* (GLPK)[11] and *LP SOLVE*[12]. However, it is beyond the scope of this thesis to list all different optimization solvers and their performance in detail and we refer the reader to one of the best sources for information about performance issues regarding different (mixed-integer) linear programming solvers provided by Hans Mittelmann [156]. In our implementation the solvers, CPLEX, Gurobi, SCIP, and CBC can be used to solve general DEPs, which can be an LP, IP, or MIP. For our (Nested) Benders Decomposition variants we support the solvers CPLEX, Gurobi, CLP, and an exact LP-solver called *TOSimplex* that was developed in our working group and is described more detailed in the following section. In Section 6.2 we compare the performance of the two commercial solvers CPLEX and Gurobi when being used to solve the DEPs of QLP instances.

6.1.5. Exact Arithmetic

A practical issue that arises during the implementation of linear and mixed-integer programming software is numerical stability. Especially for the implementation of the (Nested) Benders Decomposition algorithm, numerical stability is of particular importance and motivated by two aspects. First of all, due to the limited precision of floating point arithmetic used by most LP-solvers, errors are introduced throughout the computation, for example in the LU and Cholesky factorizations [138]. Furthermore, benders cuts are known to be typically quite bad from a numerical point of view, especially optimality cuts tend to be very dense and they often exhibit a high ratio between the maximum and minimum absolute value of the cut coefficients [95]. Also the underlying external LP-solvers suffer from these numerical difficulties and deal with them by incorporating tolerances. Arithmetic errors together with the use of tolerances make it possible that the results reported by inexact LP-solvers might not be optimal or even feasible. For a detailed explanation of possible errors see [111, 116].

In our implementation we try to avoid numerical difficulties in several ways. We implemented an own numerical datatype `QpNum` that supports all standard arithmetic operators as e.g. addition, multiplication or various comparisons operators, and which can be used like any standard build-in numerical C++ datatype. Internally it either wraps a `double` and overloads the arithmetic operators with predefined zero tolerances that can be changed

[9] `https://projects.coin-or.org/Cbc`
[10] `http://www.fico.com/en/products/fico-xpress-optimization-suite`
[11] `https://www.gnu.org/software/glpk`
[12] `http://sourceforge.net/projects/lpsolve`

by the user (e.g. to the tolerances used in the external LP-solver). Alternatively it wraps the exact rational datatype provided by the GMP library and in this case all arithmetic operations outside of the external LP-solver are computed using exact arithmetic. For the extern LP-solvers, the user can specify the feasibility and optimality tolerances, as well as other several parameters that might avoid numerical difficulties, like e.g. the *Markowitz tolerance*, which influences pivot selection during basis factorization and may improve the numerical properties of the solution.

However, even if these parameters are all adjusted to settings that should lead to a minimum amount of numerical difficulties, the underlying LP-solvers still use floating point arithmetic and hence the arithmetic incurs errors, which can lead to wrong results. Examples where popular commercial and public-domain fail to find optimal solutions are given in [83, 6]. An easy way to avoid these problems is to use an LP-solver that computes entirely using exact rational arithmetic, however, this is is much slower than floating point arithmetic and hence only small problems can be solved [6]. Instead of using exact arithmetic in each computational step, an optimal or a near-optimal basis computed by an inexact solver can be used as starting basis for an exact solver as proposed in [83].

Also our implementation provides the possibility to solve QLPs within our (Nested) Benders Decomposition algorithm using exact arithmetic, in cases when the user still observes wrong results that might result from numerical difficulties after the solver settings and tolerances were adapted. We therefore use an exact LP-solver called *TOSimplex*, which is based on the dual simplex as described in [137] and uses the GMP library for exact arithmetic internally. TOSimplex was developed in our working group and is also used in the software polymake [98], a tool to study the combinatorics and the geometry of convex polytopes and polyhedra. On the one hand, it can serve as a standalone solver and performs all intermediate computation steps itself using exact arithmetic, on the other hand it can be started from the optimal basis of an inexact LP-solver. In the latter case the solution passed to the solver is verified using exact arithmetic and recalculated if necessary. Apart from the exact solution of LPs and the standard methods to access solution information like primal and dual values or the reduced costs, TOSimplex supports advanced functions to get an exact Farkas' certificate of infeasibility and to verify an inexact Farkas' certificate of infeasibility reported by an inexact LP-solver.

When using exact arithmetic in our implementation all intermediate arithmetic operations are computed exact. At each step where an LP must be solved, one of the inexact extern LP-solvers presented in Section 6.1.4 is used to compute an inexact solution. This is passed to TOSimplex in form of the inexact optimal basis or the inexact Farkas' certificate. These are verified using exact arithmetic and TOSimplex then returns the exact solution in form of exact values for the primal and dual variables, or in form of an exact Farkas' certificate, which is then extended to a full dual ray as described in Section 5.3.3. Alternatively TOSimplex can also be used as standalone external LP-solver. These advanced functionalities, and the possibility to easily extend TOSimplex by other advanced function when needed, where the main reasons not to use one of the available exact solvers like QsOpt_ex [6]. Also the DEP of a QLP can be solved exact using TOSimplex solely as external LP-solver, alternatively TOSimplex can be used to verify the solution on an inexact LP-solver and recompute it if necessary.

6.2. Computational Environment Test Sets

In the following section we present a detailed computational study whose purpose are

- to validate that our algorithm actually solves QLP instances correct;
- to compare different computational strategies of our algorithm for the two-stage and the multi-stage case;
- to compare the performance of our algorithm to that of solving the DEP with standard LP software;
- and to compare the performance of our algorithm to that of quantifier elimination based techniques.

6.2.1. Experimental Environment and Software

All tests were run on a Intel(R) Xeon(R) CPU E5-1620 v2 with 3.70 GHz and four physical cores, 10 MB cache and 128 GB RAM running Ubuntu 14.04. The algorithmic framework was implemented in C++ and compiled using clang++ version 3.4. We use GMP version 5.1.2 and version 1.48.0 of the Boost C++ library. All instances are stored on hard disk using the QLP format described in Section 6.1.3. Times are measured in wall-clock times using the boost timer library. The external LP and MIP solvers were used and tested within our algorithmic framework with the following versions: CPLEX (12.6.0), Gurobi (5.6.3), SCIP (3.1.0), SoPlex (2.0.0), CBC (2.8.9), CLP (1.15.6). During our tests the computer had the exclusive task of solving the problems to reduce measurement errors by cpu load from other processes.

6.2.2. Test Instances

The experimental evaluation of algorithms for QLPs is difficult for two reasons. First, due to the young age of research in this area, there is a lack of benchmarks. Second, it is important to carefully choose the test set(s) in order to get meaningful results while completing the evaluation in a reasonable time. Since yet no real-world QLP instances do exist in the literature, we decided to use existing instances from well-known libraries in order to (randomly) generate QLP instances as described more detailed in the following.

The NETLIB LP Test Problem Set

The NETLIB LP Library [13] is a collection of linear programming examples from a variety of sources and in the years since its introduction in 1985 [99] it has served as a repository of linear programming (LP) instances for many researchers. It includes many real world applications and contains problems ranging from 32 variables up to 15695 variables and from 27 constraints up to 16675 constraints. While in the beginning some of the problems were considered large and difficult, they are small and easy for todays solvers. However, it

[13] http://www.netlib.org/lp/

is known that many of these problems are ill-conditioned and hence, numerical difficulties can occur [163, 132].

In the absence of real-world problems, the approach to generate QLP instances from this test-set seems obvious since many NETLIB instances were also used as basis in many computational studies in the context of Stochastic Programming [97, 80, 46]. Therefore they where were converted toTSSLP and MSSLPs. Also in the context of robust optimization NETLIB instances served as basis for many computational studies. In [23, 24] it was shown that even the feasibility properties of these instances can be severely affected by only small perturbations of the data. In [22] 90 instances from the NETLIB test set were used to apply robust optimization techniques as described in Section 2.2.2.

To convert LP instances to QLPs, we used the approach presented in [88], where blocks of universally quantified variables were added to the original LP, which can be interpreted as inserting some kinds of uncertainties that are controlled by an adversary, resulting in varying right-hand sides. For our tests we took LP instances with a maximum number of 1880 variables and 1090 constraints and generated QLPs with 4 – 18 universally quantified variables. The corresponding columns for each new variable $y_i \in \forall$ with $y_i \in [0, 1]$ each have a density of 5 – 8% of non-zero coefficients from the interval $[-1, 1]$. The coefficients were positioned randomly in the corresponding column. We furthermore varied the number of $\forall$-quantifier blocks to 1, 2, 3, 4, 5 and distributed them equally in the QLP. This results in two-stage and multi-stage QLP instances. The following problem list shows the NETLIB instances used to generate QLPs[14]: *ADLITTLE, AFIRO, AGG, AGG2, AGG3, ISRAEL, KB2, SC105, SC205, SC50A, SC50B, SCAGR25, SCAGR7, SCTAP1, SCTAP2.*

The following test-sets were created in the last years as basis for several publications and for debugging purposes:

- **ESA11:** The test-set contains 108 instances with 5 – 10 universal variables and 2 – 6 existential variable blocks. The instances have 32 – 480 variables and 27 – 471 constraints (see Appendix A.2.1 for more details of the test instances). This test-set was used in [88] to evaluate the performance of our first algorithmic implementation of the (Nested) Benders Decomposition algorithm for QLPs. The focus of this study was to to examine the effect of different structural properties of QLPs, like a varying number of existential variable blocks resulting in two-stage and the multi-stage instances, but also the effect of different positions of the universal quantifier block in the two-stage case. For more details see [88]. As a consequence there are three sets of two-stage QLPs for five and ten universal variables that only differ in the position of the universal variables block. The block is positioned after the first third, the half, and the second third of the existential variables respectively.

- **EURO13:** The test-set contains 90 instances with 10 – 18 universal variables and 2 – 6 existential variable blocks. The instances have 32 – 480 variables and 27 – 471 constraints (see Appendix A.2.2 for more details of the test instances). This test-set was used in the paper *"Solving Multistage Quantified Linear Optimization*

[14] For the other instances we were not able to genrate feasible instances that accomplished to the properties we required.

Problems with the Alpha-Beta Nested Benders Decomposition" submitted to *"EURO Journal on Computational Optimization"*. The goal was to show how game-tree search techniques as presented in Section 5.2.3 can be integrated into the standard Nested Benders Decomposition algorithm, when using a fast-forward sequencing protocol as described in Section 5.3.

- **TwoStage14:** The test-set contains 135 two-stage instances with 4 – 18 universal variables. The instances 41 – 1880 variables and 27 – 1090 constraints (see Appendix A.2.3 for more details of the test instances). The goal was to create a test-set with a steadily increasing number of universal variables ranging from 4 to 18, with each set of instances for a given number of universal variables, containing the same set of instances by way of comparison. A second goal in contrast to the ESA11 and EURO13 test-sets was to create instances whose upper bound $\exists\forall$-relaxations have no feasible solutions.

- **MultiStage14:** The test-set contains 315 multi-stage instances with 4 – 16 universal variables and 3 – 5 existential variable blocks. The instances have 41 – 1880 variables and 27 – 1090 constraints (see Appendix A.2.4 for more details of the test instances). As in the two-stage case, the goal was to create a test-set with a steadily increasing number of universal variables and stages with each set of instances for a given number of universal variables and number of existential variable block, containing the same set of instances by way of comparison. As in the two-stage case the focus lay on instances whose upper bound $\exists\forall$-relaxations have no feasible solutions.

As you can see in the tables in Appendix A.2.1 – A.2.4, the resulting robustified solutions of the QLPs nearly lose nothing in their optimal objective function value compared to the original LPs, an observation that was already made in [22] with respects to the corresponding robust counterparts (see Section 2.2.2). The aim to examine instances that have feasible upper bound $\exists\forall$-relaxations and also instances that do not have feasible upper bound $\exists\forall$-relaxations is obvious since both cases seem to be possible for real-world instances (see e.g. the QBFLIB instances in Appendix A.3).

A test-set that result from a real-world application is shown in Appendix A.1 and contains 7 two-stage QIP instances that describe *job scheduling* problems. Job scheduling is a classical optimization problem in which jobs have to be assigned to several machines such that the total time until all jobs are finished (the makespan) is minimized. An assignment that a job has to be processed by a machine is called a task and is given a certain duration. If some tasks depend on one another, a full or partial order of tasks can be given. In the corresponding quantified version, the first-stage existential variables are used to determine a schedule. The universal variables represent disturbances that might occur and must be removed by adapting the schedule with the second-stage existential variables at a given cost. The instances have 248 – 6365 variables (3 – 8 universal variables among them) and 167 – 10237 constraints. In our tests we solve the QLP-relaxations of the respective QIPs.

The Quantified Boolean Formulas Satisfiability Library (QBFLIB)

QBFLIB[15] is a collection of instances, solvers, and tools related to the well-known Quantified Boolean Formula (QBF) satisfiability library. The QBFLIB problem suite is currently comprised of more than 13000 instances from several different domains like formal verification, planning and others. In Section 2.3 we showed how such a QSAT instance can be converted into a 0/1-QIP feasibility problem. To create 0/1-QLP optimality problems we decided to use instances from the following QBFLIB test-sets:

- **cnt:** The test-set contains 11 medium-sized multi-stage instances with 10 – 12 universal variables and 11 – 13 existential variable blocks. The instances have 672 – 988 variables an 1791 – 2653 constraints.
- **impl:** The test-set contains 10 small-sized multi-stage instances with 2 – 20 universal variables and 3 – 19 existential variable decision blocks. The instances have 8 – 62 variables and 18 – 162 constraints.
- **TOILET_A:** The test-set contains 78 small- to medium-sized two-stage instances with 2 – 10 universal quantified variables. The instances have 16 – 660 variables and 39 – 13137 constraints.
- **TOILET_C:** The test-set contains 85 small- to medium-sized two-stage instances with 1 – 4 universal quantified variables. The instances have 16 – 960 variables and 37 – 8035 constraints.
- **TOILET_G:** The test-set contains 7 small-sized instances with 1 – 5 universal quantified variables. The instances have10 – 100 variables and 22 – 580 constraints.

Since we are interested in the solution of QLP optimization problems, we added an objective function to the resulting 0/1-QIP feasibility problems and then solved the corresponding QLP-relaxations. As objective function we decided to minimize the sum of the existential variables

$$\min \sum_{i=0}^{|x_\exists|} x_{\exists i}. \tag{6.1}$$

Using QSAT instances to generate QLP optimization problems seems to be a viable approach and generating optimization problems from these problems is no new idea. In [66] QSAT instances were used in the context of QCSP algorithms and converted to optimization problems by having weights associated to the variables and the objective being to minimize the total weights of the variables set to 1. The tables listed in Appendix A.3 show an overview of the resulting QLP instances.

6.3. Computational Results

In the following we describe the tests we performed in the course of the computational evaluation of our algorithm. The reported times are wall-clock times and include the times

[15] http://www.qbflib.org

for algorithm initialization, problem input, optimization, and close down. Times to read the input from hard disc were excluded. When solving the DEP of a QLP, we sometimes split up the times for the creation of the DEP and the times for the final optimization. In all experiments, the value of the objective function reported by our algorithm is compared to the solution of the DEP for both split-variable and compact-view formulation, in order to verify that the correct solution has been found[16].

The interpretation and analysis of data generated in computational evaluations is a non trivial task. In the summarizing tables in the following subsections, we report only the aggregated sums and the *shifted geometric mean* (SGM) for the solution times, which is defined as follows:

$$\left(\prod_{i=1}^{n} v_i + s\right)^{\frac{1}{n}} - s, \text{ for some } s > 0. \tag{6.2}$$

The reason to use the SGM value additional to the sum of the solution times is the following. When aggregating data, the common performance measures arithmetic mean or total sum have the drawback that a small number of the most difficult problems can dominate the other results. Therefore we additionally use the shifted geometric mean, which tends to represent the behavior on the majority of test instances, as the impact of small differences for small instances is reduced by the shifting parameter s, which we set to $s = 10$ for all following results (see also the discussion in [2]).

6.3.1. Evaluation of Two-Stage Techniques

In the following we evaluate the effects of the techniques as described in Section 5.3 when being applied to our standard Benders Decomposition algorithm (see Algorithm 8). We first define a base case to which the changes in the solution times and the number of subproblems that must be solved are compared. The base case and further variants are described in Table 6.1. Each row of the table denotes a version of the Benders Decomposition algorithm with a specific set of parameters enabled, which is denoted by a checkmark in the corresponding column of the row. A checkmark in Column 2 means that the algorithm stops solving the subproblems of a node, when one of its successors is infeasible. In this case the feasibility cut is added to the master problem at the root, which is than directly

Table 6.1. Settings Overview: Benders Decomposition.

Name	BreakInf Cut	InfOpt Cut	Inf&Opt Cut	WcOpt Cut	Move Ordering	ADV Start
BC	×	×	×	×	×	×
BD1	√	√	√	√	×	×
BD2	√	√	√	√	√	×
BD3	√	√	√	√	×	√
BD4	√	√	√	√	√	√

[16] Except the instances that could not be solved with the DEP approach.

solved again. A checkmark in Column 3 denotes that in case of an infeasible subproblem, additionally to the feasibility cut also the corresponding optimality cut is added is to the master, which is then directly resolved again. Column 3 determines whether optimality cuts are only added when all subproblems are feasible, or also while other subproblems are still infeasible. A checkmark in Column 4 means that only the optimality cut that corresponds to the worst-case subsolution of the current iteration is added. Column 5 determines that move-ordering is applied, which means that infeasible subproblem from the previous iteration are visited first in the next iteration. If no infeasible subproblems do exist, subproblems are visited in descending order as constituted by the order of their objective function values from the last iteration, beginning with the worst-case subproblem to this effect.

In the base case (BC) as described in row 1, the BD-method solves all subproblems in each iteration but only adds optimality cuts if no subproblem is infeasible. Otherwise only the feasibility cuts are added. The other extreme case is described in row 5 and adds in each iteration at most three cuts to the master problem. If a subproblem is feasible, the corresponding feasibility and optimality cut are added to the master. Furthermore, the worst-case optimality cut found so far, if one or more feasible subproblems were already visited in the current iteration, are added to the master, which is then solved again. Additionally, move-ordering is applied and the algorithm uses a warm-start from the solution obtained by solving the corresponding LP-relaxation in advance. In the latter case the solution time to solve the LP-relaxation is added to the solution time of the QLP in the following experiments. For the experiments, we chose the following test-sets:

- NETLIB: ESA11, only two-stage, $\forall = \{10\}$ (see Appendix A.2.1).
- NETLIB: EURO13, only two-stage, $\forall = \{10, 15\}$ (see Appendix A.2.2).
- NETLIB: TwoStage14, $\forall = \{6, 8, 10, 12, 14, 16\}$ (see Appendix A.2.3).
- QBFLIB: Toilet_a, $\forall = \{6, 8\}$ (see Appendix A.3).
- QBFLIB: Toilet_c, $\forall = \{6, 8\}$(see Appendix A.3).

Tables 6.2 – 6.3 show the summed up values for the solution times and the corresponding SGM values. Rows of the tables correspond to the test-sets, columns to the Benders Decomposition algorithm variants with the settings as described in Table 6.1. The results show that different cutting schemes can lead to a significant performance gain of up to 60% for the summed up solution times, as well as the corresponding SGM values. The effect of move-ordering and the advanced start procedure in contrast is alternating, in some test they lead to a performance gain, in other sets the overall performance decreases. Adding as few cuts as possible, but therefore strong cuts (worst-case cuts) and cuts that borrow additional information early in the solution process (the corresponding optimality cut in case of an infeasible subproblem) seems therefore to be also a good approach for the Nested Benders Decomposition algorithm in order to keep nodal LPs small and thus, to reduce the solution time and the time to load them into the LP-solver.

Table 6.2. Two-Stage Accelaration Techniques: Solution Times (s).

	BC		BD1		BD2	
TestSet ($\forall, S$)	$\sum$ (s)	SGM (s)	$\sum$ (s)	SGM (s)	$\sum$ (s)	SGM (s)
ESA11 (10,2)	21.00	2.08	11.00 (-47.61)	1.11 (-46.27)	19.00 (-9.52)	1.89 (-9.07)
ESA11 (10,2)	21.00	2.09	14.99 (-28.57)	1.57 (-24.65)	16.00 (-23.79)	1.62 (-22.40)
ESA11 (10,2)	24.00	2.23	17.00 (-29.16)	1.67 (-25.17)	17.99 (-25.00)	1.81 (-19.03)
EURO13 (10,2)	18.00	1.63	14.99 (-16.68)	1.42 (-12.83)	17.00 (-5.56)	1.59 (-2.13)
EURO13 (15,2)	547.00	33.83	318.99 (-41.68)	20.53 (-39.29)	504.00 (-7.86)	31.75 (-6.15)
Toilet_a (6,2)	20.00	1.24	21.00 (+4.99)	1.32 (+6.93)	26.00 (+29.98)	1.63 (+31.21)
Toilet_a (8,2)	1606.00	36.75	1601.00 (-0.31)	36.68 (-0.19)	1705.00 (+6.16)	39.94 (+8.67)
Toilet_c (6,2)	6.00	0.38	7.00 (+16.64)	0.44 (+17.26)	7.99 (+33.28)	0.51 (+34.64)
Toilet_c (8,2)	1141.00	13.97	1143.00 (+0.17)	13.97 (-0.01)	1055.00 (-7.53)	13.72 (-1.77)
TwoStage (6,2)	4.99	0.32	2.00 (-59.95)	0.12 (-60.35)	2.00 (-59.96)	0.12 (-60.36)
TwoStage (8,2)	10.00	0.64	11.99 (+19.95)	0.76 (+18.46)	9.00 (-10.01)	0.57 (-11.24)
TwoStage (10,2)	47.00	2.77	37.00 (-21.27)	2.17 (-21.77)	37.00 (-21.27)	2.24 (-19.12)
TwoStage (12,2)	199.00	9.92	161.00 (-19.09)	7.76 (-21.77)	154.00 (-22.61)	8.04 (-18.93)
TwoStage (14,2)	821.00	33.06	662.00 (-19.36)	24.04 (-27.28)	655.00 (-20.21)	28.00 (-15.31)
TwoStage (16,2)	3487.99	115.59	2717.00 (-22.10)	77.48 (-32.96)	2828.00 (-18.92)	100.43 (-13.10)

Table 6.3. Two-Stage Accelaration Techniques: Solution Times (s) (continued).

	BC		BD3		BD4	
TestSet ($\forall, S$)	$\sum$ (s)	SGM (s)	$\sum$ (s)	SGM (s)	$\sum$ (s)	SGM (s)
ESA11 (10,2)	21.00	2.08	19.00 (-9.52)	1.89 (-9.07)	19.00 (-9.52)	1.87 (-9.89)
ESA11 (10,2)	21.00	2.09	21.00 (-0.00)	2.08 (-0.53)	17.00 (-19.04)	1.72 (-17.82)
ESA11 (10,2)	24.00	2.23	23.00 (-4.16)	2.22 (-0.44)	17.00 (-29.16)	1.72 (-22.79)
EURO13 (10,2)	18.00	1.63	17.00 (-5.55)	1.50 (-7.95)	15.00 (-16.66)	1.37 (-15.54)
EURO13 (15,2)	547.00	33.83	546.00 (-0.18)	33.25 (-1.71)	466.99 (-14.62)	30.43 (-10.03)
Toilet_a (6,2)	20.00	1.24	19.00 (-4.99)	1.17 (-5.23)	24.00 (+19.99)	1.48 (+19.40)
Toilet_a (8,2)	1606.00	36.75	1613.00 (+0.43)	36.92 (+0.47)	1706.00 (+6.22)	40.04 (+8.96)
Toilet_c (6,2)	6.00	0.38	7.99 (+33.28)	0.52 (+36.15)	6.00 (-0.00)	0.38 (-0.00)
Toilet_c (8,2)	1141.00	13.97	1139.00 (-0.17)	13.88 (-0.62)	1055.00 (-7.53)	13.62 (-2.52)
TwoStage (6,2)	4.99	0.32	3.00 (-39.96)	0.19 (-40.35)	2.00 (-59.96)	0.12 (-60.36)
TwoStage (8,2)	10.00	0.64	12.00 (+19.96)	0.77 (+20.17)	6.00 (-39.97)	0.38 (-40.55)
TwoStage (10,2)	47.00	2.77	41.00 (-12.76)	2.43 (-12.23)	36.99 (-21.27)	2.28 (-17.97)
TwoStage (12,2)	199.00	9.92	194.00 (-2.51)	9.80 (-1.19)	151.00 (-24.12)	7.99 (-19.40)
TwoStage (14,2)	821.00	33.06	770.00 (-6.21)	31.27 (-5.42)	658.00 (-19.85)	27.91 (-15.58)
TwoStage (16,2)	3487.99	115.59	3291.00 (-5.64)	110.52 (-4.38)	2791.00 (-19.98)	99.36 (-14.03)

6.3.2. Evaluation of Multi-Stage Techniques

In the following we evaluate the effects of different multi-stage techniques as described in Section 5.3 when being applied to our standard Nested Benders Decomposition algorithm (see Algorithm 9). As in the two-stage case we first define a base case to which the changes in the solution times are compared. The base case and further variants are described in Table 6.4. Additional to the settings also available in the two-stage case, Column 7 and 8 determine whether $\alpha\beta$-cuts are used and the bounding mechanism by solving the QLP-relaxation respectively.

Table 6.4. Settings Overview: Nested Benders Decomposition.

Name	Seq.-Prot.	BreakInf Cut	InfOpt Cut	Inf&Opt Cut	WcOpt Cut	Move Ordering	ADV Start	AB	RC
BC	FF	√	√	√	√	√	×	×	×
NBD1	FF	√	√	√	√	√	×	√	×
NBD2	FF	√	√	√	√	√	×	√	√
NBD3	FFFB	√	√	√	√	√	×	×	×
NBD4	FFFB	√	√	√	√	√	×	√	√
NBD5	FFFFB	√	√	√	√	√	×	×	×
NBD6	FFFFB	√	√	√	√	√	×	√	√

In our tests we evaluate the effect of the different settings when being applied to the three sequencing protocols described in Section 5.3.2 – fast-forward (FF), fast-forward-fast-back (FFFB) and fast-forward-fast-feasible-back (FFFFB). The protocol fast-back (FB) was dropped because prior tests showed that it is clearly outperformed by the other protocols. For the experiments, we chose the following test-sets:

- NETLIB: EURO13, multi-stage, $\forall = \{15, 18\}$ (see Appendix A.2.2).
- NETLIB: MultiStageStage, $\forall = \{14, 16\}$ (see Appendix A.2.4).
- QBFLIB: impl, $\forall = \{2, \ldots, 20\}$ (see Appendix A.3).

Table 6.5 – Table 6.7 show the summed up values for the solution times and the corresponding SGM values for the different settings. The values in parentheses show the percentage change of these values with respect to the base case. Column 1 shows the name of the test-set, Columns 2 and 3 the solution times of the base case. Columns 4 – 6 show the solution times for different settings. The base case is the FF protocol with the optimal settings as established in the previous section for the two-stage case, except the advanced start routine.

Table 6.5 shows the solution times when the base case is compared to the Nested Benders Decomposition algorithm using the FF protocol with $\alpha\beta$-cuts as shown in Columns 4 and 5, and when additionally the bounding mechanism by solving the $\forall$-relaxation at inner nodes is used. The solution times for the latter are shown in Columns 6 and 7. As we can see, the solution process can substantially improved by the usage of $\alpha\beta$-cuts. They lead to an improvement of up to 48% with respect to the summed up solution values and up to 26%

with respect to the SGM values. That the latter result is not as strong as for the summed up values comes from the fact that a few instances can be solved much faster than in the base case while some are solved as fast as before and no positive effect can be observed for them. Additionally using the $\exists\forall$-relaxation leads to a further strong improvement resulting in overall time savings of up to about 74.34% for the summed up solution times as well as the SGM values. However, the effect is weakened when the instances that are solved do not have an upper bound, as e.g. in MultiStage14 test-set.

Table 6.5. Multi-Stage Accelaration Techniques: Solution Times (s).

	BC		NBD1		NBD2	
TestSet ($\forall$, S)	$\sum$	SGM	$\sum$	SGM	$\sum$	SGM
Euro13 (15,3)	265.00	15.67	236.00 (-10.94)	14.96 (-4.54)	71.00 (-73.20)	4.02 (-74.34)
Euro13 (15,5)	428.00	24.48	382.99 (-10.51)	23.42 (-4.30)	270.00 (-36.91)	10.84 (-55.69)
Euro13 (18,3)	2250.00	123.61	2164.00 (-3.82)	115.90 (-6.23)	755.00 (-66.44)	32.06 (-74.06)
Euro13 (18,5)	5627.99	215.77	3899.00 (-30.72)	200.12 (-7.25)	2707.00 (-51.90)	65.84 (-69.48)
MultiStage (14,3)	525.00	13.42	379.00 (-27.80)	10.91 (-18.66)	327.00 (-37.71)	9.27 (-30.88)
MultiStage (16,3)	1150.00	34.84	723.99 (-37.04)	25.46 (-26.91)	697.00 (-39.39)	24.42 (-29.89)
MultiStage (14,4)	1965.00	32.76	1019.99 (-48.09)	25.67 (-21.65)	791.00 (-59.74)	20.94 (-36.08)
MultiStage (16,4)	3490.00	73.93	2483.00 (-28.85)	70.40 (-4.77)	1473.99 (-57.76)	56.86 (-23.08)
MultiStage (14,5)	2603.00	39.16	1728.00 (-33.61)	36.54 (-6.68)	1006.00 (-61.35)	34.87 (-10.94)
MultiStage (16,5)	4466.99	83.52	2570.00 (-42.46)	76.64 (-8.23)	1696.00 (-62.03)	54.21 (-35.08)
impl	764.00	14.47	1505.00 (+96.98)	20.20 (+39.58)	417.00 (-45.41)	10.48 (-27.58)

Table 6.6 shows the solution times when the base case is compared to the Nested Benders Decomposition algorithm using the FFFB protocol with the same settings as in the base case, and when additionally $\alpha\beta$-cuts and the bounding mechanism by solving the $\exists\forall$-relaxation at inner nodes is used. The solution times for the first are shown in Columns 4 and 5 for the latter case they are shown in Columns 6 and 7. As we can see, the FFFP sequencing protocol using the standard settings, is clearly inferior to the FF sequencing protocol in all cases as we can see in Columns 4 and 5. The effect can be mitigated by the use of $\alpha\beta$-cuts and the bounding mechanism as it can be seen in Columns 6 and 7, however, in general the FF protocol, and especially with $\alpha\beta$-cuts and the bounding mechanism being enabled, is faster in all cases.

Table 6.6. Multi-Stage Accelaration Techniques: Solution Times (s)(continued).

	BC		NBD3		NBD4	
TestSet ($\forall$, S)	$\sum$	SGM	$\sum$	SGM	$\sum$	SGM
Euro13 (15,3)	265.00	15.67	297.99 (+12.45)	18.46 (+17.81)	417.00 (+57.35)	8.49 (-45.78)
Euro13 (15,5)	428.00	24.48	759.99 (+77.56)	36.02 (+47.13)	406.00 (-5.14)	14.31 (-41.52)
Euro13 (18,3)	2250.00	123.61	2379.00 (+5.73)	146.02 (+18.12)	4019.00 (+78.62)	46.15 (-62.66)
Euro13 (18,5)	5627.99	215.77	11237.00 (+99.66)	362.08 (+67.80)	4954.99 (-11.95)	100.21 (-53.55)
MultiStage (14,3)	525.00	13.42	1213.00 (+131.04)	24.76 (+84.43)	1232.00 (+134.66)	23.22 (+72.97)
MultiStage (16,3)	1150.00	34.84	3586.00 (+211.82)	50.21 (+44.10)	3610.00 (+213.91)	47.38 (+35.99)
MultiStage (14,4)	1965.00	32.76	3091.00 (+57.30)	55.41 (+69.13)	3699.00 (+88.24)	55.82 (+70.37)
MultiStage (16,4)	3490.00	73.93	11922.00 (+241.60)	193.33 (+161.49)	12465.00 (+257.16)	165.65 (+124.05)
MultiStage (14,5)	2603.00	39.16	1865.00 (-28.35)	52.28 (+33.52)	2401.00 (-7.76)	62.97 (+60.80)
MultiStage (16,5)	4466.99	83.52	4964.00 (+11.12)	138.24 (+65.52)	5931.00 (+32.77)	128.89 (+54.32)
impl	764.00	14.47	5114.00 (+569.37)	41.93 (+189.69)	714.00 (-6.54)	14.17 (-2.06)

Table 6.7 shows the solution times when the base case is compared to the Nested Benders Decomposition algorithm using the FFFFB protocol using the same settings as the base case, and when additionally $\alpha\beta$-cuts and the bounding mechanism by solving the $\exists\forall$-relaxation at inner nodes is used. The solution times for the first are shown in columns 4 and 5 for the latter case they are shown in columns 6 and 7. As for the FFFB protocol, also in this case even the base case of the FF protocol is clearly faster than the base case of the FFFFB protocol in most cases as shown in columns 4 and 5. Solely for many stages and in this case only for very few instances this setting is competitive to the base case. However, if the instances have upper bounds, as e.g. in the Euro13 test-set, the use of $\alpha\beta$-cuts in combination with bounding mechanism works comparably well to the FF protocol with the same settings. Interestingly, this effect cannot be observed when QLP instances are considered that do not have upper bounds. In this case the performance even degrades compared to the FFFFB base case and becomes considerably worse compared to the FF protocol using $\alpha\beta$-cuts in combination with the bounding mechanism.

Table 6.7. Multi-Stage Accelaration Techniques: Solution Times (s)(continued).

	BC		NBD5		NBD6	
TestSet ($\forall$, S)	$\sum$	SGM	$\sum$	SGM	$\sum$	SGM
Euro13 (15,3)	265.00	15.67	233.00 (-12.07)	15.01 (-4.22)	68.99 (-73.96)	4.22 (-73.06)
Euro13 (15,5)	428.00	24.48	496.00 (+15.88)	27.39 (+11.88)	179.99 (-57.94)	10.42 (-57.40)
Euro13 (18,3)	2250.00	123.61	2227.00 (-1.02)	124.98 (+1.10)	749.00 (-66.71)	31.49 (-74.52)
Euro13 (18,5)	5627.99	215.77	7092.00 (+26.01)	277.30 (+28.51)	2146.00 (-61.86)	72.31 (-66.48)
MultiStage (14,3)	525.00	13.42	892.99 (+70.09)	23.05 (+71.74)	883.00 (+68.19)	19.95 (+48.64)
MultiStage (16,3)	1150.00	34.84	2745.00 (+138.69)	52.75 (+51.40)	2722.00 (+136.69)	49.20 (+41.20)
MultiStage (14,4)	1965.00	32.76	2378.00 (+21.01)	45.41 (+38.59)	2579.00 (+31.24)	42.10 (+28.49)
MultiStage (16,4)	3490.00	73.93	8404.00 (+140.80)	152.53 (+106.30)	8268.00 (+136.90)	120.83 (+63.43)
MultiStage (14,5)	2603.00	39.16	1944.00 (-25.31)	47.11 (+20.32)	1504.00 (-42.22)	46.04 (+17.57)
MultiStage (16,5)	4466.99	83.52	3447.99 (-22.81)	99.45 (+19.06)	2640.00 (-40.89)	74.07 (-11.31)
impl	764.00	14.47	752.00 (-1.57)	14.07 (-2.75)	418.00 (-45.28)	10.55 (-27.09)

As a result, the FF protocol with $\alpha\beta$-cuts in combination with the bounding mechanism seems to be the most promising approach to solve multi-stage QLPs in general. However, if a QLP instance has an upper bound (a feasible $\exists\forall$-relaxation does exist), then also the FFFFB sequencing protocol seems to work well with these settings. The reason for this behavior seems to be that in the absence of an upper bound, the FF protocol leads to much stronger cuts compared to the FFFB and the FFFFB protocol, since the former stays much longer in a subtree compared to the latter two protocols. As a result, the optimality cuts and also the values used in the bounding step are much stronger compared to the other protocols.

6.3.3. Comparison with external LP-Solvers applied to the DEP

In the following we compare our (Nested) Benders Decomposition algorithm with the approach to solve the compact-view DEP of a QLP with the standard external LP-solver CPLEX. Therefore, CPLEX works with the dual simplex algorithm using the standard settings and its preprocessor being enabled. We again distinguish between the two-stage and the multi-stage case.

Two-Stage Tests: Benders Decompsotion vs. DEP

For the two-stage case we run the Benders Decomposition algorithm with all settings being enabled as visualized in row 8 of Table 6.1. For these experiments, we chose the following test-sets:

- NETLIB: ESA11, only two-stage, $\forall = \{5, 10\}$ (see Appendix A.2.1).
- NETLIB: EURO13, only two-stage, $\forall = \{10, 15, 18\}$ (see Appendix A.2.2).
- NETLIB: TwoStage, $\forall = \{4, 6, 8, 10, 12, 14, 16\}$ (see Appendix A.2.3).
- QBFLIB: Toilet_a, $\forall = \{4, 6, 8, 10\}$ (see Appendix A.3).
- QBFLIB: Toilet_c, $\forall = \{4, 6, 8, 10\}$(see Appendix A.3).

Table 6.8 shows the summed up values for the solution times and the corresponding SGM values for both solvers, starting with the solution times for CPLEX in columns 2 and 3. Column 4 and 5 show the solution times for the Benders Decomposition algorithm and Column 6 and 7 show the percentage change of these values between both algorithms.

Table 6.8. Two-Stage Tests: Benders Decomposition vs. DEP.

	Solution times DEP (s)		Solution times BD (s)			
TestSet ($\forall$, S)	$\sum$	SGM	$\sum$	SGM	$\sum$ (%)	SGM (%)
SCHEDULING	39.00	3.07	1.00	0.13	-97.43	-95.54
ESA11 (5,2)	2.00	0.21	0.10	0.01	-99.94	-99.99
ESA11 (5,2)	2.00	0.21	0.10	0.01	-99.94	-99.99
ESA11 (5,2)	1.00	0.10	0.10	0.01	-99.88	-99.97
ESA11 (10,2)	248.00	12.98	5.00	0.53	-97.98	-95.88
ESA11 (10,2)	194.00	10.02	6.99	0.73	-96.39	-92.70
ESA11 (10,2)	162.00	8.23	6.00	0.63	-96.29	-92.27
EURO13 (10,2)	140.00	7.86	6.00	0.56	-95.71	-92.83
EURO13 (15,2)	$7090.00^{(1/10)}$	315.35	342.00	22.04	-95.17	-93.00
EURO13 (18,2)	×	×	2142.00	62.04	×	×
TwoStage (4,2)	2.00	0.12	0.00	0.00	-99.91	-99.98
TwoStage (6,2)	6.00	0.38	1.00	0.06	-83.31	-83.60
TwoStage (8,2)	38.00	2.24	3.00	0.19	-92.10	-91.44
TwoStage (10,2)	400.00	14.44	8.00	0.50	-97.99	-96.47
TwoStage (12,2)	3892.00	80.00	22.00	1.33	-99.43	-98.33
TwoStage (14,2)	$15344.00^{(2/13)}$	406.59	92.99	5.12	-99.39	-98.74
TwoStage (16,2)	×	×	412.00	16.62	×	×
Toilet_a (4,2)	1.00	0.10	0.10	0.01	-99.88	-99.96
Toilet_a (6,2)	37.00	2.31	8.99	0.58	-75.67	-74.82
Toilet_a (8,2)	21401.99	233.31	442.00	13.21	-97.93	-94.33
Toilet_a (10,2)	62894.00	1815.55	7847.00	197.52	-87.52	-89.12
Toilet_c (4,2)	0.80	0.05	1.70	0.11	+97.70	+100.88
Toilet_c (6,2)	1.00	0.06	2.00	0.12	+99.70	+100.48
Toilet_c (8,2)	24.00	0.84	53.00	1.59	+120.80	+88.29
Toilet_c (10,2)	302.00	5.15	302.00	5.03	+0.00	-2.34

The superscripted pair (x/y) denotes that x instances of the corresponding test-set could not be solved within the time-limit, where y is the total number of instances in the test-set. A $\times$ denotes that the corresponding test-set is ignored for the corresponding algorithm, since most of the instances could not be solved within the time-limit of 3600 seconds per instance (e.g. the EURO13 test-set with 18 universal variables). The results show that the Benders Decomposition algorithm is clearly faster than solving the DEP in most cases, especially with an increasing number of universally quantified variables. However, for the Toilet_c test-set solving the DEP is faster in most cases, because these instances only have up to four universal variables and the resulting DEPs are small. We can see that for our QLP instances generated from the NETLIB test-set, the first instances could not be solved within the time-limit using the DEP for 14 universal variables, for 16 universal variables the majority of the instances could not be solved and in some cases even the available memory of 128 GB RAM did not suffice and the LP-solver process was terminated by the system. Our Benders Decomposition algorithm in contrast was able to solve all test instances within the time-limit using at most 2 GB RAM.

Multi-Stage Tests: Nested Benders Decomposition vs. DEP.

For the multi-stage case we run the Nested Benders Decomposition algorithm using the fast-forward sequencing protocol with all settings except advanced start being enabled as visualized in row 3 of Table 6.4. For the experiments, we chose the following test-sets:

- NETLIB: ESA11, $\forall = 10$ (see Appendix A.2.1).
- NETLIB: EURO13, $\forall = \{10, 15, 18\}$ (see Appendix A.2.2).
- NETLIB: MultiStage, $\forall = \{6, 8, 10, 12, 14, 16\}$ (see Appendix A.2.3).
- QBFLIB: impl, $\forall = \{2, \ldots, 20\}$ (see Appendix A.3).
- QBFLIB: cnt, $\forall = \{11, 12, 13\}$(see Appendix A.3).

Table 6.9 shows the summed up values of the solution times and the corresponding SGM values for both solvers, in the same order as above for the two-stage case. Again, the results show that the Nested Benders Decomposition algorithm is faster than solving the DEP in most cases, especially with an increasing number of universally quantified variables. However, for an increasing number of existential variable blocks, the effect appears later than in the two-stage case. One reason is that the solution times for solving the DEP decrease with an increasing number of quantifier blocks, because in split-variable formulation and equally distributed quantifier blocks, the resulting DEP gets smaller than in the two-stage case. For the NBD algorithm the opposite is the case because there is an increasing overhead for tree traversal (e.g. saving and restoring cuts and bases).

Apart from time savings, another great advantage of our algorithm is its memory consumption. In the test-sets with the raised (*) – EURO13 test-sets with 18 universal variables – the instance SCTAP1 could not be solved with the DEP approach due to memory problems, even though 128 GB RAM were available. At the same time, our Nested Benders

Decomposition algorithm did not consume more than 2 GB RAM for any instance in the entire test-sets. Thus, while special hardware with very much RAM is needed to solve QLPs of interesting size with the DEP approach, our algorithm is capable to run on very common standard hardware.

However, it should be noted that in the last QBFLIB (cnt) test-set three instances could not be solved with our Nested Benders Decomposition algorithm within the defined time-limit of 3600 seconds. However, also the solution of two of these instances with the DEP approach failed.

Table 6.9. Multi-Stage Tests: Nested Benders Decomposition vs. DEP.

	Solution times DEP (s)		Solution times NBD (s)			
TestSet ($\forall$, S)	$\sum$	SGM	$\sum$	SGM	$\sum$ (%)	SGM (%)
ESA11 (10,3)	118.00	7.81	5.00	0.52	-95.76	-93.28
ESA11 (10,5)	79.99	5.96	27.99	2.43	-64.99	-59.11
Euro13 (10,3)	83.00	5.43	5.00	0.45	-93.97	-91.62
Euro13 (15,3)	9214.00	225.95	76.00	4.35	-99.17	-98.07
Euro13 (18,3) *	$22881.00^{(1/10)}$	2004.36	434.99	24.39	-98.09	-98.78
Euro13 (10,3)	68.99	4.75	18.00	1.58	-73.91	-66.64
Euro13 (15,3)	$7716.99^{(4/9)}$	216.56	272.00	11.05	-96.47	-94.89
Euro13 (18,3)*	$27171.99^{(6/9)}$	2870.73	759.00	44.07	-97.20	-98.46
MultiStage (6,3)	8.00	0.51	8.00	0.47	+0.01	-8.50
MultiStage (8,3)	32.00	1.90	15.99	0.97	-50.01	-49.03
MultiStage (10,3)	311.99	12.85	34.00	1.79	-89.10	-86.06
MultiStage (12,3)	2732.00	66.88	109.00	4.29	-96.01	-93.57
MultiStage (14,3)	$16936.00^{(1/13)}$	387.73	347.00	9.58	-97.95	-97.52
MultiStage (16,3)	$27186.99^{(5/13)}$	1191.33	708.00	25.15	-97.39	-97.88
MultiStage (6,4)	5.00	0.32	27.99	1.49	+459.72	+361.98
MultiStage (8,4)	31.00	1.88	27.00	1.60	-12.91	-14.91
MultiStage (10,4)	223.00	10.09	68.00	3.53	-69.50	-64.94
MultiStage (12,4)	2355.00	62.82	211.00	7.71	-91.04	-87.71
MultiStage (14,4)	$18085.00^{(1/13)}$	422.77	829.00	21.32	-95.41	-94.95
MultiStage (16,4)	$24432.00^{(5/13)}$	739.60	1499.99	57.74	-93.86	-92.19
MultiStage (6,5)	4.00	0.25	13.00	0.83	+224.78	+230.35
MultiStage (8,5)	25.00	1.53	62.00	2.67	+147.99	+74.76
MultiStage (10,5)	153.00	7.88	208.00	6.32	+35.94	-19.70
MultiStage (12,5)	1625.00	47.41	318.00	11.39	-80.43	-75.97
MultiStage (14,5)	$17246.00^{(1/13)}$	434.69	1059.00	36.59	-93.85	-91.58
MultiStage (16,5)	$24634.00^{(5/13)}$	783.72	1710.00	54.79	-93.05	-93.00
impl	$8288.00^{(2/10)}$	54.61	425.99	10.62	-94.86	-80.55
cnt	$7756.00^{(2/11)}$	54.04	$10763.99^{(3/11)}$	64.29	+38.78	+18.97

6.3.4. Comparions with Quantifier Elimination Techniques

An algorithm to solve QLPs based on the Fourier-Motzkin Elimination method was presented in Section 3.1. In Section 2.2 we already mentioned that more advanced quantifier elimination techniques were also successfully used to solve quantified formulas over the

reals (i.e., first-order formulae over the real numbers). As an example in nonlinear real arithmetic they can be used to decide the following formula:

$$\forall x \, \exists y \, (x^2 + xy + b > 0 \wedge x + ay^2 + b \leq 0). \tag{6.3}$$

This formula can be thought of as a generalization of QLPs, and thus, existing solvers for this class of problems can theoretically also be used to solve QLPs. However, it is known that large instances in general are currently out-of-reach, but some problems in low dimensions and with a small number of quantifier alternations can be solved [50]. In the following we examine if these methods are also appropriate to solve the QLP instances used in our experimental study. We decided to use the following Solvers:

- **QEPCAD-B:** QEPCAD-B[17] [55] is an implementation of quantifier elimination by partial cylindrical algebraic decomposition (CAD) for computing with real algebraic sets.

- **Redlog:** Redlog[18] [84] is part of the interactive computer algebra system *Reduce* and supports various functions on first-order formulas. It supports Nonlinear Real Arithmetic and Presburger Arithmetic among others and uses quantifier elimination procedures for the various supported theories.

- **RSolver:** RSolver[19] [177] is a program for solving quantified inequality constraints over the real numbers.

In the following, these solvers are compared with the Quantifier Elimination Algorithm (QEA) as described in Section 3.1 and with our (Nested) Benders Decomposition algorithm as described in Section 5.2. For our tests we restricted ourselves to the five smallest instances QBFLIB test set with less than 75 variables and 360 constraints from the test-sets TOILET_A, TOILET_C and TOILET_G. We used a time-limit of 3600 seconds for each of the solvers, and furthermore only checked the instances for feasibility, since Redllog, QEPCAD and Rsolver do not support the optimization of an objective function.

The results are displayed in Table 6.10, where the rows correspond to single instances. Column 1 contains the name of an instance, where instances with a raised $*$ are infeasible. Column 2 and 3 show the corresponding number of existential and universal variables respectively. Column 4, 5, and 6 show the number of constraints, the density of the constraint matrix and the number of existential quantifier blocks. Columns 7 – 11 show the solution times of the specific solvers. A cross indicates that a solver was not able to solve the corresponding instance within the time limit, otherwise the solution time in seconds is shown.

As we can see, all solvers perform very bad compared to our Benders Decomposition algorithm, whose solution times are shown in Column 11 and which solves all instances

[17] `http://www.usna.edu/CS/~qepcad/B/QEPCAD.html`
[18] `http://www.redlog.eu`
[19] `http://rsolver.sourceforge.net`

nearly instantly. While RSolver did not even solve one single instance within the time-limit of 3600 seconds as shown in Column 9, QEPCAD-B was at least capable to solve the smallest instance – toilet_g_02_01.2 with 11 variables and 22 constraints – in a few seconds as shown in Column 8. Redlog performs better and is able to solve at least 2–3 instances of each test-set. However, the solution times increase significantly even when the number of variables or constraints only grows slightly as we can see in column 7. A similar observation can be made for the QEA-method as shown in Column 10. However, while the other methods failed to solve the instances due to the time-limit of 3600 seconds, the QEA-method exceeded the available memory of 128 GB due to a very large number of new constraints that were computed during computation (see also Section 3.1).

Table 6.10. Quantifier Elimination vs. Benders Decomposition: Solution times (s).

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	Redlog	QEPCAD	RSolver	QEA	NBD
toilet_a_02_01.2*	16	2	39	12.25	2	2.24	×	×	5.768	0.001
toilet_a_02_01.3	24	2	64	8.59	2	2855.92	×	×	×	0.001
toilet_a_02_01.4	32	2	89	6.61	2	×	×	×	×	0.001
toilet_a_02_05.2	48	2	163	4.39	2	×	×	×	×	0.001
toilet_a_04_01.2*	28	4	129	10.71	2	×	×	×	×	0.003
toilet_a_04_01.3	42	4	179	6.74	2	×	×	×	×	0.002
toilet_c_02_01.2*	16	1	37	12.08	2	0.54	×	×	1.456	0.001
toilet_c_02_01.3	24	1	62	8.58	2	1588.13	×	×	123.76	0.001
toilet_c_02_01.4	32	1	87	6.62	2	×	×	×	×	0.001
toilet_c_02_05.2	48	1	161	4.41	2	×	×	×	×	0.001
toilet_c_04_01.2*	28	2	85	7.45	2	×	×	×	×	0.103
toilet_c_04_01.3	42	2	135	5.10	2	1394.75	×	×	×	0.043
toilet_g_02_01.2	10	1	22	18.18	2	0.57	6.44	×	0.001	0.001
toilet_g_04_01.2	20	2	52	10.49	2	16.38	×	×	0.023	0.001
toilet_g_06_01.2	30	3	90	8.48	2	2550.97	×	×	0.537	0.001
toilet_g_08_01.2	40	3	136	6.84	2	×	×	×	2.991	0.001
toilet_g_10_01.2	50	4	190	6.63	2	×	×	×	20.990	0.001
toilet_g_15_01.2	75	4	360	4.91	2	×	×	×	61.085	0.003

Apart from these tests, we also tried to apply the QEA-method to small QLP optimization problems with less than 100 variables from our NETLIB test-sets ESA11, EURO13, TwoStage and MultiStage. However, the QEA-method was not able to solve any of these instances because it again exceeded the available memory in each case after the elimination of a few variables. The overall observed results confirm that solving QLP instances with quantifier elimination and polyhedral projection techniques seems disillusioning. Instead, techniques that explicitly handle the underlying special structure of the linear constraints in the solution process seem to be mandatory to solve QLP optimization problems of interesting size.

7. Conclusion and Outlook

7.1. Conclusion

In this thesis we presented the concept of quantified linear and integer programming as an independent mathematical programming paradigm in the context of optimization under uncertainty. We gave a detailed introduction for the continuous and the integer case and highlighted the strong similarities to two-person zero-sum games. We introduced variable quantification as an alternative approach to model uncertainty or adversarial situations, apart from the dominating modeling paradigms stochastic programming and robust optimization. We furthermore presented several illustrating examples how to model real-world decision and optimization problems with this approach.

We focussed on QLPs and provided a detailed study of their algorithmic properties by a comprehensive geometric analysis and the result that the solution space of a QLP forms a polytope in $\mathbb{Q}^n$. We presented a complete and thorough complexity analysis by using a polyhedral approach and showed that the vertices of a QLP solution polytope can be encoded with polynomially many bits if the number of quantifier changes is a constant, leading to the conclusion that in this case the QLP problem belongs to the complexity class **PSPACE**. Furthermore, we showed that it suffices to inspect the bounds of the universal variables in order to solve the QLP problem, reducing a QLP to exponentially many LPs using the rules imposed by the universal player branching tree.

Based on these investigations, we proposed an algorithm that exploits the problem's hybrid nature of being a two-person zero-sum game on the one side, and being a convex multi-stage decision problem on the other side, to combine linear programming techniques with solution techniques from game-tree search. We proposed an extension of the classical Nested Benders Decomposition algorithm to solve QLP optimization problems. The proposed algorithm was enhanced by $\alpha\beta$-cuts and move-ordering, two techniques that are used in the classical Alpha-Beta Algorithm to evaluate minimax-trees, which are the basis of many game playing programs to represent positions and moves. We furthermore showed how QLP-relaxations that result from quantifier shifting can improve our algorithm with an operation similar to the bounding step as it is used in a traditional branch-and-bound algorithm when solving (mixed-)integer linear programs.

The applicability of our algorithmic approach was evaluated in a detailed computational study where we solved instances generated from the well-known NETLIB linear programming test set, as well as instances generated from the quantified boolean formula (QBF) satisfiability test set QBFLIB. We showed that the integration of game-tree techniques and the information obtained from the solution of QLP-relaxations can lead to a significant speedup compared to our standard Nested Benders Decomposition implementation. Furthermore, we demonstrated the clear advantage of our algorithmic approach against solv-

ing the DEP of a QLP with external standard LP solver software like CPLEX and Gurobi. While solving the DEP is slightly faster for a small number of universal variables, our algorithm clearly outperforms the DEP approach for an increasing number of universally quantified variables. Apart from the significant time savings, another great advantage are the small memory requirements of our algorithm compared tot he DEP approach. While in the latter case, some DEPs could even not be solved with 128 GB of memory, our algorithm did not use more than 2 GB of memory for any of the instances considered in our tests. For sake of completeness we also examined the applicability of solvers based on quantifier elimination, which are mainly used in the context of more general quantified formulas over the reals, as well as polyhedral projection when being applied to QLPs. The results suggest that these methods are not nearly competitive to our algorithm and the DEP approach.

This thesis is furthermore supplemented by the software `QlpOpt`, which is a solver and framework for quantified linear programming that features different solution techniques and a variety of methods to work with QLPs and QIPs. It uses state of the art software library components for LP and MIP optimization, which can be updated and replaced easily as new advances are made in these technologies. By the use of standard libraries for operating system abstractions and other platform dependent functions, we support interoperability allowing `QlpOpt` to run on a variety of platforms like Apple OS X, Linux and Microsoft Windows variants.

7.2. Outlook

The results of this thesis mainly affect QLPs, which apart from being a highly interesting problem class themselves, are relaxations of QIPs to the continuous case and at the same time they have structural properties of traditional LPs. When solving (mixed-)integer linear programs (MIPs), polyhedral techniques are very successful in theory and practice and the solution of IPs and MIPs typically involves repeated solution of the problem's LP-relaxation. We think that QLPs can influence the solution of QIPs and QMIPs in a similar way, resulting in a considerable speed up of the solution process. Consequently, we think that the algorithmic advance in the solution of QLPs presented in this thesis will have a strong practical impact.

The presented integration of game-tree techniques into existing LP-methods is appealing and rises the question whether techniques from other fields, like quantified boolean reasoning or constraint satisfaction, can also be integrated.

Apart from that, we hope that the framework proposed in this thesis finds an audience within the wider optimization community, insofar as practical applications are concerned and solved. Maybe in a few years quantified linear and integer programming plays such an important role in the field of optimization under uncertainty as stochastic programming and robust optimization does nowadays.

A. Test Set Statictics

The following tables show statistics of the problem instances considered in Section 6.3. We give the name of the instances, the number of existential variables ($x_\exists$), the number of universal variables ($x_\forall$), the number of constraints ($|C|$), the density of the constraint matrix (D) and the number of stages (S). We furthermore give the lower bound resulting from the $\forall\exists$-relaxation ($Z_{\forall\exists}$), the original objective function value (Z_Q) and the upper bound resulting from the solution of the $\exists\forall$-relaxation ($Z_{\exists\forall}$). If a problem is infeasible we denote this by $\times$ in the corresponding row and column.

A.1. Scheduling QLPs

Table A.1. Scheduling

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
scheduling1	248	3	167	4.30	2	64.29	64.29	$\times$
scheduling2	1310	5	1685	0.95	2	64.07	64.07	$\times$
scheduling2b	1310	5	1733	0.97	2	195.22	195.22	$\times$
scheduling3	283	4	289	2.82	2	8.14	8.14	$\times$
scheduling4a	693	5	949	1.31	2	9.07	9.07	$\times$
scheduling4b	1945	6	3421	0.57	2	11.04	11.04	$\times$
scheduling5	6365	8	10237	0.21	2	13.01	13.01	$\times$

A.2. NETLIB QLPs

A.2.1. NETLIB: ESA11

Table A.2. NETLIB: ESA11 ($x_\forall = 5, S = 2$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	5	27	11.31	2	-460.65	-460.50	-459.60
KB2.qlp	41	5	43	16.99	2	-1749.81	-1749.74	-1749.72
SC105.qlp	103	5	104	3.65	2	-51.83	-51.51	-51.25
SC205.qlp	203	5	204	1.90	2	-51.90	-51.22	-51.15
SC50A.qlp	48	5	49	7.32	2	-64.14	-63.93	-63.52
SC50B.qlp	48	5	48	7.00	2	-69.66	-69.40	-69.13
SCAGR25.qlp	500	5	471	0.90	2	-14752575.65	-14752566.07	-14751794.28
SCAGR7.qlp	140	5	129	3.10	2	-2331345.51	-2331343.24	-2331312.21
SCTAP1.qlp	480	5	300	1.42	2	1454.93	1492.08	1570.24

Table A.3. NETLIB: ESA11 ($x_\forall = 5, S = 2$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	5	27	11.31	2	-460.58	-458.37	-457.43
KB2.qlp	41	5	43	16.99	2	-1749.77	-1749.69	-1749.67
SC105.qlp	103	5	104	3.65	2	-52.11	-51.48	-51.36
SC205.qlp	203	5	204	1.90	2	-51.57	-51.07	-51.05
SC50A.qlp	48	5	49	7.32	2	-64.35	-63.88	-63.43
SC50B.qlp	48	5	48	7.00	2	-69.85	-69.56	-69.15
SCAGR25.qlp	500	5	471	0.90	2	-14753410.34	-14753056.78	-14752743.46
SCAGR7.qlp	140	5	129	3.10	2	-2331386.06	-2331334.74	-2331320.36
SCTAP1.qlp	480	5	300	1.42	2	1448.30	1505.99	1543.38

Table A.4. NETLIB: ESA11 ($x_\forall = 5, S = 2$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	5	27	11.31	2	-459.20	-456.76	-456.76
KB2.qlp	41	5	43	16.99	2	-1749.75	-1749.73	-1749.72
SC105.qlp	103	5	104	3.65	2	-51.89	-51.29	-51.20
SC205.qlp	203	5	204	1.90	2	-51.89	-51.21	-51.20
SC50A.qlp	48	5	49	7.32	2	-63.69	-63.25	-62.95
SC50B.qlp	48	5	48	7.00	2	-69.56	-69.09	-68.90
SCAGR25.qlp	500	5	471	0.90	2	-14752613.29	-14751831.82	-14751708.05
SCAGR7.qlp	140	5	129	3.10	2	-2331361.91	-2331330.21	-2331330.05
SCTAP1.qlp	480	5	300	1.42	2	1454.07	1537.23	1551.23

Table A.5. NETLIB: ESA11 ($x_\forall = 5, S = 3$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	5	27	11.31	3	-461.45	-461.45	-461.15
KB2.qlp	41	5	43	16.99	3	-1749.60	-1749.56	-1749.55
SC105.qlp	103	5	104	3.65	3	-51.60	-51.14	-50.96
SC205.qlp	203	5	204	1.90	3	-51.85	-51.30	-51.26
SC50A.qlp	48	5	49	7.32	3	-64.44	-64.04	-63.38
SC50B.qlp	48	5	48	7.00	3	-69.61	-69.35	-68.74
SCAGR25.qlp	500	5	471	0.90	3	-14753061.50	-14752718.62	-14752123.24
SCAGR7.qlp	140	5	129	3.10	3	-2331369.25	-2331342.36	-2331322.74
SCTAP1.qlp	480	5	300	1.42	3	1448.96	1507.51	1555.11

Table A.6. NETLIB: ESA11 ($x_\forall = 5, S = 6$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	5	27	11.31	6	-460.43	-459.89	-459.28
KB2.qlp	41	5	43	16.99	6	-1749.71	-1749.68	-1749.67
SC105.qlp	103	5	104	3.65	6	-51.97	-51.53	-51.33
SC205.qlp	203	5	204	1.90	6	-51.87	-51.32	-51.24
SC50A.qlp	48	5	49	7.32	6	-63.94	-63.52	-63.41
SC50B.qlp	48	5	48	7.00	6	-69.19	-68.87	-68.68
SCAGR25.qlp	500	5	471	0.90	6	-14752774.89	-14752518.85	-14752367.04
SCAGR7.qlp	140	5	129	3.10	6	-2331377.27	-2331349.45	-2331333.58
SCTAP1.qlp	480	5	300	1.42	6	1458.80	1520.93	1550.43

Table A.7. NETLIB: ESA11 ($x_\forall = 10, S = 2$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	10	27	12.61	2	-453.80	-450.49	-446.73
KB2.qlp	41	10	43	17.60	2	-1749.69	-1749.61	-1749.55
SC105.qlp	103	10	104	4.59	2	-51.78	-50.70	-50.24
SC205.qlp	203	10	204	2.44	2	-51.27	-50.18	-50.05
SC50A.qlp	48	10	49	8.80	2	-63.81	-62.93	-62.02
SC50B.qlp	48	10	48	8.55	2	-69.70	-69.04	-68.46
SCAGR25.qlp	500	10	471	1.13	2	-14751981.78	-14751959.88	-14750401.50
SCAGR7.qlp	140	10	129	3.82	2	-2331375.98	-2331345.91	-2331267.60
SCTAP1.qlp	480	10	300	1.66	2	1488.88	1581.53	1709.98

Table A.8. NETLIB: ESA11 ($x_\forall = 10, S = 2$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	10	27	12.61	2	-458.83	-456.29	-455.11
KB2.qlp	41	10	43	17.60	2	-1749.67	-1749.52	-1749.46
SC105.qlp	103	10	104	4.59	2	-51.65	-50.61	-50.35
SC205.qlp	203	10	204	2.44	2	-51.48	-50.15	-50.10
SC50A.qlp	48	10	49	8.80	2	-63.09	-62.34	-61.90
SC50B.qlp	48	10	48	8.55	2	-69.35	-68.62	-67.98
SCAGR25.qlp	500	10	471	1.13	2	-14752457.45	-14752062.47	-14751308.43
SCAGR7.qlp	140	10	129	3.82	2	-2331329.26	-2331293.43	-2331279.65
SCTAP1.qlp	480	10	300	1.66	2	1512.30	1645.07	1709.54

Table A.9. NETLIB: ESA11 ($x_\forall = 10, S = 2$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	10	27	12.61	2	-458.74	-455.56	-454.71
KB2.qlp	41	10	43	17.60	2	-1749.81	-1749.68	-1749.65
SC105.qlp	103	10	104	4.59	2	-51.52	-50.40	-50.24
SC205.qlp	203	10	204	2.44	2	-51.58	-50.40	-50.37
SC50A.qlp	48	10	49	8.80	2	-63.34	-62.47	-62.19
SC50B.qlp	48	10	48	8.55	2	-69.25	-68.40	-67.93
SCAGR25.qlp	500	10	471	1.13	2	-14753196.86	-14751487.33	-14751406.54
SCAGR7.qlp	140	10	129	3.82	2	-2331349.41	-2331291.68	-2331288.27
SCTAP1.qlp	480	10	300	1.66	2	1449.38	1615.70	1642.55

Table A.10. NETLIB: ESA11 ($x_\forall = 10, S = 3$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	10	27	12.61	3	-460.11	-458.64	-455.12
KB2.qlp	41	10	43	17.60	3	-1749.77	-1749.60	-1749.53
SC105.qlp	103	10	104	4.59	3	-51.12	-50.03	-49.75
SC205.qlp	203	10	204	2.44	3	-51.46	-50.31	-50.25
SC50A.qlp	48	10	49	8.80	3	-63.81	-63.21	-62.71
SC50B.qlp	48	10	48	8.55	3	-69.26	-68.63	-68.13
SCAGR25.qlp	500	10	471	1.13	3	-14752561.92	-14751631.40	-14750725.98
SCAGR7.qlp	140	10	129	3.82	3	-2331346.09	-2331322.48	-2331266.52
SCTAP1.qlp	480	10	300	1.66	3	1487.85	1608.56	1674.58

Table A.11. NETLIB: ESA11 ($x_\forall = 10, S = 6$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	10	27	12.61	6	-454.83	-452.84	-448.78
KB2.qlp	41	10	43	17.60	6	-1749.36	-1749.30	-1749.22
SC105.qlp	103	10	104	4.59	6	-51.70	-50.57	-50.26
SC205.qlp	203	10	204	2.44	6	-51.81	-50.85	-50.71
SC50A.qlp	48	10	49	8.80	6	-63.43	-62.23	-61.62
SC50B.qlp	48	10	48	8.55	6	-69.76	-69.15	-68.37
SCAGR25.qlp	500	10	471	1.13	6	-14752779.62	-14752153.65	-14751306.05
SCAGR7.qlp	140	10	129	3.82	6	-2331327.62	-2331281.48	-2331249.17
SCTAP1.qlp	480	10	300	1.66	6	1507.35	1650.54	1732.91

A.2.2. NETLIB: EURO13

Table A.12. NETLIB: EURO13 ($x_\forall = 10, S = 2$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	10	27	12.61	2	-423.69	-401.27	-395.10
KB2.qlp	41	10	43	17.60	2	-1749.32	-1748.53	-1748.42
RECIPELP.qlp	180	10	91	5.11	2	-266.62	-266.62	-266.62
SC105.qlp	103	10	104	4.59	2	-49.26	-43.52	-43.11
SC205.qlp	203	10	204	2.44	2	-51.03	-44.75	-44.55
SC50A.qlp	48	10	49	8.80	2	-63.80	-59.94	-58.10
SC50B.qlp	48	10	48	8.55	2	-66.66	-64.40	-60.25
SCAGR25.qlp	500	10	471	1.13	2	-14747499.81	-14745150.32	-14740602.59
SCAGR7.qlp	140	10	129	3.82	2	-2331101.64	-2330827.81	-2330696.43
SCTAP1.qlp	480	10	300	1.66	2	1941.55	2813.08	3155.94

Table A.13. NETLIB: EURO13 ($x_\forall = 10, S = 3$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	10	27	12.61	3	-409.67	-389.19	-373.66
KB2.qlp	41	10	43	17.60	3	-1749.25	-1748.54	-1748.31
RECIPELP.qlp	180	10	91	5.11	3	-266.62	-266.62	-266.62
SC105.qlp	103	10	104	4.59	3	-50.15	-44.73	-44.14
SC205.qlp	203	10	204	2.44	3	-43.42	-30.46	-30.46
SC50A.qlp	48	10	49	8.80	3	-56.63	-49.31	-49.31
SC50B.qlp	48	10	48	8.55	3	-65.15	-62.46	-59.46
SCAGR25.qlp	500	10	471	1.13	3	-14745322.54	-14741546.24	-14737234.05
SCAGR7.qlp	140	10	129	3.82	3	-2331143.18	-2330989.41	-2330891.07
SCTAP1.qlp	480	10	300	1.66	3	1608.79	2422.07	2902.25

Table A.14. NETLIB: EURO13 ($x_\forall = 10, S = 6$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	10	27	12.61	6	-418.62	-402.86	-386.56
KB2.qlp	41	10	43	17.60	6	-1748.69	-1748.13	-1747.94
RECIPELP.qlp	180	10	91	5.11	6	-266.62	-266.62	-266.62
SC105.qlp	103	10	104	4.59	6	-48.68	-44.02	-43.04
SC205.qlp	203	10	204	2.44	6	-49.81	-45.00	-44.39
SC50A.qlp	48	10	49	8.80	6	-59.54	-46.30	-46.30
SC50B.qlp	48	10	48	8.55	6	-65.26	-61.97	-59.21
SCAGR25.qlp	500	10	471	1.13	6	-14749091.45	-14746149.51	-14743323.91
SCAGR7.qlp	140	10	129	3.82	6	-2331213.44	-2330949.78	-2330890.70
SCTAP1.qlp	480	10	300	1.66	6	1991.61	2777.73	3255.32

Table A.15. NETLIB: EURO13 ($x_\forall = 15, S = 2$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	15	27	13.63	2	-400.21	-382.61	-373.97
KB2.qlp	41	15	43	18.11	2	-1748.87	-1748.24	-1747.65
RECIPELP.qlp	180	15	91	5.60	2	-266.62	-266.62	-266.62
SC105.qlp	103	15	104	5.46	2	-24.35	-18.64	-18.64
SC205.qlp	203	15	204	2.96	2	-49.81	-38.44	-38.44
SC50A.qlp	48	15	49	10.04	2	-56.38	-52.08	-48.60
SC50B.qlp	48	15	48	9.85	2	-62.91	-58.17	-54.45
SCAGR25.qlp	500	15	471	1.36	2	-14747268.16	-14741752.51	-14733682.41
SCAGR7.qlp	140	15	129	4.50	2	-2331081.60	-2330652.39	-2330622.24
SCTAP1.qlp	480	15	300	1.90	2	1932.26	3211.79	3826.39

Table A.16. NETLIB: EURO13 ($x_\forall = 15, S = 3$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	15	27	13.63	3	-402.59	-388.63	-375.58
KB2.qlp	41	15	43	18.11	3	-1747.77	-1746.84	-1746.42
RECIPELP.qlp	180	15	91	5.60	3	-266.62	-266.62	-266.62
SC105.qlp	103	15	104	5.46	3	-34.26	-28.99	-28.99
SC205.qlp	203	15	204	2.96	3	-33.78	-8.63	-8.63
SC50A.qlp	48	15	49	10.04	3	-56.16	-50.90	-50.24
SC50B.qlp	48	15	48	9.85	3	-63.76	-60.64	-56.72
SCAGR25.qlp	500	15	471	1.36	3	-14747833.31	-14743581.98	-14735157.64
SCAGR7.qlp	140	15	129	4.50	3	-2330910.15	-2330616.91	-2330339.21
SCTAP1.qlp	480	15	300	1.90	3	1917.74	3222.43	3890.89

Table A.17. NETLIB: EURO13 ($x_\forall = 15, S = 6$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	15	27	13.63	6	-399.54	-387.72	-372.36
KB2.qlp	41	15	43	18.11	6	-1748.71	×	-1747.42
RECIPELP.qlp	180	15	91	5.60	6	-266.62	-266.62	-266.62
SC105.qlp	103	15	104	5.46	6	-28.33	-12.42	-12.42
SC205.qlp	203	15	204	2.96	6	-50.30	-33.01	-33.01
SC50A.qlp	48	15	49	10.04	6	-59.75	-53.55	-50.11
SC50B.qlp	48	15	48	9.85	6	-63.53	-60.03	-55.58
SCAGR25.qlp	500	15	471	1.36	6	-14740206.01	-14734603.06	-14724728.97
SCAGR7.qlp	140	15	129	4.50	6	-2331067.85	-2330712.00	-2330464.38
SCTAP1.qlp	480	15	300	1.90	6	1999.68	3399.77	3978.51

Table A.18. NETLIB: EURO13 ($x_\forall = 18, S = 2$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	18	27	14.15	2	-452.96	-441.66	-437.47
KB2.qlp	41	18	43	18.37	2	-1749.79	-1749.63	-1749.54
RECIPELP.qlp	180	18	91	5.88	2	-266.62	-266.62	-266.62
SC105.qlp	103	18	104	5.94	2	-51.32	-49.36	-48.84
SC205.qlp	203	18	204	3.26	2	-50.85	-48.64	-48.55
SC50A.qlp	48	18	49	10.70	2	-63.70	-62.52	-61.58
SC50B.qlp	48	18	48	10.54	2	-68.59	-67.36	-66.11
SCAGR25.qlp	500	18	471	1.50	2	-14751672.85	-14750731.74	-14749142.59
SCAGR7.qlp	140	18	129	4.89	2	-2331237.18	-2331147.77	-2331112.10
SCTAP1.qlp	480	18	300	2.04	2	1502.82	1766.46	1867.01

Table A.19. NETLIB: EURO13 ($x_\forall = 18, S = 3$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	18	27	14.15	3	-448.85	-444.93	-439.45
KB2.qlp	41	18	43	18.37	3	-1749.37	-1749.24	-1749.17
RECIPELP.qlp	180	18	91	5.88	3	-266.62	-266.62	-266.62
SC105.qlp	103	18	104	5.94	3	-51.28	-49.66	-49.15
SC205.qlp	203	18	204	3.26	3	-50.86	-48.86	-48.75
SC50A.qlp	48	18	49	10.70	3	-63.29	-61.97	-61.00
SC50B.qlp	48	18	48	10.54	3	-69.18	-67.96	-67.23
SCAGR25.qlp	500	18	471	1.50	3	-14751890.62	-14750876.72	-14749487.70
SCAGR7.qlp	140	18	129	4.89	3	-2331293.12	-2331218.66	-2331154.22
SCTAP1.qlp	480	18	300	2.04	3	1487.30	1760.62	1895.01

Table A.20. NETLIB: EURO13 ($x_\forall = 18, S = 6$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	18	27	14.15	6	-447.72	-438.98	-435.76
KB2.qlp	41	18	43	18.37	6	-1749.61	-1749.50	-1749.38
RECIPELP.qlp	180	18	91	5.88	6	-266.62	-266.62	-266.62
SC105.qlp	103	18	104	5.94	6	-50.97	-49.31	-48.84
SC205.qlp	203	18	204	3.26	6	-51.61	-49.83	-49.55
SC50A.qlp	48	18	49	10.70	6	-63.39	-62.32	-61.40
SC50B.qlp	48	18	48	10.54	6	-67.76	-66.79	-65.87
SCAGR25.qlp	500	18	471	1.50	6	-14751276.75	-14750250.15	-14749085.58
SCAGR7.qlp	140	18	129	4.89	6	-2331322.57	-2331256.63	-2331184.94
SCTAP1.qlp	480	18	300	2.04	6	1497.87	1714.64	1857.36

A.2.3. NETLIB: TwoStage

Table A.21. NETLIB: TwoStage ($x_\forall = 4, S = 2$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	4	56	6.84	2	225914.64	226089.40	×
AFIRO.qlp	32	4	27	8.95	2	-463.90	-463.90	×
AGG2.qlp	302	4	516	2.75	2	-20239252.36	-20238985.38	×
AGG3.qlp	302	4	516	2.76	2	10312234.77	10312473.87	×
ISRAEL.qlp	142	4	174	8.99	2	-896638.40	-896638.08	-896638.08
KB2.qlp	41	4	43	15.04	2	-1744.59	-1737.17	×
SC105.qlp	103	4	104	2.64	2	-52.20	-52.18	×
SC205.qlp	203	4	204	1.36	2	-52.19	-52.19	×
SC50A.qlp	48	4	49	5.26	2	-64.51	-64.51	×
SC50B.qlp	48	4	48	4.93	2	-69.97	-69.97	×
SCAGR25.qlp	500	4	471	0.68	2	-14752396.82	-14752396.64	×
SCAGR7.qlp	140	4	129	2.35	2	-2331389.82	-2331389.82	×
SCTAP1.qlp	480	4	300	1.19	2	1439.59	1442.70	×
SCTAP2.qlp	1880	4	1090	0.34	2	1747.30	1751.26	×
SHARE2B.qlp	79	4	96	8.87	2	-412.14	-412.14	×

Table A.22. NETLIB: TwoStage ($x_\forall = 6, S = 2$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	6	56	6.80	2	226857.75	227041.05	×
AFIRO.qlp	32	6	27	8.67	2	-463.68	-463.68	-463.68
AGG2.qlp	302	6	516	2.76	2	-20239125.11	-20239007.19	×
AGG3.qlp	302	6	516	2.76	2	10312127.14	10312137.72	×
ISRAEL.qlp	142	6	174	8.95	2	-896604.69	-896604.66	-896604.53
KB2.qlp	41	6	43	14.50	2	-1749.90	-1749.90	×
SC105.qlp	103	6	104	2.67	2	-52.08	-52.04	×
SC205.qlp	203	6	204	1.38	2	-52.09	-52.09	×
SC50A.qlp	48	6	49	5.18	2	-64.58	-64.55	×
SC50B.qlp	48	6	48	4.82	2	-70.00	-69.97	×
SCAGR25.qlp	500	6	471	0.69	2	-14752789.58	-14752789.36	×
SCAGR7.qlp	140	6	129	2.35	2	-2329042.30	-2329042.30	×
SCTAP1.qlp	480	6	300	1.20	2	1509.94	1516.87	×
SCTAP2.qlp	1880	6	1090	0.34	2	1823.10	1834.13	×
SHARE2B.qlp	79	6	96	8.70	2	-410.11	-410.11	-410.11

Table A.23. NETLIB: TwoStage ($x_\forall = 8, S = 2$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	8	56	6.72	2	226238.87	226268.15	×
AFIRO.qlp	32	8	27	8.43	2	-464.56	-464.56	×
AGG2.qlp	302	8	516	2.77	2	-20239189.30	-20239016.54	×
AGG3.qlp	302	8	516	2.78	2	10312243.89	10312607.76	×
ISRAEL.qlp	142	8	174	8.85	2	-896493.47	-896493.44	-896493.36
KB2.qlp	41	8	43	14.05	2	-1749.81	-1749.81	×
SC105.qlp	103	8	104	2.62	2	-52.10	-52.06	×
SC205.qlp	203	8	204	1.42	2	-52.02	-51.98	×
SC50A.qlp	48	8	49	5.07	2	-64.54	-64.54	×
SC50B.qlp	48	8	48	4.72	2	-69.95	-69.95	×
SCAGR25.qlp	500	8	471	0.70	2	-14752015.59	-14751878.13	×
SCAGR7.qlp	140	8	129	2.38	2	-2331378.30	-2331379.03	×
SCTAP1.qlp	480	8	300	1.21	2	1425.65	1430.61	×
SCTAP2.qlp	1880	8	1090	0.34	2	1757.62	1790.41	×
SHARE2B.qlp	79	8	96	8.64	2	-388.49	-387.27	-387.27

Table A.24. NETLIB: TwoStage ($x_\forall = 10, S = 2$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	10	56	6.63	2	225937.65	226024.29	×
AFIRO.qlp	32	10	27	8.20	2	-464.46	-464.46	×
AGG2.qlp	302	10	516	2.76	2	-20239161.51	-20238970.17	×
AGG3.qlp	302	10	516	2.76	2	10312259.63	10312346.54	×
ISRAEL.qlp	142	10	174	8.76	2	-896635.46	-896569.61	-896569.47
KB2.qlp	41	10	43	13.59	2	-1742.93	-1742.93	×
SC105.qlp	103	10	104	2.72	2	-52.03	-51.90	×
SC205.qlp	203	10	204	1.43	2	-52.10	-51.93	×
SC50A.qlp	48	10	49	5.14	2	-64.28	-64.25	×
SC50B.qlp	48	10	48	4.71	2	-69.78	-69.77	×
SCAGR25.qlp	500	10	471	0.71	2	-14751377.21	-14751122.51	×
SCAGR7.qlp	140	10	129	2.35	2	-2330539.65	-2330531.65	×
SCTAP1.qlp	480	10	300	1.22	2	1425.38	1436.03	×
SCTAP2.qlp	1880	10	1090	0.35	2	1787.95	1802.04	×
SHARE2B.qlp	79	10	96	8.37	2	-410.14	-409.80	×

Table A.25. NETLIB: TwoStage ($x_\forall = 12, S = 2$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	12	56	6.52	2	227662.20	227671.42	×
AFIRO.qlp	32	12	27	8.00	2	-461.94	-461.94	-461.94
AGG2.qlp	302	12	516	2.77	2	-20238948.42	-20238638.04	×
AGG3.qlp	302	12	516	2.78	2	10312424.69	10312530.73	×
ISRAEL.qlp	142	12	174	8.72	2	-896424.07	-896390.38	-896390.33
KB2.qlp	41	12	43	13.21	2	-1749.90	-1749.90	×
SC105.qlp	103	12	104	2.71	2	-52.06	-52.02	×
SC205.qlp	203	12	204	1.45	2	-52.11	-52.06	×
SC50A.qlp	48	12	49	4.93	2	-64.47	-64.47	×
SC50B.qlp	48	12	48	4.58	2	-69.90	-69.88	×
SCAGR25.qlp	500	12	471	0.73	2	-14749843.40	-14749735.24	×
SCAGR7.qlp	140	12	129	2.38	2	-2330153.91	-2330145.32	×
SCTAP1.qlp	480	12	300	1.23	2	1467.25	1484.04	×
SCTAP2.qlp	1880	12	1090	0.35	2	1784.33	1817.66	×
SHARE2B.qlp	79	12	96	8.25	2	-412.43	-411.86	×

Table A.26. NETLIB: TwoStage ($x_\forall = 14, S = 2$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	14	56	6.48	2	228098.71	228538.92	×
AFIRO.qlp	32	14	27	7.81	2	-464.55	-464.55	×
AGG2.qlp	302	14	516	2.78	2	-20238439.04	-20238009.60	×
AGG3.qlp	302	14	516	2.77	2	10312331.54	10312552.67	×
ISRAEL.qlp	142	14	174	8.66	2	-896607.04	-896601.39	-896601.39
KB2.qlp	41	14	43	12.85	2	-1727.85	-1727.85	×
SC105.qlp	103	14	104	2.74	2	-51.78	-51.67	×
SC205.qlp	203	14	204	1.47	2	-52.01	-51.86	×
SC50A.qlp	48	14	49	4.87	2	-64.20	-64.15	×
SC50B.qlp	48	14	48	4.64	2	-69.82	-69.76	×
SCAGR25.qlp	500	14	471	0.73	2	-14750314.74	-14750231.46	×
SCAGR7.qlp	140	14	129	2.40	2	-2329997.54	-2329968.55	×
SCTAP1.qlp	480	14	300	1.24	2	1444.67	1465.96	×
SCTAP2.qlp	1880	14	1090	0.35	2	1845.67	1882.16	×
SHARE2B.qlp	79	14	96	8.09	2	-413.82	-413.79	-413.79

Table A.27. NETLIB: TwoStage ($x_\forall = 16, S = 2$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	16	56	6.45	2	229987.60	230087.38	×
AFIRO.qlp	32	16	27	7.64	2	-460.19	-460.19	×
AGG2.qlp	302	16	516	2.78	2	-20236283.66	-20236101.49	×
AGG3.qlp	302	16	516	2.79	2	10312376.38	10312507.92	×
ISRAEL.qlp	142	16	174	8.61	2	-896392.79	-896352.41	-896352.36
KB2.qlp	41	16	43	12.53	2	-1749.89	-1749.89	×
SC105.qlp	103	16	104	2.76	2	-51.83	-51.75	×
SC205.qlp	203	16	204	1.50	2	-52.01	-51.87	×
SC50A.qlp	48	16	49	4.72	2	-64.05	-64.05	×
SC50B.qlp	48	16	48	4.62	2	-69.65	-69.64	×
SCAGR25.qlp	500	16	471	0.74	2	-14750440.83	-14750267.79	×
SCAGR7.qlp	140	16	129	2.39	2	-2330998.41	-2330977.29	×
SCTAP1.qlp	480	16	300	1.26	2	1462.35	1483.63	×
SCTAP2.qlp	1880	16	1090	0.35	2	1845.74	1885.31	×
SHARE2B.qlp	79	16	96	8.06	2	-409.86	-409.41	×

Table A.28. NETLIB: TwoStage ($x_\forall = 18, S = 2$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	18	56	6.35	2	226870.49	227746.61	×
AFIRO.qlp	32	18	27	7.48	2	-460.42	-460.42	×
AGG2.qlp	302	18	516	2.79	2	-20239126.23	-20238630.38	×
AGG3.qlp	302	18	516	2.79	2	10312926.55	10313280.81	×
ISRAEL.qlp	142	18	174	8.54	2	-896473.23	-896461.66	-896461.21
KB2.qlp	41	18	43	12.22	2	-1745.59	-1745.59	×
SC105.qlp	103	18	104	2.73	2	-52.00	-51.89	×
SC205.qlp	203	18	204	1.52	2	-52.02	-51.81	×
SC50A.qlp	48	18	49	4.76	2	-64.21	-64.17	×
SC50B.qlp	48	18	48	4.58	2	-69.53	-69.43	×
SCAGR25.qlp	500	18	471	0.75	2	-14747813.99	-14747621.35	×
SCAGR7.qlp	140	18	129	2.40	2	-2329690.19	-2329682.80	×
SCTAP1.qlp	480	18	300	1.25	2	1762.82	1799.37	×
SCTAP2.qlp	1880	18	1090	0.36	2	1770.44	1787.21	×
SHARE2B.qlp	79	18	96	7.95	2	-408.70	-408.29	×

A.2.4. NETLIB: MultiStage

Table A.29. NETLIB: MultiStage ($x_\forall = 4, S = 3$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	4	56	6.88	3	225632.49	225632.49	×
AFIRO.qlp	32	4	27	8.95	3	-464.50	-464.50	×
AGG2.qlp	302	4	516	2.75	3	-20239234.48	-20239159.61	×
AGG3.qlp	302	4	516	2.77	3	10312161.32	10312164.85	×
ISRAEL.qlp	142	4	174	8.99	3	-896604.58	-896600.38	-896600.25
KB2.qlp	41	4	43	15.09	3	-1741.19	-1741.19	×
SC105.qlp	103	4	104	2.62	3	-52.18	-52.13	×
SC205.qlp	203	4	204	1.36	3	-52.19	-52.19	×
SC50A.qlp	48	4	49	5.26	3	-64.49	-64.49	×
SC50B.qlp	48	4	48	4.89	3	-69.90	-69.90	×
SCAGR25.qlp	500	4	471	0.68	3	-14753177.03	-14753157.98	×
SCAGR7.qlp	140	4	129	2.33	3	-2331389.81	-2331389.79	×
SCTAP1.qlp	480	4	300	1.19	3	1417.06	1418.13	×
SCTAP2.qlp	1880	4	1090	0.34	3	1738.75	1740.91	×
SHARE2B.qlp	79	4	96	8.80	3	-414.52	-414.52	×

Table A.30. NETLIB: MultiStage ($x_\forall = 6, S = 3$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	6	56	6.78	3	225567.37	225567.37	225567.37
AFIRO.qlp	32	6	27	8.67	3	-463.81	-463.81	×
AGG2.qlp	302	6	516	2.76	3	-20239221.21	-20238874.24	×
AGG3.qlp	302	6	516	2.77	3	10312503.78	10312923.14	×
ISRAEL.qlp	142	6	174	8.92	3	-896494.00	-896493.96	-896493.93
KB2.qlp	41	6	43	14.50	3	-1749.90	-1749.89	×
SC105.qlp	103	6	104	2.66	3	-52.18	-52.17	×
SC205.qlp	203	6	204	1.38	3	-52.17	-52.15	×
SC50A.qlp	48	6	49	5.18	3	-64.29	-64.29	×
SC50B.qlp	48	6	48	4.82	3	-69.80	-69.80	×
SCAGR25.qlp	500	6	471	0.70	3	-14752712.83	-14752697.09	×
SCAGR7.qlp	140	6	129	2.35	3	-2330741.49	-2330741.19	×
SCTAP1.qlp	480	6	300	1.21	3	1422.09	1429.07	×
SCTAP2.qlp	1880	6	1090	0.34	3	1744.47	1746.17	×
SHARE2B.qlp	79	6	96	8.75	3	-414.28	-411.84	×

Table A.31. NETLIB: MultiStage ($x_\forall = 8, S = 3$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	8	56	6.70	3	226413.28	226517.93	×
AFIRO.qlp	32	8	27	8.43	3	-463.53	-463.53	×
AGG2.qlp	302	8	516	2.75	3	-20239127.53	-20239002.32	×
AGG3.qlp	302	8	516	2.77	3	10312922.65	10313044.69	×
ISRAEL.qlp	142	8	174	8.85	3	-896343.45	-896343.39	-896343.38
KB2.qlp	41	8	43	14.14	3	-1749.90	-1749.89	×
SC105.qlp	103	8	104	2.66	3	-52.12	-52.09	×
SC205.qlp	203	8	204	1.40	3	-52.14	-52.08	×
SC50A.qlp	48	8	49	5.14	3	-64.52	-64.51	×
SC50B.qlp	48	8	48	4.80	3	-69.87	-69.87	×
SCAGR25.qlp	500	8	471	0.70	3	-14752370.79	-14752369.52	×
SCAGR7.qlp	140	8	129	2.39	3	-2329793.20	-2329779.92	×
SCTAP1.qlp	480	8	300	1.20	3	1434.50	1437.89	×
SCTAP2.qlp	1880	8	1090	0.34	3	1750.02	1755.10	×
SHARE2B.qlp	79	8	96	8.51	3	-413.54	-413.25	-413.25

Table A.32. NETLIB: MultiStage ($x_\forall = 10, S = 3$)

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	10	56	6.59	3	225956.88	226086.88	×
AFIRO.qlp	32	10	27	8.20	3	-463.96	-463.96	×
AGG2.qlp	302	10	516	2.77	3	-20239206.73	-20239200.73	×
AGG3.qlp	302	10	516	2.77	3	10312492.78	10313255.97	×
ISRAEL.qlp	142	10	174	8.79	3	-896636.49	-896618.85	-896618.83
KB2.qlp	41	10	43	13.68	3	-1749.90	-1749.88	×
SC105.qlp	103	10	104	2.70	3	-52.06	-52.05	×
SC205.qlp	203	10	204	1.39	3	-51.95	-51.90	×
SC50A.qlp	48	10	49	5.00	3	-64.29	-64.29	×
SC50B.qlp	48	10	48	4.67	3	-69.75	-69.75	×
SCAGR25.qlp	500	10	471	0.70	3	-14752793.45	-14752786.42	×
SCAGR7.qlp	140	10	129	2.39	3	-2331003.80	-2331001.03	×
SCTAP1.qlp	480	10	300	1.23	3	1423.55	1427.94	×
SCTAP2.qlp	1880	10	1090	0.34	3	1746.43	1756.93	×
SHARE2B.qlp	79	10	96	8.36	3	-412.97	-412.96	×

Table A.33. NETLIB: MultiStage ($x_\forall = 12, S = 3$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	12	56	6.55	3	226406.17	226557.48	×
AFIRO.qlp	32	12	27	8.00	3	-462.68	-462.68	×
AGG2.qlp	302	12	516	2.76	3	-20239029.66	-20238831.28	×
AGG3.qlp	302	12	516	2.78	3	10312167.55	10312337.84	×
ISRAEL.qlp	142	12	174	8.73	3	-896590.60	-896575.40	-896575.38
KB2.qlp	41	12	43	13.12	3	-1743.87	-3300	×
SC105.qlp	103	12	104	2.66	3	-52.13	-52.07	×
SC205.qlp	203	12	204	1.47	3	-52.11	-52.06	×
SC50A.qlp	48	12	49	4.97	3	-64.38	-64.38	×
SC50B.qlp	48	12	48	4.62	3	-69.92	-69.92	×
SCAGR25.qlp	500	12	471	0.72	3	-14752128.04	-14752103.62	×
SCAGR7.qlp	140	12	129	2.39	3	-2330700.24	-2330700.17	×
SCTAP1.qlp	480	12	300	1.22	3	1425.34	1426.62	×
SCTAP2.qlp	1880	12	1090	0.35	3	1749.79	1762.45	×
SHARE2B.qlp	79	12	96	8.24	3	-412.84	-411.60	×

Table A.34. NETLIB: MultiStage ($x_\forall = 14, S = 3$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	14	27	7.81	3	-464.05	-464.05	×
AGG2.qlp	302	14	516	2.77	3	-20239099.65	-20238595.96	×
AGG3.qlp	302	14	516	2.77	3	10312307.25	10312653.20	×
ISRAEL.qlp	142	14	174	8.62	3	-896627.88	-896627.83	-896627.82
KB2.qlp	41	14	43	12.77	3	-1749.84	-1749.84	×
SC105.qlp	103	14	104	2.67	3	-52.12	-52.08	×
SC205.qlp	203	14	204	1.50	3	-52.08	-52.02	×
SC50A.qlp	48	14	49	4.84	3	-63.99	-63.98	×
SC50B.qlp	48	14	48	4.50	3	-69.80	-69.80	×
SCAGR25.qlp	500	14	471	0.73	3	-14752107.59	-14752106.89	×
SCAGR7.qlp	140	14	129	2.37	3	-2330449.70	-2330445.01	×
SCTAP1.qlp	480	14	300	1.24	3	1453.82	1463.06	×
SCTAP2.qlp	1880	14	1090	0.35	3	1784.35	1796.26	×
SHARE2B.qlp	79	14	96	8.12	3	-395.33	-395.22	×

Table A.35. NETLIB: MultiStage ($x_\forall = 16, S = 3$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	16	56	6.46	3	226217.95	226276.62	×
AFIRO.qlp	32	16	27	7.64	3	-464.15	-464.15	×
AGG2.qlp	302	16	516	2.77	3	-20238975.59	-20238060.42	×
AGG3.qlp	302	16	516	2.80	3	10312300.61	10313182.49	×
ISRAEL.qlp	142	16	174	8.59	3	-896585.00	-896568.48	-896568.27
SC105.qlp	103	16	104	2.71	3	-51.94	-51.88	×
SC205.qlp	203	16	204	1.49	3	-52.14	-52.07	×
SC50A.qlp	48	16	49	4.85	3	-64.34	-64.33	×
SC50B.qlp	48	16	48	4.39	3	-69.39	-69.39	×
SCAGR25.qlp	500	16	471	0.74	3	-14751925.43	-14751892.53	×
SCAGR7.qlp	140	16	129	2.39	3	-2331037.45	-2331037.45	×
SCTAP1.qlp	480	16	300	1.25	3	1539.82	1548.25	×
SCTAP2.qlp	1880	16	1090	0.35	3	1765.49	1779.76	×
SHARE2B.qlp	79	16	96	8.02	3	-406.91	-406.82	×

Table A.36. NETLIB: MultiStage ($x_\forall = 4, S = 4$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	4	56	6.91	4	225663.69	225666.55	×
AFIRO.qlp	32	4	27	8.95	4	-463.71	-463.71	×
AGG2.qlp	302	4	516	2.76	4	-20239193.57	-20239130.85	×
AGG3.qlp	302	4	516	2.77	4	10312129.39	10312130.26	×
ISRAEL.qlp	142	4	174	9.03	4	-896639.34	-896639.25	-896639.25
KB2.qlp	41	4	43	15.09	4	-1749.88	-1749.88	-1749.88
SC105.qlp	103	4	104	2.62	4	-52.18	-52.16	×
SC205.qlp	203	4	204	1.38	4	-52.12	-52.11	×
SC50A.qlp	48	4	49	5.34	4	-64.58	-64.58	×
SC50B.qlp	48	4	48	4.97	4	-69.99	-69.99	×
SCAGR25.qlp	500	4	471	0.68	4	-14753177.84	-14753177.78	×
SCAGR7.qlp	140	4	129	2.32	4	-2331389.82	-2331389.46	×
SCTAP1.qlp	480	4	300	1.20	4	1531.74	1534.18	×
SCTAP2.qlp	1880	4	1090	0.33	4	1736.97	1739.11	×
SHARE2B.qlp	79	4	96	8.84	4	-415.61	-415.61	-415.61

Table A.37. NETLIB: MultiStage ($x_\forall = 6, S = 4$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	6	56	6.78	4	226391.00	226417.39	×
AFIRO.qlp	32	6	27	8.67	4	-464.59	-464.59	-464.59
AGG2.qlp	302	6	516	2.75	4	-20239201.41	-20239189.56	×
AGG3.qlp	302	6	516	2.78	4	10312280.52	10312457.95	×
ISRAEL.qlp	142	6	174	8.92	4	-896608.56	-896608.50	-896608.46
KB2.qlp	41	6	43	14.55	4	-1749.89	-1749.89	×
SC105.qlp	103	6	104	2.64	4	-52.15	-52.15	×
SC205.qlp	203	6	204	1.39	4	-52.11	-52.10	×
SC50A.qlp	48	6	49	5.18	4	-64.53	-64.53	×
SC50B.qlp	4H8	6	48	4.82	4	-69.93	-69.93	×
SCAGR25.qlp	500	6	471	0.70	4	-14752764.82	-14752732.85	×
SCAGR7.qlp	140	6	129	2.36	4	-2331107.89	-2331107.84	×
SCTAP1.qlp	480	6	300	1.19	4	1447.29	1448.79	×
SCTAP2.qlp	1880	6	1090	0.34	4	1742.36	1748.84	×
SHARE2B.qlp	79	6	96	8.64	4	-415.03	-415.03	-415.03

Table A.38. NETLIB: MultiStage ($x_\forall = 8, S = 4$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	8	56	6.70	4	226283.67	226345.05	×
AFIRO.qlp	32	8	27	8.43	4	-463.88	-463.88	-463.88
AGG2.qlp	302	8	516	2.77	4	-20238410.70	-20238064.07	×
AGG3.qlp	302	8	516	2.79	4	10312401.04	10312613.00	×
ISRAEL.qlp	142	8	174	8.87	4	-896631.82	-896631.70	-896631.67
KB2.qlp	41	8	43	14.05	4	-1746.26	-1746.11	×
SC105.qlp	103	8	104	2.68	4	-52.08	-52.07	×
SC205.qlp	203	8	204	1.42	4	-52.05	-52.04	×
SC50A.qlp	48	8	49	5.10	4	-64.55	-64.54	×
SC50B.qlp	48	8	48	4.80	4	-69.97	-69.96	×
SCAGR25.qlp	500	8	471	0.70	4	-14752786.99	-14752786.91	×
SCAGR7.qlp	140	8	129	2.38	4	-2330506.56	-2330506.55	×
SCTAP1.qlp	480	8	300	1.20	4	1421.09	1427.19	×
SCTAP2.qlp	1880	8	1090	0.34	4	1743.35	1757.90	×
SHARE2B.qlp	79	8	96	8.56	4	-413.42	-412.47	×

Table A.39. NETLIB: MultiStage ($x_\forall = 10, S = 4$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	10	56	6.66	4	225914.50	225957.40	×
AFIRO.qlp	32	10	27	8.20	4	-464.28	-464.28	×
AGG2.qlp	302	10	516	2.78	4	-20239179.15	-20239017.78	×
AGG3.qlp	302	10	516	2.78	4	10312282.64	10312620.68	×
ISRAEL.qlp	142	10	174	8.81	4	-896305.02	-896304.84	-896298.81
KB2.qlp	41	10	43	13.63	4	-1749.89	-1749.89	×
SC105.qlp	103	10	104	2.71	4	-52.12	-52.12	×
SC205.qlp	203	10	204	1.42	4	-52.07	-52.05	×
SC50A.qlp	48	10	49	5.03	4	-64.37	-64.37	×
SC50B.qlp	48	10	48	4.74	4	-69.91	-69.91	×
SCAGR25.qlp	500	10	471	0.71	4	-14752523.66	-14752506.93	×
SCAGR7.qlp	140	10	129	2.37	4	-2330618.85	-2330618.80	×
SCTAP1.qlp	480	10	300	1.23	4	1419.92	1423.20	×
SCTAP2.qlp	1880	10	1090	0.34	4	1746.57	1754.64	×
SHARE2B.qlp	79	10	96	8.33	4	-414.63	-413.23	×

Table A.40. NETLIB: MultiStage ($x_\forall = 12, S = 4$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	12	56	6.72	4	226699.41	227070.85	×
AFIRO.qlp	32	12	27	8.00	4	-464.50	-464.50	×
AGG2.qlp	302	12	516	2.76	4	-20238864.00	-20238287.97	×
AGG3.qlp	302	12	516	2.79	4	10312192.33	10312480.55	×
ISRAEL.qlp	142	12	174	8.74	4	-896468.23	-896427.21	-896427.04
KB2.qlp	41	12	43	13.73	4	-1726.16	-1725.11	×
SC105.qlp	103	12	104	2.71	4	-52.04	-52.01	×
SC205.qlp	203	12	204	1.44	4	-52.07	-52.03	×
SC50A.qlp	48	12	49	4.93	4	-64.27	-64.27	×
SC50B.qlp	48	12	48	4.69	4	-69.81	-69.80	×
SCAGR25.qlp	500	12	471	0.72	4	-14752623.93	-14752582.23	×
SCAGR7.qlp	140	12	129	2.40	4	-2330122.31	-2330122.31	×
SCTAP1.qlp	480	12	300	1.23	4	1485.36	1488.70	×
SCTAP2.qlp	1880	12	1090	0.34	4	1755.30	1765.22	×
SHARE2B.qlp	79	12	96	8.29	4	-409.14	-409.08	×

Table A.41. NETLIB: MultiStage ($x_\forall = 14, S = 4$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	14	56	6.50	4	226211.73	226282.65	×
AFIRO.qlp	32	14	27	7.81	4	-462.11	-462.11	×
AGG2.qlp	302	14	516	2.77	4	-20239167.34	-20238553.65	×
AGG3.qlp	302	14	516	2.79	4	10312284.59	10312609.44	×
ISRAEL.qlp	142	14	174	8.63	4	-896626.64	-896623.08	-896623.08
SC105.qlp	103	14	104	2.71	4	-52.00	-51.95	×
SC205.qlp	203	14	204	1.45	4	-52.00	-51.97	×
SC50A.qlp	48	14	49	4.77	4	-64.43	-64.43	×
SC50B.qlp	48	14	48	4.54	4	-69.91	-69.89	×
SCAGR25.qlp	500	14	471	0.73	4	-14751910.05	-14751792.33	×
SCAGR7.qlp	140	14	129	2.38	4	-2330875.76	-2330875.63	×
SCTAP1.qlp	480	14	300	1.23	4	1440.55	1443.89	×
SCTAP2.qlp	1880	14	1090	0.35	4	1755.49	1768.38	×
SHARE2B.qlp	79	14	96	8.18	4	-414.05	-412.48	×

Table A.42. NETLIB: MultiStage ($x_\forall = 16, S = 4$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	16	56	6.42	4	226846.31	2277809.76	×
AFIRO.qlp	32	16	27	7.64	4	-463.76	-463.76	×
AGG2.qlp	302	16	516	2.78	4	-20238065.46	-20237685.00	×
AGG3.qlp	302	16	516	2.79	4	10312295.57	10312956.59	×
ISRAEL.qlp	142	16	174	8.61	4	-896569.73	-896555.78	-896555.55
SC105.qlp	103	16	104	2.69	4	-52.03	-52.00	×
SC205.qlp	203	16	204	1.47	4	-51.94	-51.90	×
SC50A.qlp	48	16	49	4.78	4	-64.29	-64.28	×
SC50B.qlp	48	16	48	4.59	4	-69.57	-69.56	×
SCAGR25.qlp	500	16	471	0.74	4	-14751174.80	-14751054.17	×
SCAGR7.qlp	140	16	129	2.39	4	-2329405.27	-2329405.22	×
SCTAP1.qlp	480	16	300	1.24	4	1428.86	1436.20	×
SCTAP2.qlp	1880	16	1090	0.35	4	1778.86	1796.41	×
SHARE2B.qlp	79	16	96	8.04	4	-412.24	-411.78	×

Table A.43. NETLIB: MultiStage ($x_\forall = 4, S = 5$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	4	56	6.90	5	226846.59	226846.59	×
AFIRO.qlp	32	4	27	8.95	5	-464.28	-464.28	×
AGG2.qlp	302	4	516	2.76	5	-20239251.76	-20239076.79	×
AGG3.qlp	302	4	516	2.77	5	10312128.56	10312272.34	×
ISRAEL.qlp	142	4	174	9.01	5	-896644.35	-896644.34	-896637.55
KB2.qlp	41	4	43	15.09	5	-1749.90	-1749.90	×
SC105.qlp	103	4	104	2.66	5	-52.19	-52.18	×
SC205.qlp	203	4	204	1.38	5	-52.10	-52.06	×
SC50A.qlp	48	4	49	5.30	5	-64.56	-64.56	-64.56
SC50B.qlp	48	4	48	4.93	5	-69.99	-69.99	-69.99
SCAGR25.qlp	500	4	471	0.68	5	-14753186.76	-14753186.67	×
SCAGR7.qlp	140	4	129	2.34	5	-2331388.33	-2331388.28	×
SCTAP1.qlp	480	4	300	1.19	5	1416.74	1417.51	×
SCTAP2.qlp	1880	4	1090	0.33	5	1734.64	1735.20	×
SHARE2B.qlp	79	4	96	8.82	5	-414.60	-414.60	-414.24

Table A.44. NETLIB: MultiStage ($x_\forall = 6, S = 5$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	6	56	6.81	5	225494.96	225582.62	×
AFIRO.qlp	32	6	27	8.67	5	-464.41	-464.41	×
AGG2.qlp	302	6	516	2.76	5	-20239252.26	-20239069.06	×
AGG3.qlp	302	6	516	2.77	5	10312122.44	10312469.05	×
ISRAEL.qlp	142	6	174	8.94	5	-896640.68	-896640.68	-896640.65
KB2.qlp	41	6	43	14.60	5	-1746.80	-1746.80	×
SC105.qlp	103	6	104	2.64	5	-52.16	-52.12	×
SC205.qlp	203	6	204	1.37	5	-52.13	-52.12	×
SC50A.qlp	48	6	49	5.22	5	-64.41	-64.40	×
SC50B.qlp	48	6	48	4.90	5	-69.95	-69.93	×
SCAGR25.qlp	500	6	471	0.70	5	-14753217.05	-14753216.45	×
SCAGR7.qlp	140	6	129	2.39	5	-2329334.44	-2329334.37	×
SCTAP1.qlp	480	6	300	1.21	5	1440.31	1442.07	×
SCTAP2.qlp	1880	6	1090	0.34	5	1740.91	1744.41	×
SHARE2B.qlp	79	6	96	8.68	5	-414.64	-414.64	×

Table A.45. NETLIB: MultiStage ($x_\forall = 8, S = 5$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	8	56	6.73	5	226274.56	226436.72	×
AFIRO.qlp	32	8	27	8.43	5	-464.14	-464.14	-464.14
AGG2.qlp	302	8	516	2.76	5	-20237970.21	-20237569.25	×
AGG3.qlp	302	8	516	2.78	5	10312144.79	10312331.38	×
ISRAEL.qlp	142	8	174	8.90	5	-896641.66	-896641.65	-896641.64
KB2.qlp	41	8	43	14.14	5	-1749.90	-1749.90	×
SC105.qlp	103	8	104	2.72	5	-52.18	-52.15	×
SC205.qlp	203	8	204	1.40	5	-52.15	-52.13	×
SC50A.qlp	48	8	49	5.10	5	-64.48	-64.48	×
SC50B.qlp	48	8	48	4.84	5	-69.78	-69.77	×
SCAGR25.qlp	500	8	471	0.71	5	-14752045.31	-14752042.53	×
SCAGR7.qlp	140	8	129	2.39	5	-2331293.76	-2331289.89	×
SCTAP1.qlp	480	8	300	1.21	5	1424.52	1427.35	×
SCTAP2.qlp	1880	8	1090	0.34	5	1752.64	1760.45	×
SHARE2B.qlp	79	8	96	8.60	5	-415.33	-415.20	-415.20

Table A.46. NETLIB: MultiStage ($x_\forall = 10, S = 5$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	10	56	6.66	5	226113.44	226375.90	×
AFIRO.qlp	32	10	27	8.20	5	-464.63	-464.63	×
AGG2.qlp	302	10	516	2.76	5	-20238916.27	-20238438.00	×
AGG3.qlp	302	10	516	2.78	5	10312203.73	10312353.20	×
ISRAEL.qlp	142	10	174	8.76	5	-896474.17	-896474.10	-896469.97
KB2.qlp	41	10	43	13.59	5	-1744.57	-1744.57	×
SC105.qlp	103	10	104	2.70	5	-52.18	-52.15	×
SC205.qlp	203	10	204	1.46	5	-52.09	-52.03	×
SC50A.qlp	48	10	49	5.00	5	-64.43	-64.43	×
SC50B.qlp	48	10	48	4.63	5	-69.94	-69.94	×
SCAGR25.qlp	500	10	471	0.71	5	-14752406.70	-14752388.02	×
SCAGR7.qlp	140	10	129	2.37	5	-2331104.54	-2331098.79	×
SCTAP1.qlp	480	10	300	1.21	5	1434.76	1441.55	×
SCTAP2.qlp	1880	10	1090	0.35	5	1760.09	1770.20	×
SHARE2B.qlp	79	10	96	8.44	5	-412.37	-411.59	×

Table A.47. NETLIB: MultiStage ($x_\forall = 12, S = 5$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
ADLITTLE.qlp	97	12	56	6.59	5	227266.84	230132.67	×
AFIRO.qlp	32	12	27	8.00	5	-463.70	-463.70	×
AGG2.qlp	302	12	516	2.78	5	-20239196.83	-20238687.01	×
AGG3.qlp	302	12	516	2.79	5	10312229.59	10312526.75	×
ISRAEL.qlp	142	12	174	8.74	5	-896596.39	-896594.59	-896593.91
KB2.qlp	41	12	43	13.16	5	-1749.90	-1749.37	×
SC105.qlp	103	12	104	2.69	5	-52.08	-52.07	×
SC205.qlp	203	12	204	1.42	5	-52.13	-52.11	×
SC50A.qlp	48	12	49	4.97	5	-64.41	-64.41	×
SC50B.qlp	48	12	48	4.69	5	-69.74	-69.74	×
SCAGR25.qlp	500	12	471	0.73	5	-14752293.33	-14752276.97	×
SCAGR7.qlp	140	12	129	2.39	5	-2330155.58	-2330154.60	×
SCTAP1.qlp	480	12	300	1.24	5	1641.04	1646.38	×
SCTAP2.qlp	1880	12	1090	0.35	5	1768.50	1775.91	×
SHARE2B.qlp	79	12	96	8.29	5	-405.29	-400.43	×

Table A.48. NETLIB: MultiStage ($x_\forall = 14, S = 5$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	14	27	7.81	5	-462.65	-462.65	×
AGG2.qlp	302	14	516	2.77	5	-20239100.69	-20238101.93	×
AGG3.qlp	302	14	516	2.80	5	10312627.61	10313062.66	×
ISRAEL.qlp	142	14	174	8.64	5	-896494.62	-896490.13	-896490.05
KB2.qlp	41	14	43	12.85	5	-1747.62	-1747.61	×
SC105.qlp	103	14	104	2.73	5	-51.97	-51.91	×
SC205.qlp	203	14	204	1.48	5	-52.07	-52.05	×
SC50A.qlp	48	14	49	4.90	5	-64.53	-64.53	×
SC50B.qlp	48	14	48	4.54	5	-69.81	-69.80	×
SCAGR25.qlp	500	14	471	0.74	5	-14751608.99	-14751560.38	×
SCAGR7.qlp	140	14	129	2.40	5	-2331156.92	-2331156.49	×
SCTAP1.qlp	480	14	300	1.22	5	1480.00	1486.08	×
SCTAP2.qlp	1880	14	1090	0.35	5	1761.30	1778.78	×
SHARE2B.qlp	79	14	96	8.20	5	-414.86	-413.42	×

Table A.49. NETLIB: MultiStage ($x_\forall = 16, S = 5$)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
AFIRO.qlp	32	16	27	7.64	5	-463.74	-463.74	×
AGG2.qlp	302	16	516	2.79	5	-20237218.83	-20236392.01	×
AGG3.qlp	302	16	516	2.79	5	10312413.00	10313020.12	×
ISRAEL.qlp	142	16	174	8.57	5	-896644.58	-896644.54	-896644.53
KB2.qlp	41	16	43	12.48	5	-1749.54	-1749.54	×
SC105.qlp	103	16	104	2.72	5	-51.99	-51.92	×
SC205.qlp	203	16	204	1.47	5	-52.01	-51.99	×
SC50A.qlp	48	16	49	4.91	5	-64.31	-64.29	×
SC50B.qlp	48	16	48	4.59	5	-69.62	-69.60	×
SCAGR25.qlp	500	16	471	0.75	5	-14752293.98	-14752259.23	×
SCAGR7.qlp	140	16	129	2.42	5	-2330673.11	-2330673.10	×
SCTAP1.qlp	480	16	300	1.24	5	1569.48	1576.10	×
SCTAP2.qlp	1880	16	1090	0.36	5	1768.17	1786.37	×
SHARE2B.qlp	79	16	96	7.94	5	-391.55	-30200096.00	×

A.3. QBFLIB QLP Otimization Problems

Table A.50. QBFLIB: cnt Min-QSAT

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
cnt11e	840	11	2256	0.35	12	14	24.11	×
cnt11re	875	11	2355	0.34	12	14	24.11	×
cnt11	805	11	2146	0.36	12	26.50	48	×
cnt11r	840	11	2245	0.35	12	16	28.76	×
cnt12	950	12	2533	0.31	13	28	TODO	×
cnt10	672	10	1791	0.43	11	25	53	×
cnt12e	988	12	2653	0.30	13	15	25.22	×
cnt10re	736	10	1981	0.40	11	13	23.01	×
cnt10r	704	10	1881	0.42	11	15	27.31	×
cnt12r	988	12	2641	0.30	13	17	30.22	×
cnt10e	704	10	1891	0.42	11	13	23.01	×

Table A.51. QBFLIB: impl Min-QSAT

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
impl02.qdimacs.qip	8	2	18	28.33	3	6	10	10
impl04.qdimacs.qip	14	4	34	16.18	5	9	18	18
impl06.qdimacs.qip	20	6	50	11.31	7	12	26	26
impl08.qdimacs.qip	26	8	66	8.69	9	15	34	34
impl10.qdimacs.qip	32	10	82	7.06	11	18	42	42
impl12.qdimacs.qip	38	12	98	5.94	13	21	50	50
impl14.qdimacs.qip	44	14	114	5.13	15	24	58	58
impl16.qdimacs.qip	50	16	130	4.51	17	27	66	66
impl18.qdimacs.qip	56	18	146	4.03	19	30	74	74
impl20.qdimacs.qip	62	20	162	3.64	21	33	82	82

Table A.52. QBFLIB: Toilet Min-QSAT

Name	$\lvert x_\exists \rvert$	$\lvert x_\forall \rvert$	$\lvert C \rvert$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
TOILET2.1.iv.3	27	1	70	8.42	2	8.50	10.50	×
TOILET2.1.iv.4	36	1	99	6.50	2	10	12	×
TOILET6.1.iv.11	291	3	1046	0.80	2	21.20	32.22	×
TOILET6.1.iv.12	318	3	1144	0.73	2	22.40	33.64	×
TOILET7.1.iv.13	396	3	1491	0.59	2	23.60	37.28	×
TOILET7.1.iv.14	427	3	1608	0.55	2	24.80	38.71	×
TOILET10.1.iv.20	850	4	3588	0.28	2	32.83	53.96	×
TOILET16.1.iv.32	2128	4	10734	0.11	2	47.50	84.45	×

Table A.53. QBFLIB: Toilet_g Min-QSAT

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
toilet_g_02_01.2	10	1	22	18.18	2	4	6	×
toilet_g_04_01.2	20	2	52	10.49	2	5	11	×
toilet_g_06_01.2	30	3	90	8.48	2	5	15	×
toilet_g_08_01.2	40	3	136	6.84	2	6	20	×
toilet_g_10_01.2	50	4	190	6.63	2	6	24	×
toilet_g_15_01.2	75	4	360	4.91	2	6	34	×
toilet_g_20_01.2	100	5	580	4.53	2	7	45	×

Table A.54. QBFLIB: Toilet_a Min-QSAT (1)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
toilet_a_02_01.2.	16	2	39	12.25	2	×	×	×
toilet_a_02_01.3.	24	2	64	8.59	2	9	9	×
toilet_a_02_01.4.	32	2	89	6.61	2	10	10	×
toilet_a_02_05.2.	48	2	163	4.39	2	6.40	6.40	×
toilet_a_02_10.2.	88	2	408	2.39	2	6.20	6.20	×
toilet_a_04_01.2.	28	4	129	10.71	2	×	×	×
toilet_a_04_01.3.	42	4	179	6.74	2	18	18	×
toilet_a_04_01.4.	56	4	229	4.86	2	20	20	×
toilet_a_04_01.5.	70	4	279	3.78	2	21.67	21.67	×
toilet_a_04_01.6.	84	4	329	3.09	2	23	23	×
toilet_a_04_01.7.	98	4	379	2.60	2	24.25	24.25	×
toilet_a_04_01.8.	112	4	429	2.25	2	25.25	25.25	×
toilet_a_04_05.2.	76	4	389	3.25	2	12.80	12.80	×
toilet_a_04_10.2.	136	4	894	1.66	2	12.40	12.40	×
toilet_a_06_01.10.	200	6	1123	1.86	2	40.75	40.75	×
toilet_a_06_01.11.	220	6	1202	1.65	2	42.20	42.20	×
toilet_a_06_01.12.	240	6	1281	1.47	2	43.40	43.40	×
toilet_a_06_01.2.	40	6	491	12.74	2	×	×	×
toilet_a_06_01.3.	60	6	570	8.12	2	27	27	×
toilet_a_06_01.4.	80	6	649	5.79	2	30	30	×
toilet_a_06_01.5.	100	6	728	4.42	2	32.33	32.33	×
toilet_a_06_01.6.	120	6	807	3.53	2	34.33	34.33	×
toilet_a_06_01.7.	140	6	886	2.91	2	36.25	36.25	×
toilet_a_06_01.8.	160	6	965	2.46	2	37.75	37.75	×
toilet_a_06_01.9.	180	6	1044	2.12	2	39.25	39.25	×
toilet_a_06_05.2.	104	6	919	3.76	2	19.20	19.20	×
toilet_a_06_05.3.	156	6	1278	2.23	2	21.40	21.40	×
toilet_a_06_05.4.	208	6	1637	1.55	2	22.60	22.60	×
toilet_a_06_10.2.	184	6	1724	1.68	2	18.60	18.60	×

Table A.55. QBFLIB: Toilet_a Min-QSAT (2)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
toilet_a_08_01.10.	260	8	3101	2.49	2	54.25	54.25	×
toilet_a_08_01.11.	286	8	3213	2.22	2	56.20	56.20	×
toilet_a_08_01.12.	312	8	3325	1.99	2	57.80	57.80	×
toilet_a_08_01.13.	338	8	3437	1.80	2	59.40	59.40	×
toilet_a_08_01.14.	364	8	3549	1.64	2	61	61.00	×
toilet_a_08_01.15.	390	8	3661	1.51	2	62.60	62.60	×
toilet_a_08_01.16.	416	8	3773	1.39	2	64.17	64.17	×
toilet_a_08_01.2.	52	8	2205	14.13	2	×	×	×
toilet_a_08_01.3.	78	8	2317	9.51	2	36	36	×
toilet_a_08_01.4.	104	8	2429	7.06	2	40	40	×
toilet_a_08_01.5.	130	8	2541	5.55	2	43	43.00	×
toilet_a_08_01.6.	156	8	2653	4.53	2	45.67	45.67	×
toilet_a_08_01.7.	182	8	2765	3.80	2	48.25	48.25	×
toilet_a_08_01.8.	208	8	2877	3.25	2	50.25	50.25	×
toilet_a_08_01.9.	234	8	2989	2.83	2	52.25	52.25	×
toilet_a_08_05.10.	660	8	6929	0.63	2	35.70	35.70	×
toilet_a_08_05.2.	132	8	2833	5.06	2	25.60	25.60	×
toilet_a_08_05.3.	198	8	3345	3.08	2	28.20	28.20	×
toilet_a_08_05.4.	264	8	3857	2.13	2	29.80	29.80	×
toilet_a_08_05.5.	330	8	4369	1.59	2	31.40	31.40	×
toilet_a_08_05.6.	396	8	4881	1.25	2	32.50	32.50	×
toilet_a_08_05.7.	462	8	5393	1.02	2	33.30	33.30	×
toilet_a_08_05.8.	528	8	5905	0.85	2	34.10	34.10	×
toilet_a_08_05.9.	594	8	6417	0.73	2	34.90	34.90	×
toilet_a_08_10.2.	232	8	3978	2.35	2	24.80	24.80	×
toilet_a_10_01.10.	320	10	11647	3.01	2	67.75	67.75	×
toilet_a_10_01.11.	352	10	11796	2.72	2	70.20	70.20	×
toilet_a_10_01.12.	384	10	11945	2.47	2	72.20	72.20	×
toilet_a_10_01.13.	416	10	12094	2.26	2	74.20	74.20	×
toilet_a_10_01.14.	448	10	12243	2.09	2	76.20	76.20	×
toilet_a_10_01.15.	480	10	12392	1.93	2	78.20	78.20	×
toilet_a_10_01.16.	512	10	12541	1.80	2	80.17	80.17	×
toilet_a_10_01.17.	544	10	12690	1.68	2	81.83	81.83	×
toilet_a_10_01.18.	576	10	12839	1.57	2	83.50	83.50	×
toilet_a_10_01.19.	608	10	12988	1.48	2	85.17	85.17	×
toilet_a_10_01.2.	64	10	10455	14.60	2	×	×	×
toilet_a_10_01.20.	640	10	13137	1.39	2	86.83	86.83	×
toilet_a_10_01.3.	96	10	10604	10.08	2	45	45	×
toilet_a_10_01.4.	128	10	10753	7.66	2	50	50	×
toilet_a_10_01.5.	160	10	10902	6.15	2	53.67	53.67	×
toilet_a_10_01.6.	192	10	11051	5.12	2	57	57.00	×
toilet_a_10_01.7.	224	10	11200	4.37	2	60.25	60.25	×
toilet_a_10_01.8.	256	10	11349	3.81	2	62.75	62.75	×
toilet_a_10_01.9.	288	10	11498	3.36	2	65.25	65.25	×
toilet_a_10_05.2.	160	10	11315	5.97	2	32	32	×
toilet_a_10_05.3.	240	10	12000	3.88	2	35	35	×
toilet_a_10_05.4.	320	10	12685	2.82	2	37	37.00	×
toilet_a_10_10.2.	280	10	12840	3.17	2	31	31.00	×

Table A.56. QBFLIB: Toilet_c Min-QSAT (1)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
toilet_c_02_01.2.	16	1	37	12.08	2	5	×	×
toilet_c_02_01.3.	24	1	62	8.58	2	5.50	7.50	×
toilet_c_02_01.4.	32	1	87	6.62	2	6	8	×
toilet_c_02_05.2.	48	1	161	4.41	2	4.20	5.40	×
toilet_c_02_10.2.	88	1	406	2.40	2	4.10	5.20	×
toilet_c_04_01.2.	28	2	85	7.45	2	6	×	×
toilet_c_04_01.3.	42	2	135	5.10	2	6.50	12.50	×
toilet_c_04_01.4.	56	2	185	3.88	2	7	13	×
toilet_c_04_01.5.	70	2	235	3.13	2	7.50	13.50	×
toilet_c_04_01.6.	84	2	285	2.62	2	8	14	×
toilet_c_04_01.7.	98	2	335	2.25	2	8.25	14.38	×
toilet_c_04_01.8.	112	2	385	1.98	2	8.50	14.73	×
toilet_c_04_05.2.	76	2	345	2.82	2	5.20	8.80	×
toilet_c_04_10.2.	136	2	850	1.56	2	5.10	8.40	×
toilet_c_06_01.10.	200	3	781	1.13	2	9	19.82	×
toilet_c_06_01.11.	220	3	860	1.03	2	9.20	20.22	×
toilet_c_06_01.12.	240	3	939	0.94	2	9.40	20.64	×
toilet_c_06_01.2.	40	3	149	5.87	2	6	×	×
toilet_c_06_01.3.	60	3	228	3.85	2	6.50	16.50	×
toilet_c_06_01.4.	80	3	307	2.86	2	7	17	×
toilet_c_06_01.5.	100	3	386	2.28	2	7.50	17.50	×
toilet_c_06_01.6.	120	3	465	1.90	2	8	18	×
toilet_c_06_01.7.	140	3	544	1.62	2	8.25	18.50	×
toilet_c_06_01.8.	160	3	623	1.42	2	8.50	19	×
toilet_c_06_01.9.	180	3	702	1.26	2	8.75	19.42	×
toilet_c_06_05.2.	104	3	577	2.11	2	5.20	11.20	×
toilet_c_06_05.3.	156	3	936	1.42	2	5.70	11.70	×
toilet_c_06_05.4.	208	3	1295	1.07	2	6.20	12.20	×
toilet_c_06_10.2.	184	3	1382	1.16	2	5.10	10.60	×
toilet_c_08_01.10.	260	3	1125	0.87	2	10	24.53	×
toilet_c_08_01.11.	286	3	1237	0.79	2	10.20	24.96	×
toilet_c_08_01.12.	312	3	1349	0.73	2	10.40	25.41	×
toilet_c_08_01.13.	338	3	1461	0.67	2	10.60	25.87	×
toilet_c_08_01.14.	364	3	1573	0.62	2	10.80	26.30	×
toilet_c_08_01.15.	390	3	1685	0.58	2	11	26.72	×
toilet_c_08_01.16.	416	3	1797	0.54	2	11.17	27.12	×
toilet_c_08_01.2	52	3	229	4.72	2	7	×	×
toilet_c_08_01.3	78	3	341	3.05	2	7.50	21.50	×
toilet_c_08_01.4	104	3	453	2.25	2	8	22	×
toilet_c_08_01.5	130	3	565	1.78	2	8.50	22.50	×
toilet_c_08_01.6	156	3	677	1.48	2	9	23	×
toilet_c_08_01.7	182	3	789	1.26	2	9.25	23.38	×
toilet_c_08_01.8	208	3	901	1.10	2	9.50	23.73	×
toilet_c_08_01.9	234	3	1013	0.97	2	9.75	24.11	×
toilet_c_08_05.10	660	3	4953	0.34	2	7.90	17.43	×
toilet_c_08_05.2	132	3	857	1.68	2	6.20	14.60	×
toilet_c_08_05.3	198	3	1369	1.12	2	6.70	15.10	×
toilet_c_08_05.4	264	3	1881	0.85	2	7.20	15.60	×
toilet_c_08_05.5	330	3	2393	0.68	2	7.40	15.95	×
toilet_c_08_05.6	396	3	2905	0.56	2	7.50	16.24	×
toilet_c_08_05.7	462	3	3417	0.48	2	7.60	16.56	×
toilet_c_08_05.8	528	3	3929	0.42	2	7.70	16.85	×
toilet_c_08_05.9	594	3	4441	0.38	2	7.80	17.13	×
toilet_c_08_10.2	232	3	2002	0.93	2	6.10	13.80	×

Table A.57. QBFLIB: Toilet_c Min-QSAT (2)

Name	$\|x_\exists\|$	$\|x_\forall\|$	$\|C\|$	D	S	$Z_{\forall\exists}$	Z_Q	$Z_{\exists\forall}$
toilet_c_10_01.10	320	4	1517	0.73	2	10	28.53	×
toilet_c_10_01.11	352	4	1666	0.66	2	10.20	28.96	×
toilet_c_10_01.12	384	4	1815	0.60	2	10.40	29.41	×
toilet_c_10_01.13	416	4	1964	0.56	2	10.60	29.87	×
toilet_c_10_01.14	448	4	2113	0.51	2	10.80	30.34	×
toilet_c_10_01.15	480	4	2262	0.48	2	11	30.80	×
toilet_c_10_01.16	512	4	2411	0.45	2	11.17	31.25	×
toilet_c_10_01.17	544	4	2560	0.42	2	11.33	31.68	×
toilet_c_10_01.18	576	4	2709	0.40	2	11.50	32.10	×
toilet_c_10_01.19	608	4	2858	0.37	2	11.67	32.52	×
toilet_c_10_01.2	64	4	325	4.34	2	7	×	×
toilet_c_10_01.20	640	4	3007	0.36	2	11.83	32.96	×
toilet_c_10_01.3	96	4	474	2.72	2	7.50	25.50	×
toilet_c_10_01.4	128	4	623	1.97	2	8	26	×
toilet_c_10_01.5	160	4	772	1.54	2	8.50	26.50	×
toilet_c_10_01.6	192	4	921	1.26	2	9	27	×
toilet_c_10_01.7	224	4	1070	1.07	2	9.25	27.38	×
toilet_c_10_01.8	256	4	1219	0.93	2	9.50	27.73	×
toilet_c_10_01.9.	288	4	1368	0.82	2	9.75	28.11	×
toilet_c_10_05.10	800	4	6665	0.28	2	7.90	19.92	×
toilet_c_10_05.11	880	4	7350	0.25	2	8	20.25	×
toilet_c_10_05.12	960	4	8035	0.23	2	8.07	20.54	×
toilet_c_10_05.2	160	4	1185	1.44	2	6.20	17	×
toilet_c_10_05.3	240	4	1870	0.95	2	6.70	17.50	×
toilet_c_10_05.4	320	4	2555	0.71	2	7.20	18.00	×
toilet_c_10_05.5	400	4	3240	0.56	2	7.40	18.35	×
toilet_c_10_05.6	480	4	3925	0.47	2	7.50	18.64	×
toilet_c_10_05.7	560	4	4610	0.40	2	7.60	18.97	×
toilet_c_10_05.8	640	4	5295	0.35	2	7.70	19.30	×
toilet_c_10_05.9	720	4	5980	0.31	2	7.80	19.61	×
toilet_c_10_10.2	280	4	2710	0.78	2	6.10	16	×

Bibliography

[1] Gecode: Generic constraint development environment. `http://www.gecode.org/`. [Online; accessed 1-July-2014].

[2] T. Achterberg. *Constraint Integer Programming*. PhD thesis, 2007.

[3] T. Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

[4] T. Achterberg, T. Berthold, T. Koch, and K. Wolter. Constraint integer programming: a new approach to integrate cp and mip. In *Proceedings of the 5th international conference on Integration of AI and OR techniques in constraint programming for combinatorial optimization problems*, CPAIOR'08, pages 6–20, Berlin, Heidelberg, 2008. Springer.

[5] F. Altenstedt. *Aspects on asset liability management via stochastic programming*. Number N.S., 2024 in Doktorsavhandlingar vid Chalmers Tekniska Högskola. Chalmers Univ. of Technology, Göteborg, 2003.

[6] D. Applegate, W. J. Cook, S. Dash, and D. G. Espinoza. Exact solutions to linear programming problems. *Oper. Res. Lett.*, 35(6):693–699, 2007.

[7] L. Aranburu, L. Escudero, M. Garn, and G. Prez. A so-called cluster benders decomposition approach for solving two-stage stochastic linear problems. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 20(2):279–295, 2012.

[8] O. Arieli and M. W. A. Caminada. A qbf-based formalization of abstract argumentation semantics. *J. Applied Logic*, 11(2):229–252, 2013.

[9] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[10] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. Erratum: Information Processing Letters 14(4): 195 (1982).

[11] A. Atamtürk and M. Zhang. Two-stage robust network flow and design under demand uncertainty. *Operations Research*, 55(4):662–673, 2007.

[12] A. Ayari and D. A. Basin. Qubos: Deciding quantified boolean logic using propositional satisfiability solvers. In M. Aagaard and J. W. O'Leary, editors, *FMCAD*, volume 2517 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2002.

[13] E. Balas. Projection and lifting in combinatorial optimization. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization*, volume 2241 of *Lecture Notes in Computer Science*, pages 26–56. Springer, 2001.

[14] S. Basu, R. D. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry*, volume 10. Springer, 2006.

[15] E. M. L. Beale. On minimizing a convex function subject to linear inequalities. *J. Roy. Statist. Soc. Ser. B.*, 17, 1955.

[16] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[17] A. Ben-Tal, D. den Hertog, A. De Waegenaere, B. Melenberg, and G. Rennen. Robust solutions of optimization problems affected by uncertain probabilities. *Manage. Sci.*, 59(2):341–357, Feb. 2013.

[18] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Princeton University Press, 2009.

[19] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376, Mar. 2004.

[20] A. Ben-Tal and A. Nemirovski. Robust truss topology design via semidefinite programming, research report 4/95. In *Optimization Laboratory, Faculty of Industrial Engineering and Management, Technion The Israel Institute of Technology, Technion City, Haifa 32000*, 1995.

[21] A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23:769–805, 1998.

[22] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25:1–13, 1999.

[23] A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88:411–424, 2000.

[24] A. Ben-Tal and A. Nemirovski. Robust optimization - methodology and applications. *Mathematical Programming*, 92(3):453–480, 2002.

[25] J. F. Benders. Partitioning Procedures for Solving Mixed Variables Programming Problems. *Numerische Mathematik*, 4:238–252, 1962.

[26] M. Benedetti, A. Lallouet, and J. Vautard. Reusing csp propagators for qcsps. In *CSCLP*, pages 63–77, 2006.

[27] M. Benedetti, A. Lallouet, and J. Vautard. Qcsp made practical by virtue of restricted quantification. In *IJCAI*, pages 38–43, 2007.

[28] M. Benedetti, A. Lallouet, and J. Vautard. Modeling adversary scheduling with qcsp$^+$. In *SAC*, pages 151–155, 2008.

[29] M. Benedetti, A. Lallouet, and J. Vautard. Quantified constraint optimization. In *CP*, pages 463–477, 2008.

[30] M. Benedetti and H. Mangassarian. Qbf-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.

[31] F. Benhamou and F. Goualard. Universally quantified interval constraints. In *CP*, pages 67–82, 2000.

[32] H. Bennaceur. A comparison between sat and csp techniques. *Constraints*, 9(2):123–138, Apr. 2004.

[33] T. Benoist, E. Gaudin, and B. Rottembourg. Constraint programming contribution to benders decomposition: a case study. In *In CP-02*, pages 603–617, 2002.

[34] M. Bergner, M. E. Lübbecke, E. Malaguti, and E. Traversi. Partial convexification of general mips by dantzig-wolfe reformulation. In *Integer Programming and Combinatorial Optimization, volume 6655 of Lect. Notes Comput. Sci*, pages 39–51. Springer, 2011.

[35] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000.

[36] D. Bertsimas, D. B. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM Rev.*, 53(3):464–501, Aug. 2011.

[37] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1-3):49–71, 2003.

[38] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[39] D. Bertsimas and M. Sim. Robust Discrete Optimization Under Ellipsoidal Uncertainty Sets. 2004.

[40] D. Bertsimas and A. Thiele. A robust optimization approach to supply chain management. In *IPCO*, pages 86–100, 2004.

[41] D. Bertsimas and A. Thiele. Robust and Data-Driven Optimization: Modern Decision-Making Under Uncertainty. In *Tutorials on Operations Research, INFORMS*, 2006.

[42] H.-G. Beyer and B. Sendhoff. Robust optimization A comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196:3190–3218, 2007.

[43] A. Biere. Resolve and expand. In *SAT*, 2004.

[44] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[45] J. R. Birge. Decomposition and partitioning methods for multistage stochastic linear programs. *Oper. Res.*, 33(5):989–1007, 1985.

[46] J. R. Birge, C. J. Donohue, D. F. Holmes, and O. G. Svintsitski. A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs. *Mathematical Programming*, 75(2):327–352, Nov. 1996.

[47] J. R. Birge and F. Louveaux. *Introduction to stochastic programming*. Springer series in operations research. Springer, New York, 1997.

[48] J. R. Birge and F. V. Louveaux. A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, 34(3):384–392, 1988.

[49] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50:3–15, 2002.

[50] L. Bordeaux. A constraint-propagation approach to quantified constraints. In *Student sessions at CP 2002*, Ithaca, NY, 2002.

[51] L. Bordeaux and E. Monfroy. Beyond NP: Arc-consistency for quantified constraints. In P. Van Hentenryck, editor, *Proc. of the 8th Int. Conf. on Constraint Programming (CP)*, number 2470 in LNCS, pages 371–386, Ithaca, USA, 2002. Springer.

[52] F. Börner, A. A. Bulatov, H. Chen, P. Jeavons, and A. A. Krokhin. The complexity of constraint satisfaction games and qcsp. *Inf. Comput.*, 207(9):923–944, 2009.

[53] F. Börner, A. A. Bulatov, P. Jeavons, and A. A. Krokhin. Quantified constraints: Algorithms and complexity. In M. Baaz and J. A. Makowsky, editors, *CSL*, volume 2803 of *Lecture Notes in Computer Science*, pages 58–70. Springer, 2003.

[54] S. C. Brailsford, C. N. Potts, and B. M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, 1999.

[55] C. W. Brown. Qepcad b: A program for computing with semi-algebraic sets using cads. *SIGSAM BULLETIN*, 37:97–108, 2003.

[56] C. W. Brown and J. H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, ISSAC '07, pages 54–60, New York, NY, USA, 2007. ACM.

[57] P. Brucker. *Scheduling Algorithms*. Springer, Secaucus, NJ, USA, 3rd edition, 2001.

[58] H. K. Büning and U. Bubeck. Theory of quantified boolean formulas. In *Handbook of Satisfiability*, pages 735–760. 2009.

[59] H. K. Büning, M. Karpinski, and A. Flögel. Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.

[60] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified boolean formulae. In *AAAI/IAAI*, pages 262–267, 1998.

[61] M. Cadoli, M. Schaerf, M. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. In *Journal of Automated Reasoning*, pages 262–267. AAAI Press, 1999.

[62] X. Cai, D. C. Mckinney, L. S. Lasdon, and D. W. Watkins. Solving large nonconvex water resources management models using generalized benders decomposition. *Operations Research*, 49:235–245, 2001.

[63] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Jounal of the ACM*, 28(1):114–133, Jan. 1981.

[64] V. Chandru and M. R. Rao. Algorithms and theory of computation handbook. chapter Linear Programming, pages 30–30. Chapman & Hall/CRC, 2010.

[65] A. Charnes and W. W. Cooper. Chance-Constrained programming. *Management Science*, 6(1):73–79, 1959.

[66] H. Chen and M. Pál. Optimization, games, and quantified constraint satisfaction. In *MFCS*, pages 239–250, 2004.

[67] H. M. Chen. *The computational complexity of quantified constraint satisfaction*. PhD thesis, Ithaca, NY, USA, 2004.

[68] K. Claessen, N. Eén, M. Sheeran, N. Sörensson, A. Voronov, and K. Åkesson. Sat-solving in practice, with a tutorial example from supervisory control. *Discrete Event Dynamic Systems*, 19(4):495–524, 2009.

[69] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, 12(3):299–328, Sept. 1991.

[70] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[71] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[72] G. B. Dantzig. Linear programming under uncertainty. *Management Science*, 1(3-4):197–206, 1955.

[73] G. B. Dantzig and B. C. Eaves. Fourier-motzkin elimination and its dual. *J. Comb. Theory, Ser. A*, 14(3):288–297, 1973.

[74] G. B. Dantzig and A. Medansky. On the solution of two-stage linear programs under uncertainty. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 165–176, Berkeley, Calif., 1961. University of California Press.

[75] G. B. Dantzig and M. N. Thapa. *Linear Programming 1: Introduction*. Springer, Secaucus, NJ, USA, 1997.

[76] G. B. Dantzig and M. N. Thapa. *Linear Programming 2: Theory and extensions*. Springer series in operations research. Springer, New York, 2003.

[77] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[78] M. Davis and H. Putnam. A computing procedure for quantification theory. *Jounal of the ACM*, 7(3):201–215, 1960.

[79] M. Dempster and R. Thompson. Parallelization and aggregation ofnested benders decomposition. *Annals of Operations Research*, 81(0):163–188, 1998.

[80] M. A. H. Dempster and R. T. Thompson. Parallelization and aggregation of nested Benders decomposition. *Annals of Operations Research*, 81(0):163–188, Jan. 1998.

[81] N. Dershowitz, Z. Hanna, and J. Katz. Bounded model checking with qbf. In *in Intl Conf. on Theory and Applications of Satisfiability Testing*, pages 408–414. Springer, 2005.

[82] K. Devlin. *The millennium problems. The seven greatest unsolved mathematical puzzles of our time.* New York, NY: Basic Books. xii, 237 p. $ 26.00 , 2002.

[83] M. Dhiflaoui, S. Funke, C. Kwappik, K. Mehlhorn, M. Seel, E. Schömer, R. Schulte, and D. Weber. Certifying and repairing solutions to large lps how good are lp-solvers? In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 255–256, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[84] A. Dolzmann and T. Sturm. Redlog computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31:2–9, 1996.

[85] A. Dolzmann, T. Sturm, and V. Weispfenning. Real quantifier elimination in practice. In *Algorithmic Algebra and Number Theory*, pages 221–247. Springer, 1998.

[86] B. Dörig, T. Ederer, P. Hedrich, U. Lorenz, P. F. Pelz, and P. Pöttgen. Technical operations research (TOR) exemplified by a hydrostatic power transmission system. *9.IFK Proceedings Vol.1*, Mar 2014.

[87] T. Ederer. `http://www.mathematik.tu-darmstadt.de/preprint.php?id=2685`, Jan. 2014. [Online; accessed 1-Jul-2014].

[88] T. Ederer, U. Lorenz, A. Martin, and J. Wolf. Quantified linear programs: a computational study. *Proc. of the 11th Ann. Europ. Symp. on Algorithms (ESA 2012), pages=203–214, year=2011, publisher=Springer*.

[89] T. Ederer, U. Lorenz, T. Opfer, and J. Wolf. Modeling games with the help of quantified integer linear programs. In *ACG*, pages 270–281, 2011.

[90] U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving advanced reasoning tasks using quantified boolean formulas. In *AAAI/IAAI*, pages 417–422, 2000.

[91] P. Eirinakis, S. Ruggieri, K. Subramani, and P. J. Wojciechowski. Computational complexity of inclusion queries over polyhedral sets. In *ISAIM*, 2012.

[92] F. Eisenbrand and T. Rothvoß. A PTAS for static priority real-time scheduling with resource augmentation. *Proc. of ICALP'08*, pages 246–257, 2008.

[93] A. Eremin and M. Wallace. Hybrid benders decomposition algorithms in constraint logic programming. In T. Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2001.

[94] C. Fábián and Z. Szöke. Solving two-stage stochastic programming problems with level decomposition. *Computational Management Science*, 4(4):313–353, 2007.

[95] M. Fischetti, D. Salvagnin, and A. Zanette. Minimal Infeasible Subsystems and Benders Cuts. Technical report, University of Padova, Feb. 2008.

[96] L. Fortnow and S. Homer. A short history of computational complexity. In *The History of Mathematical Logic*. North-Holland, 2002.

[97] H. I. Gassmann. Mslip: A computer code for the multistage stochastic linear programming problem. *Mathematical Programming*, 47(3):407–423, Aug. 1990.

[98] E. Gawrilow and M. Joswig. Polymake: a framework for analyzing convex polytopes. In G. Kalai and G. M. Ziegler, editors, *Polytopes — Combinatorics and Computation*, pages 43–74. Birkhäuser, 2000.

[99] D. M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 13:10–12, 1985.

[100] K. Geddes, S. Czapor, and G. Labahn. *Algorithms for computer algebra*. Kluwer Acad., Boston [u.a.], 1992.

[101] B. Geißler. *Towards Globally Optimal Solutions for MINLPs by Discretization Techniques with Applications in Gas Network Optimization*. PhD thesis, 211.

[102] B. Geißler, A. Martin, A. Morsi, and L. Schewe. Using piecewise linear functions for solving minlps. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 287–314. Springer New York, 2012.

[103] I. P. Gent, P. Nightingale, and A. G. D. Rowley. Encoding quantified csps as quantified boolean formulae. In *ECAI*, pages 176–180, 2004.

[104] I. P. Gent, P. Nightingale, and K. Stergiou. Qcsp-solve: A solver for quantified constraint satisfaction problems. In *In Proc. of Int. Joint. Conf. on Artificial Intelligence (IJCAI*, pages 138–143. Morgan Kaufmann, 2005.

[105] I. P. Gent and A. G. D. Rowley. Encoding connect-4 using quantified boolean formulae. In *in Modelling and Reformulating Constraint Satisfaction Problems*, pages 78–93, 2003.

[106] I. P. Gent and T. Walsh. Beyond np: the qsat phase transition. In *AAAI/IAAI*, pages 648–653, 1999.

[107] A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[108] R. Gerber, W. Pugh, and M. Saksena. Parametric dispatching of hard real-time tasks. *IEEE Transactions on Computers*, 44:471–479, 1995.

[109] L. E. Ghaoui and H. Lebret. Robust solutions to least-squares problems with uncertain data, 1997.

[110] L. E. Ghaoui, F. Oustry, and H. Lebret. Robust solutions to uncertain semidefinite programs. *SIAM J. OPTIMIZATION*, 9(1):33–52, 1998.

[111] P. Gill, W. Murray, M. Saunders, and M. Wright. *Numerical Linear Algebra and Optimization*. Basic Books, 1961.

[112] E. Giunchiglia, P. Marin, and M. Narizzano. Qube7.0. *JSAT*, 7(2-3):83–88, 2010.

[113] E. Giunchiglia, M. Narizzano, L. Pulina, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2014. `www.qbflib.org`.

[114] E. Giunchiglia, M. Narizzano, and A. Tacchella. Learning for quantified boolean logic satisfiability. In *AAAI/IAAI*, pages 649–654, 2002.

[115] E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for quantified boolean logic satisfiability. *Artif. Intell.*, 145(1-2):99–120, 2003.

[116] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, Mar. 1991.

[117] S. Grothklags. Fleet assignment with connection dependent ground times. In *Proc. of the 11th Ann. Europ. Symp. on Algorithms (ESA 2003)*, pages 667–678, 2003.

[118] M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 2nd edition, 1993.

[119] D. Haugland and S. W. Wallace. Solving many linear programs that differ only in the right-hand side. *European J. Oper. Res.*, 37(3):318–324, 1988.

[120] J. Ho, E. Loute, and K. C. o. B. A. University of Tennessee. *Computational Experience with Advanced Implementation of Decomposition Algorithms for Linear Programming*. Working paper (University of Tennessee, Knoxville. College of Business Administration). College of Business Administration, University of Tennessee, 1982.

[121] J. Ho and R. Sundarraj. *DECOMP: an implementation of Dantzig-Wolfe decomposition for linear programming*. Lecture notes in economics and mathematical systems. Springer, 1989.

[122] J. K. Ho and E. Loute. An advanced implementation of the dantzigwolfe decomposition algorithm for linear programming. *Mathematical Programming*, 20(1):303–326, 1981.

[123] J. K. Ho and R. P. Sundarraj. Distributed nested decomposition of staircase linear programs. *ACM Trans. Math. Softw.*, 23(2):148–173, June 1997.

[124] D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS, 1997.

[125] T. Huynh, L. Joskowicz, C. Lassez, and J.-L. Lassez. Reasoning about linear constraints using parametric queries. In *FSTTCS*, pages 1–20, 1990.

[126] T. Huynh, C. Lassez, and J.-L. Lassez. Fourier algorithm revisited. In H. Kirchner and W. Wechler, editors, *ALP*, volume 463 of *Lecture Notes in Computer Science*, pages 117–131. Springer, 1990.

[127] M. Janota, W. Klieber, J. Marques-Silva, and E. Clarke. Solving qbf with counterexample guided refinement. In *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*, SAT'12, pages 114–128, Berlin, Heidelberg, 2012. Springer.

[128] P. Kall and S. Wallace. *Stochastic programming*. Wiley-Interscience series in systems and optimization. Wiley, 1994.

[129] R. Kannan. A polynomial algorithm for the two-variable integer programming problem. *Jounal of the ACM*, 27(1):118–122, Jan. 1980.

[130] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 302–311, New York, NY, USA, 1984. ACM.

[131] R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[132] C. Keil and C. Jansson. Computational experience with rigorous error bounds for the netlib linear programming library. *Reliable Computing*, 12:4–303, 2006.

[133] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979.

[134] A. J. Kleywegt and A. Shapiro. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, page 502, 2001.

[135] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artif. Intell.*, 6(4):293–326, 1975.

[136] A. Koberstein. *The Dual Simplex Method, Techniques for a Fast and Stable Implementation*. PhD thesis, Universität Paderborn, 2005. `http://books.google.de/books?id=QgJkuAAACAAJ`.

[137] A. Koberstein. Progress in the dual simplex algorithm for solving large scale lp problems: techniques for a fast and stable implementation. *Computational Optimization and Applications*, 41(2):185–204, 2008.

[138] T. Koch. The final NETLIB-LP results. *Op. Res. Letters*, 32, 2003.

[139] M.-C. Lai, H.-s. Sohn, T.-L. B. Tseng, and C. Chiang. A hybrid algorithm for capacitated plant location problem. *Expert Syst. Appl.*, 37(12):8599–8605, Dec. 2010.

[140] J.-L. Lassez and M. J. Maher. On fourier's algorithm for linear arithmetic constraints. *Journal of Automated Reasoning*, 9(3):373–379, 1992.

[141] F. Lewis, D. Vrabie, and V. Syrmos. *Optimal Control*. Wiley, 2012.

[142] R. Li, D. Zhou, and D. Du. Satisfiability and integer programming as complementary tools. In *ASP-DAC*, pages 879–882, 2004.

[143] J. Linderoth and S. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Comput. Optim. Appl.*, 24(2-3):207–250, Feb. 2003.

[144] F. Lonsing and A. Biere. Nenofex: Expanding nnf for qbf solving. In *SAT*, pages 196–210, 2008.

[145] F. Lonsing and A. Biere. Depqbf: A dependency-aware qbf solver. *JSAT*, 7(2-3):71–76, 2010.

[146] R. Loos and V. Weispfenning. Applying linear quantifier elimination. *Comput. J.*, 36(5):450–462, 1993.

[147] U. Lorenz, A. Martin, and J. Wolf. Polyhedral and algorithmic properties of quantified linear programs. In *Proc. of the 18th Ann. Europ. Symp. on Algorithms (ESA 2011*, pages 512–523, 2010.

[148] U. Lorenz, T. Opfer, and J. Wolf. Solution techniques for quantified linear programs and the links to gaming. *Computers and Games - 8th International Conference, CG2013, August 13-15, 2013, Yokohama, Japan*, forthcoming, 2014.

[149] D. Mahlke, A. Martin, and S. Moritz. A mixed integer approach for time-dependent gas network optimization. *Optimization Methods and Software*, 25(4):625–644, 2010.

[150] S. Malik and L. Zhang. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM*, 52(8):76–82, Aug. 2009.

[151] N. Mamoulis and K. Stergiou. Algorithms for quantified constraint satisfaction problems. In *In Proceedings of CP-2004*, pages 752–756, 2004.

[152] J. Marques-Silva. Practical applications of boolean satisfiability. In *In Workshop on Discrete Event Systems (WODES)*. IEEE Press, 2008.

[153] J. P. Marques-silva and K. A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.

[154] A. Martin, K. Klamroth, J. Lang, G. Leugering, A. Morsi, M. Oberlack, M. Ostrowski, and R. Rosen. *Mathematical Optimization of Water Networks*. International Series of Numerical Mathematics. Springer Basel, 2012.

[155] N. Megow and T. Vredeveld. Approximation results for preemtive stochastic online scheduling. *Proc. of the 14th Ann. Europ. Symp. on Algorithms (ESA 2006*, 2006.

[156] H. Mittelmann. Benchmarks for optimization software. `http://plato.asu.edu/bench.html`. [Online; accessed 1-July-2014].

[157] A. Morsi, B. Geißler, and A. Martin. Mixed integer optimization of water supply networks. In A. Martin, K. Klamroth, J. Lang, G. Leugering, A. Morsi, M. Oberlack, M. Ostrowski, and R. Rosen, editors, *Mathematical Optimization of Water Networks*, volume 162 of *International Series of Numerical Mathematics*, pages 35–54. Springer Basel, 2012.

[158] D. Morton. An enhanced decomposition algorithm for multistage stochastic hydroelectric scheduling. *Annals of Operations Research*, 64(1):211–235, Dec. 1996.

[159] T. Motzkin. *Beiträge zur Theorie der linearen Ungleichungen*. Azriel, 1936.

[160] M. Narizzano, L. Pulina, and A. Tacchella. The qbfeval web portal. In *Logics in Artificial Intelligence*, pages 494–497. Springer Berlin Heidelberg, 2006.

[161] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[162] P. Nightingale. Non-binary quantified csp: algorithms and modelling. *Constraints*, 14(4):539–581, 2009.

[163] F. Ordez and R. M. Freund. Computational experience and the explanatory value of condition measures for linear optimization. *SIAM Journal on Optimization*, 14(2):307–333, 2003.

[164] C. H. Papadimitriou. Games against nature. *J. Comput. Syst. Sci.*, 31(2):288–301, 1985.

[165] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[166] P. F. Pelz, U. Lorenz, T. Ederer, S. Lang, and G. Ludwig. Designing pump systems by discrete mathematical topology optimization: The artificial fluid systems designer (afsd). *International Rotating Equipment Conference*, 2012.

[167] P. F. Pelz, U. Lorenz, T. Ederer, M. Metzler, and P. Pöttgen. Global system optimization and scaling for turbo systems and machines. In *15th International Symposium on Transport Phenomena and Dynamics of Rotating Machinery, ISROMAC-15*, Feb 2014.

[168] P. F. Pelz, U. Lorenz, and G. Ludwig. Besser geht's nicht. TOR plant das energetisch optimale Fluidsystem. *chemie & more*, (1/2014), Januar 2014.

[169] C. Peschiera, L. Pulina, and A. Tacchella. Designing a solver competition: the qbfeval'10 case study. In A. Stump, G. Sutcliffe, and C. Tinelli, editors, *EMSQMS@IJCAR*, volume 6 of *EPiC Series*, pages 19–32. EasyChair, 2010.

[170] M. E. Pfetsch, A. Fügenschuh, B. Geißler, N. Geißler, R. Gollmer, B. Hiller, J. Humpola, T. Koch, T. Lehmann, A. Martin, A. Morsi, J. Rövekamp, L. Schewe, M. Schmidt, R. Schultz, R. Schwarz, J. Schweiger, C. Stangl, M. C. Steinbach, S. Vigerske, and B. M. Willert. Validation of nominations in gas network optimization: Models, methods, and solutions. 2013. accepted in Optim. Meth. Softw.

[171] W. Pijls and A. de Bruin. Game tree algorithms and solution trees. *Theor. Comput. Sci.*, 252(1-2):197–215, 2001.

[172] Y. Pochet and L. A. Wolsey. *Production Planning by Mixed Integer Programming (Springer Series in Operations Research and Financial Engineering)*. Springer, Secaucus, NJ, USA, 2006.

[173] C. A. Poojari and J. E. Beasley. Improving benders decomposition using a genetic algorithm. *European Journal of Operational Research*, 199(1):89–97, 2009.

[174] A. Prékopa. *Stochastic Programming*. Mathematics and Its Applications. Springer, 1995.

[175] A. J. Qureshi, J.-Y. Dantan, J. Bruyere, and R. Bigot. Set based robust design of mechanical systems using the quantifier constraint satisfaction algorithm. *Eng. Appl. Artif. Intell.*, 23(7):1173–1186, Oct. 2010.

[176] S. Ratschan. Continuous first-order constraint satisfaction. In *AISC*, pages 181–195, 2002.

[177] S. Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Trans. Comput. Log.*, 7(4):723–748, 2006.

[178] S. Ratschan. Applications of quantified constraint solving over the reals - bibliography. *CoRR*, abs/1205.5571, 2012.

[179] J. Renegar. Some perturbation theory for linear programming. *Mathematical Programming*, 65:73–91, 1992.

[180] J. Rios. Algorithm 928: A general, parallel implementation of dantzig–wolfe decomposition. *ACM Trans. Math. Softw.*, 39(3):21:1–21:10, May 2013.

[181] R. T. Rockafellar and R. J. B. Wets. Scenarios and Policy Aggregation in Optimization under Uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 1991.

[182] K. H. Rosen. *Handbook of Discrete and Combinatorial Mathematics, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2010.

[183] P. Rubin. Dual values for primal variable bounds. `http://orinanobworld.blogspot.de/2011/07/farkas-certificates-in-cplex.html`, Sep 2010.

[184] P. Rubin. Farkas certificates in cplex. `http://orinanobworld.blogspot.de/2011/07/farkas-certificates-in-cplex.html`, Jul 2011. [Online; accessed 1-July-2014].

[185] S. Ruggieri, P. Eirinakis, K. Subramani, and P. J. Wojciechowski. On the complexity of quantified linear systems. *Theor. Comput. Sci.*, 518:128–134, 2014.

[186] A. Ruszczyński. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35(3):309–333, July 1986.

[187] A. Ruszczynski. The regularized decomposition for stochastic programming problems, 1995.

[188] A. Ruszczynski. Decomposition methods in stochastic programming. *Mathematical Programming*, 79:333–353, 1997.

[189] A. Ruszczyński and A. Shapiro. *Stochastic Programming*. Handbooks in operations research and management science. Elsevier, 2003.

[190] A. Ruszczynski and A. Swietanowski. Accelerating the regularized decomposition method for two stage stochastic linear problems. *European Journal of Operational Research*, 101(2):328–342, 1997.

[191] A. Ruszczyski. Parallel decomposition of multistage stochastic programming problems. *Mathematical Programming*, 58(1-3):201–228, 1993.

[192] M. Sachenbacher and P. Maier. Test strategy generation using quantified csps. In *CP*, pages 566–570, 2008.

[193] N. V. Sahinidis. Optimization under uncertainty: State-of-the-art and opportunities. *Computers and Chemical Engineering*, 28:971–983, 2004.

[194] M. C. Saksena. *Parametric Scheduling for Hard Real-time Systems*. PhD thesis, College Park, MD, USA, 1994.

[195] M. Schaefer and C. Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT News*, pages 32–49, 2002.

[196] T. J. Schaefer. On the complexity of some two-person perfect-information games. *J. Comput. Syst. Sci.*, 16(2):185–225, 1978.

[197] A. Schrijver. *Theory of linear and integer programming*. John Wiley and Sons, Inc., New York, NY, USA, 1986.

[198] R. Schultz. Stochastic programming with integer variables. *Math. Progr.*, 97:285–309, 2003.

[199] S. Sen. *Algorithms for Stochastic Mixed-Integer Programming Models*. Handbooks in OR and MS.

[200] A. Shapiro. On complexity of multistage stochastic programs. *Operations Research Letters*, 34:2006, 2006.

[201] M. Sim. *Robust optimization*. PhD thesis, Massachusetts Institute of Technology, 2004.

[202] K. J. Singh, A. B. Philpott, and R. K. Wood. Dantzig-wolfe decomposition for solving multistage stochastic capacity-planning problems. *Operations Research*, 57(5):1271–1286, 2009.

[203] J. Sirikum, A. Techanitisawad, and V. Kachitvichyanukul. A new efficient ga-benders' decomposition method: For power generation expansion planning with emission controls. *Power Systems, IEEE Transactions on*, 22(3):1092–1100, Aug 2007.

[204] A. L. Soyster. Convex Programming with Set-Inclusive Constraints and Applications to Inexact Linear Programming. *Operations Research*, 21(5):1154–1157, 1973.

[205] L. J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

[206] K. Subramani. Parametric scheduling - algorithms and complexity. In *HiPC*, pages 36–46, 2001.

[207] K. Subramani. A specification framework for real-time scheduling. In *SOFSEM*, pages 195–207, 2002.

[208] K. Subramani. An analysis of quantified linear programs. In *DMTCS*, pages 265–277, 2003.

[209] K. Subramani. Analyzing selected quantified integer programs. In *IJCAR*, pages 342–356, 2004.

[210] K. Subramani. Partially clairvoyant scheduling for aggregate constraints. *JAMDS*, 9(4):225–240, 2005.

[211] K. Subramani. Tractable fragments of presburger arithmetic. *Theory Comput. Syst.*, 38(5):647–668, 2005.

[212] K. Subramani. On a decision procedure for quantified linear programs. *Annals of Mathematics and Artificial Intelligence*, 51(1):55–77, 2007.

[213] A. Tarski. *A decision method for elementary algebra and geometry*. Rand report. Rand Corporation, 1948.

[214] J. Tebboth. *A Computational Study of Dantzig-Wolfe Decomposition*. PhD thesis, University of Buckingham, 2001.

[215] A. Thiele, C. Murat, and V. Gabrel. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 2013.

[216] S. Trukhanov, L. Ntaimo, and A. Schaefer. Adaptive multicut aggregation for two-stage stochastic linear programs with recourse. *European Journal of Operational Research*, 206(2):395 – 406, 2010.

[217] M. H. van der Vlerk. Stochastic integer programming bibliography. World Wide Web, `http://www.eco.rug.nl/mally/biblio/sip.html`, 1996-2007. [Online; accessed 1-July-2014].

[218] M. H. van der Vlerk. Stochastic programming bibliography. World Wide Web, `http://www.eco.rug.nl/mally/spbib.html`, 1996-2007. [Online; accessed 1-July-2014].

[219] R. M. Van Slyke and R. Wets. L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663, 1969.

[220] G. Verger and C. Bessière. : A bottom-up approach for solving quantified csps. In *CP*, pages 635–649, 2006.

[221] B. Verweij, S. Ahmed, A. J. Kleywegt, G. Nemhauser, and E. Shapiro. The sample average approximation method applied to stochastic routing problems: a computational study. *Computational Optimization and Applications*, 24:2003, 2003.

[222] J. P. Vielma, S. Ahmed, and G. Nemhauser. Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Oper. Res.*, 58(2):303–315, Mar. 2010.

[223] M. Wäldele, H. Birkhofer, A. Fügenschuh, and A. Martin. Modeling properties for the design of branched sheet metal products. In A. Chakrabarti, editor, *Research into Design: Supporting Multiple Facets of Product Development*, pages 287 – 294. Research Publishing, 2009.

[224] S. W. Wallace and W. T. Ziemba. *Applications of Stochastic Programming*. MPS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2005.

[225] T. Walsh. Sat v csp. In R. Dechter, editor, *CP*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.

[226] V. Weispfenning. The complexity of linear problems in fields. *J. Symb. Comput.*, 5(1-2):3–27, Feb. 1988.

[227] V. Weispfenning. Simulation and optimization by quantifier elimination. *Journal of Symbolic Computation*, 24:208, 1996.

[228] V. Weispfenning. Quantifier elimination for real algebra - the quadratic case and beyond. *Appl. Algebra Eng. Commun. Comput.*, 8(2):85–101, 1997.

[229] R. J.-B. Wets. Stochastic programs with fixed recourse: The equivalent deterministic program. *SIAM Review*, 16(3):pp. 309–339, 1974.

[230] R. J. B. Wets. Large scale linear programming techniques. In *Numerical techniques for stochastic optimization*, volume 10 of *Springer Ser. Comput. Math.*, pages 65–93. Springer, Berlin, 1988.

[231] R. Wittrock. *Advances in a Nested Decomposition Algorithm for Solving Staircase Linear Programs*. Stanford University, 1983.

[232] C. Wolf and A. Koberstein. Dynamic sequencing and cut consolidation for the parallel hybrid-cut nested l-shaped method. *European Journal of Operational Research*, 230(1):143–156, 2013.

[233] R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996. `http://www.zib.de/Publications/abstracts/TR-96-09/`.

[234] H.-X. Yu and L. Jin. An brief introduction to robust optimization approach. *Int. J. Pure Appl. Math.*, 74(1):121–124, 2012.

[235] R. Zabih and D. A. McAllester. A rearrangement search strategy for determining propositional satisfiability. In H. E. Shrobe, T. M. Mitchell, and R. G. Smith, editors, *AAAI*, pages 155–160. AAAI Press / The MIT Press, 1988.

[236] L. Zhang. Conflict driven learning in a quantified boolean satisfiability solver. In *in ICCAD 02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 442–449. ACM Press, 2002.

[237] L. Zhang, H. Hermanns, F. Eisenbrand, and D. Jansen. Flow faster: Efficient decision algorithms for probabilistic simulations. *Logical Methods in Computer Science*, 4(4), 2008.

[238] N.-F. Zhou, M. Tsuru, and E. Nobuyama. A comparison of cp, ip, and sat solvers through a common interface. In *Proceedings of the 2012 IEEE 24th International Conference on Tools with Artificial Intelligence - Volume 01*, ICTAI '12, pages 41–48, Washington, DC, USA, 2012. IEEE Computer Society.

[239] V. Zverovich, C. I. Fbin, E. F. D. Ellison, and G. Mitra. A computational study of a solver system for processing two-stage stochastic lps with enhanced benders decomposition. *Mathematical Programming Comput.*, 4(3):211–238, 2012.

List of Figures

List of Tables