POLYPUBLIE
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Titre: Title:	Exploiting structure in Mixed-Integer Linear and Non-linear Programming
Auteur: Author:	Mathieu Tanneau
Date:	2020
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Tanneau, M. (2020). Exploiting structure in Mixed-Integer Linear and Non-linear Programming [Ph.D. thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/5480/

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/5480/
Directeurs de recherche: Advisors:	Andrea Lodi, & Miguel F. Anjos
Programme: Program:	Doctorat en mathématiques

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Exploiting structure in Mixed-Integer Linear and Non-linear Programming

MATHIEU TANNEAU
Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Mathématiques de l'ingénieur

Octobre 2020

© Mathieu Tanneau, 2020.

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Exploiting structure in Mixed-Integer Linear and Non-linear Programming

présentée par **Mathieu TANNEAU**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Louis-Martin ROUSSEAU, président
Andrea LODI, membre et directeur de recherche
Miguel F. ANJOS, membre et codirecteur de recherche
Dominique ORBAN, membre
Daniel BIENSTOCK, membre externe

DEDICATION

“Mathematics allows us to go beyond the intuition,
and explore territories that do not fit within our grasp”
- Cédric Villani

ACKNOWLEDGEMENTS

My first thanks go to my advisors, Miguel Anjos and Andrea Lodi, who have provided me with guidance, support, and, perhaps most importantly, freedom. What I have learned from you stretches far beyond the sole realm of Mathematics.

Thank you Marie, Koladé, Mehdi, Khalid, Mariia, Karine, Marilyne, and all the support staff at GERAD and the CERC DS4DM. Little would be possible without your daily involvement, and your ability to fix the problems that math alone cannot solve.

Thank you to my colleagues and friends. Our countless conversations, lunchs, tea breaks and corridor meetings have been a permanent source of inspiration and motivation for me. Not seeing you was probably the hardest part of those last six months.

Thank you to all my flatmates who, over the years, have reminded me that there is more to life than just math. A special thanks to Baptiste and Laurentia: together we both endured and supported each other during lockdown.

Warm thanks go to my family, whose love and support never wanes, even from afar.

RÉSUMÉ

La programmation mathématique en nombres entiers (PNE), par son caractère générique, permet de formuler de nombreux problèmes de décision de la vie réelle. Ainsi, les outils de PNE sont utilisés quotidiennement pour, par exemple, fabriquer des horaires de personnel ou des itinéraires de véhicules de livraison, opérer des réseaux électriques, gérer des inventaires, ou encore décider de politiques d'investissement. Ce succès est, en grande partie, rendu possible par le développement, au cours des dernières décennies, d'outils logiciels robustes et performants, capables de résoudre efficacement un grand nombre de problèmes pratiques.

Toutefois, l'amélioration de tels outils s'est accompagnée d'une augmentation de la diversité, taille et complexité des problèmes que les praticiens cherchent à résoudre, dépassant parfois les capacités des principaux solveurs de PNE. Dans ce contexte, l'objectif de cette thèse est le développement d'algorithmes spécialisés, i.e., qui exploitent la structure mathématique d'une classe de problèmes donnée, et leur intégration au sein du cadre algorithmique de la PNE. Ce travail explore ainsi une voie hybride entre solveurs PNE génériques et spécialisés, et se situe à l'intersection entre la conception d'algorithmes de PNE, et leur implémentation pratique.

Tout d'abord, nous proposons une formulation PNE pour la gestion d'un réseau électrique comprenant de multiples ressources énergétiques distribuées. Étant donnés la structure du réseau et certains enjeux de respect de la vie privée, le problème est reformulé par décomposition de Dantzig-Wolfe, et résolu par un algorithme de génération de colonnes. Cette approche est générique, i.e., n'importe quel type de ressource peut être pris en compte, et efficace. En effet, des tests numériques montrent que des systèmes comportant plusieurs milliers de ressources peuvent être résolus à l'optimalité.

Ensuite, nous présentons la conception algorithmique et l'implémentation de Tulip, un solveur de programmation linéaire basé sur un algorithme de points intérieurs. Plus spécifiquement, le code de Tulip est structuré de façon à permettre l'utilisation de librairies d'algèbre linéaire spécialisées, qui peuvent être implémentées directement par l'utilisateur. Plusieurs tests numériques démontrent l'efficacité du code, tant pour des instances non-structurées que présentant une certaine structure. Dans le premier cas, la performance de Tulip est comparable à celle de solveurs en source libre. Dans le second, l'utilisation de routines spécialisées améliore la performance d'un facteur 10, et rend Tulip compétitif avec les meilleurs solveurs commerciaux.

Enfin, nous étudions les aspects pratiques de plans coupants disjonctifs pour l'optimisation conique en variables mixtes. En nous appuyant sur la dualité conique, nous formulons un programme d'optimisation conique pour séparer de tels coupes, et analysons les aspects théoriques de plusieurs normalisations. De plus, nous mettons en lumière certaines difficultés rencontrées dans le cadre d'algorithmes d'approximation polyhédrale, et proposons des extensions à plusieurs techniques initialement proposées dans le contexte linéaire. L'efficacité de notre approche est validée à travers plusieurs expériences numériques, et est compétitive avec la génération de coupes par le solveur CPLEX.

ABSTRACT

The past 70 years have witnessed tremendous performance improvements of algorithms and software for mixed-integer programming. These developments have been met by a likewise increase in the size and complexity of the models that practitioners aim to solve: there is no shortage of challenging problems. In that context, this thesis investigates the use of structure-exploiting techniques within generic optimization paradigms, to tackle large-scale or otherwise intractable instances. In particular, we build on the observation that structure-exploiting techniques can often be viewed as specialized implementations of one or several individual components of generic MIP algorithms.

First, we consider the coordination of Distributed Energy Resources in power systems. This problem is first formulated as a centralized –but intractable– mixed-integer linear program, then reformulated using Dantzig-Wolfe decomposition and solved by a column-generation algorithm. The proposed framework is generic, i.e., it can handle resources of arbitrary nature, and computationally efficient: problems with thousands or resources are solved to proven optimality in less than an hour. Finally, besides scalability, the decomposition scheme naturally addresses some privacy concerns regarding the operation of individual resources.

Second, we focus on structure-exploiting interior-point methods for linear programming. To that end, we develop an open-source interior-point solver, Tulip, whose algorithmic framework is disentangled from linear algebra implementations. This flexible design allows to easily integrate specialized, user-provided linear algebra routines. Through various numerical experiments, we demonstrate the flexibility, efficiency and robustness of the code.

Finally, we investigate the separation of disjunctive cutting planes in mixed-integer conic programming, and demonstrate the benefits of exploiting a problem's conic structure. Leveraging conic duality, the separation of disjunctive cuts is formulated in a single cut-generating *conic* problem (CGCP); this dual perspective offers a number of insights that were unavailable to previous approaches. In particular, we study the theoretical and computational merits of several normalization conditions, including the loss of strong conic duality in the separation problem, and the separation of conic-infeasible points in the context of outer-approximation algorithms. Computational experiments demonstrate the efficiency of the proposed approach on several classes of instances.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE OF CONTENTS viii

LIST OF TABLES xii

LIST OF FIGURES xiii

LIST OF SYMBOLS AND ACRONYMS xiv

LIST OF APPENDICES xv

CHAPTER 1 INTRODUCTION 1
- 1.1 Background 2
- 1.2 Objective and contributions 3
- 1.3 Thesis outline 4

CHAPTER 2 CRITICAL LITERATURE REVIEW 5
- 2.1 Structure exploitation in generic MIP solvers 5
- 2.2 Decomposition techniques and available software 5
  - 2.2.1 Dantzig-Wolfe decomposition 6
  - 2.2.2 Benders decomposition 6
  - 2.2.3 Software 7

CHAPTER 3 ARTICLE 1 - A DECENTRALIZED FRAMEWORK FOR THE OPTIMAL COORDINATION OF DISTRIBUTED ENERGY RESOURCES 8
- 3.1 Nomenclature 8
  - 3.1.1 Parameters 8
  - 3.1.2 Sets 10
  - 3.1.3 Variables 10

3.2 Introduction 11
3.2.1 Contribution and outline 13
3.3 Problem formulation 14
3.3.1 Device constraints 14
3.3.2 Household and aggregator constraints 16
3.3.3 Aggregation problem 17
3.4 Distributed Column-Generation framework 18
3.4.1 Dantzig-Wolfe decomposition 18
3.4.2 Distributed column generation 19
3.4.3 Branch-and-price 21
3.5 Computational results 23
3.5.1 Numerical instantiation 24
3.5.2 Implementation details 26
3.5.3 Performance analysis 26
3.5.4 Evolution of the aggregated load 28
3.5.5 Further discussion 30
3.6 Conclusion 31

CHAPTER 4 ARTICLE 2 - DESIGN AND IMPLEMENTATION OF A MODULAR INTERIOR-POINT SOLVER FOR LINEAR OPTIMIZATION 33
4.1 Introduction 33
4.1.1 Exploiting structure in IPMs 34
4.1.2 Contributions and outline 35
4.2 Notations 36
4.3 Regularized homogeneous interior-point algorithm 37
4.3.1 Homogeneous self-dual embedding 38
4.3.2 Regularized formulation 38
4.3.3 Regularized homogeneous algorithm 40
4.4 Solving linear systems 45
4.4.1 Augmented system 46
4.4.2 Black-box linear solvers 47
4.5 Presolve 48
4.5.1 Presolve 48
4.5.2 Postsolve 49
4.6 Implementation details 50
4.6.1 Bounds on variables 50

4.6.2 Solver parameters 52
4.6.3 Default linear solvers 52
4.7 Computational results 53
4.7.1 Results on general LP instances 55
4.7.2 Results on structured LP instances 57
4.7.3 Solving problems in extended precision 68
4.8 Conclusion 69

CHAPTER 5 ARTICLE 3 - DISJUNCTIVE CUTS IN MIXED-INTEGER CONIC OPTIMIZATION 72
5.1 Introduction 72
5.1.1 Disjunctive cuts: the MILP case 73
5.1.2 Disjunctive cuts: the MI-CONV case 74
5.1.3 Disjunctive cuts: the MI-CONIC case 75
5.1.4 Contribution and outline 76
5.2 Background 77
5.2.1 Cones and conic duality 78
5.2.2 Valid inequalities 80
5.2.3 Disjunctive inequalities 83
5.3 Cut separation 85
5.3.1 Separation problem 86
5.3.2 Membership problem 87
5.4 The roles of normalization 88
5.4.1 Alpha normalization 89
5.4.2 Polar normalization 92
5.4.3 Standard normalization 95
5.4.4 Trivial normalization 97
5.4.5 Uniform normalization 100
5.5 Separating conic-infeasible points 103
5.6 Cut lifting and strenghtening 106
5.6.1 Cut lifting 106
5.6.2 Cut strengthening 108
5.7 Computational results 110
5.7.1 Instances 110
5.7.2 Implementation details 111
5.7.3 Gap closed 113

5.7.4 Cut sparsity 119
5.8 Conclusion 121
5.8.1 What we have learned... 121
5.8.2 ... and what lies ahead 122

CHAPTER 6 GENERAL DISCUSSION 123

CHAPTER 7 CONCLUSION AND RECOMMENDATIONS 124
7.1 Summary of Works 124
7.2 Limitations and future research 125
7.2.1 Towards open and modular ecosystems for optimization 125
7.2.2 Integrating structure-exploiting techniques and Machine Learning . . 125

REFERENCES 127

APPENDICES 142
A.1 Dantzig-Wolfe decomposition 142
A.2 Column generation 143

LIST OF TABLES

Table 3.1 Devices classification and ownership rates 24
Table 3.2 MILP statistics for $T = 24$, $\xi^{\text{dep}} = 0.66$ 28
Table 3.3 CG statistics for $T = 24$, $\xi^{\text{dep}} = 0.66$ 28
Table 4.1 Results on the Mittelmann test set 56
Table 4.2 Column-generation statistics - DER instances 60
Table 4.3 Column-generation statistics - TSSP instances 61
Table 4.4 Performance comparison of IPM solvers - DER instances 66
Table 4.5 Performance comparison of IPM solvers - TSSP instances 67
Table 4.6 Problematic instances from the Mittelmann benchmark 70
Table 5.1 CGCP statistics for Example 4 and α normalization 91
Table 5.2 CGCP statistics for Example 4 and polar normalization 94
Table 5.3 CGCP statistics for Example 4 and standard normalization 97
Table 5.4 CGCP statistics for Example 4 and trivial normalization 101
Table 5.5 CGCP statistics for Example 4 and uniform normalization 104
Table 5.6 Instance statistics 111
Table 5.7 Gap closed and computing time - 10 rounds 115
Table 5.8 Gap closed and computing time - 200 rounds 115
Table 5.9 Cut statistics - 10 rounds 120
Table 5.10 Cut statistics - 200 rounds 120
Table B.1 Structured instances: performance comparison of IPM solvers 146

LIST OF FIGURES

Figure 3.1 Average computing times for CG and MILP formulations 29
Figure 3.2 Average number of CG iterations 29
Figure 3.3 Time spent solving the master problem and pricing sub-problems . . 30
Figure 3.4 Normalized HOEP and aggregated load 30
Figure 3.5 Average computing time and root gap 32
Figure 4.1 Code example for selecting a linear algebra implementation 54
Figure 4.2 Code example for choosing a specialized factorization 65
Figure 4.3 Code example for choosing a model's arithmetic 70
Figure 5.1 The three settings from Example 4, projected onto the $x_0 = 1$ space. The domain of the continuous relaxation is in gray, the split hull in orange, and the split disjunction is indicated in black. 89
Figure 5.2 Split cuts (in red) obtained with the α normalization. 90
Figure 5.3 Split cuts (in red) obtained with the polar normalization. 94
Figure 5.4 Split cuts (in red) obtained with the standard normalization. 96
Figure 5.5 Split cuts (in red) obtained with the trivial normalization. 100
Figure 5.6 Split cuts obtained with the uniform normalization. The continuous relaxation is in gray, the split hull in orange, and the obtained cut is in red. 103
Figure 5.7 Instance `flay02m`: gap closed per round (top) and time (bottom) . . 116
Figure 5.8 Instance `sssd-weak-15-4`: gap closed per round (top) and time (bottom). The polar normalization was terminated after two rounds. . . . 117
Figure 5.9 Instance `tls4`: gap closed per round (top) and time (bottom) 118

LIST OF SYMBOLS AND ACRONYMS

ADMM	Alternating Direction Method of Multipliers
CG	Column Generation
CGLP	Cut-Generating Linear Program
CGCP	Cut-Generating Conic Program
DER	Distributed Energy Resource
DR	Demand Response
HOEP	Hourly Ontario Energy Price
IPM	Interior-Point Method
LP	Linear Programming
MCP	Membership Conic Problem
MIP	Mixed-Integer Programming
MILP	Mixed-Integer Linear Programming
MIQP	Mixed-Integer Quadratic Programming
MI-CONV	Mixed-Integer Convex Programming
MI-CONIC	Mixed-Integer Conic Programming
MINLP	Mixed-Integer Nonlinear Programming
MIQCQP	Mixed-Integer Quadratically-Constrained Quadratic Programming
MISOCP	Mixed-Integer Second-Order Cone Programming
TSSP	Two-Stage Stochastic Program

LIST OF APPENDICES

Appendix A Dantzig-Wolfe decomposition and column generation 142
Appendix B Detailed results on structured LP instances 146

CHAPTER 1 INTRODUCTION

Mixed-Integer Programming (MIP) is routinely used in numerous fields, from routing and scheduling to power systems, inventory management and financial engineering. Two factors have contributed to this widespread adoption. First, MIP is a generic paradigm that can model a breadth of real-life problems, enabling its use in various applications. Second, the continuous development, for the past 70 years, of robust and ever-faster optimization software [1–3], combined with advances in computing resources, have made it possible to solve practical problems at scale. Nevertheless, progress in MIP software has been met by a likewise increase in the size and diversity of the problems that practitioners aim to solve, at times exceeding the capabilities of existing MIP technology.

On the one hand, generic solvers, building on decades on development, deliver the best overall performance across a wide range of problem classes: they are numerically robust and widely applicable, but sometimes run out of steam on particularly hard instances. On the other hand, practical needs have fostered the development of specialized solvers, that exploit specific information about a given class of problems. Such examples include Concorde [4] for the Traveling Salesman Problem (TSP), or Geosteiner [5] for computing Euclidean Steiner trees in a plane. Nevertheless, although specialized solvers typically yield enhanced performance, they are restricted to the class of problems for which they were designed. In most cases, their usability is further hampered by limited support and interfacing capabilities, as well as less extensive documentation.

Problem structure refers to intrinsic mathematical properties that are shared among a class of problems, e.g., problems with a separable objective, or with a block-angular constraint matrix. Thus, structure can be exploited at various levels, from stronger formulations down to linear algebra and memory management. Successfully doing so in practice then requires (i) a structure-exploiting algorithm, (ii) its integration within a solver's algorithmic and software environment, and (iii) the ability for the user to convey structural information, or for the solver to detect it automatically.

The present thesis focuses on (i) and (ii), and builds on the observation that structure-exploiting techniques can often be cast as specialized implementations of certain MIP algorithmic components. We explore a hybrid route between problem-specific and generic MIP solvers, wherein users can provide specialized code for each individual component of a generic MIP solver –beyond existing callback capabilities. Therefore, our work lies at the intersection of algorithmic aspects of MIP technology, and their implementation in optimization software.

1.1 Background

In its most general form, a MIP problem can be stated as

$$\min_{x} \quad f(x) \tag{1.1a}$$

$$s.t. \quad x \in \mathcal{X}, \tag{1.1b}$$

$$x_j \in \mathbb{Z}, \forall j \in \mathcal{J}, \tag{1.1c}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{J} \subseteq \{1, ..., n\}$. The optimal value of Problem (1.1) is

$$Z^* = \inf \{ f(x) \mid (1.1b) - (1.1c) \},$$

and a point $x \in \mathbb{R}^n$ is called a *feasible solution* if it satisfies the constraints (1.1b)-(1.1c). Mixed-Integer Linear Programming (MILP) corresponds to the case where f is linear and $\mathcal{X}$ is described by a finite number of linear inequalities, i.e., $\mathcal{X} = \{x \mid Ax \geq b\}$. In Mixed-Integer Convex Programming (MI-CONV), f is a convex function and $\mathcal{X}$ is a convex set, typically described by inequalities of the form $g(x) \leq 0$ where g is convex. The broader class of Mixed-Integer Nonlinear Programming (MINLP) includes cases where $\mathcal{X} = \{x \mid g(x) \leq 0\}$, and f, g are arbitrary.

Exact methods, which are the focus of this work, try to achieve two goals. On the primal side, one seeks an optimal solution x^*, i.e., a feasible solution with $f(x^*) = Z^*$. On the dual side, one tries to prove that x^* is indeed optimal, i.e., that there exist no feasible solution $\bar{x}$ with $f(\bar{x}) < f(x^*)$. Typically, this is done by computing a valid *dual bound* $\underline{Z} \leq Z^*$, whose value is improved until it reaches Z^*. Thus, optimality is declared when, up to numerical tolerances, $f(x^*) = \underline{Z}$.

The workhorse of exact MIP methods is the branch-and-bound algorithm [6]. In its simplest form, branch-and-bound is an implicit enumeration scheme, wherein one iteratively partitions the search space into smaller domains (branching), while regions that provably do not contain any optimal solution are discarded (bounding). All modern MIP solvers employ (sophisticated) variants of the branch-and-bound algorithm, combined with a clever use of presolve, cutting planes, and heuristics [2,7]. Presolve techniques [8] reformulate the original problem into a smaller, equivalent problem, by applying a number of simple rules, e.g., substituting fixed variables, removing redundant constraints, tightening bounds, etc. Heuristics [9] help find high-quality feasible solutions quickly. Cutting planes strengthen the continuous relaxation, thereby improving the quality of the dual bound and reducing the number of branch-and-bound nodes.

The effectiveness of MIP solvers stems from the combined use of each of these components, some of which can be specialized through callback functionalities. These are generally restricted to user-provided cuts, heuristic solutions, and possibly branching decisions. Lower-level changes, e.g., using a different LP algorithm, are impossible at worst, for instance when using commercial software, and daunting at best when source code can be modified. As a result, despite improving on one or few algorithmic components, ad-hoc implementations often lose out on several crucial tools such as powerful branching rules.

1.2 Objective and contributions

The objective of this thesis is to investigate the design and implementation of structure-exploiting algorithmic strategies within the generic MIP framework. Our motivating example stems from power systems operations, a field in which MIP formulations and conic relaxations are routinely employed [10].

Our first contribution focuses on the operation of power systems in the presence of numerous and heterogeneous Distributed Energy Resources (DERs), which we formulate as a large MIP problem. Motivated by scalability issues and privacy concerns for individual resources, and informed by an electricity grid's spatial structure, we propose a decomposition scheme and column-generation algorithm. The resulting approach can accommodate any type of resource whose operation can be formulated as a MIP problem, and is highly scalable, handling instances with thousands of DERs.

In our second contribution, we introduce the algorithmic design and implementation of Tulip, an open-source interior-point solver for linear optimization. Tulip's main feature is that its algorithmic framework is fully disentangled from linear algebra implementations, thus allowing to seamlessly integrate specialized routines for structured problems. Through computational experiments, we find that Tulip is competitive with open-source interior-point solvers on generic LP instances. Furthermore, we illustrate Tulip's flexibility and implement specialized linear algebra routines for structured master problems in the context of Dantzig-Wolfe decomposition. These routines yield a tenfold speedup on large and dense instances that arise in power systems operation and two-stage stochastic programming.

Our third and last contribution investigates the separation of disjunctive cuts in mixed-integer conic programming. Building on conic duality, we state an algebraic characterization of disjunctive inequalities, and formulate a cut-generating conic problem (CGCP) to separate them. Then, we compare the theoretical merits and demerits of several normalization conditions for the CGCP, thereby deriving a number of insights on cut separation in a nonlinear setting. In addition, we propose conic extensions of the classical lifting and strengthening procedures for disjunctive cuts. Finally, we study the practical behavior of various normalization conditions, and show that our CGCP-based approach is competitive with CPLEX internal lift-and-project cut generation.

1.3 Thesis outline

The rest of this thesis is organized as follows. In Chapter 2, we present a brief overview of existing literature on structure-exploiting techniques for MIP. Our three contributions outlined above are in Chapters 3, 4, 5, respectively; each is self-contained and contains relevant background material and references. Specifically, Chapter 3 reviews mathematical formulations and decomposition methods for the coordination of DERs, Chapter 4 presents structure-exploiting approaches in interior-point methods, and Chapter 5 includes background material on conic optimization and disjunctive cuts, both in the linear and nonlinear settings. Finally, Chapter 7 summarizes our work and discusses possible extensions.

CHAPTER 2 CRITICAL LITERATURE REVIEW

In this chapter, we provide background material and references that complement those found in the subsequent chapters. Given that structure exploitation encompasses a broad array of specialized algorithms, an exhaustive review of all such techniques is beyond the scope of this work. We start with a brief overview of existing ways in which generic MIP solvers can exploit structure. Then, we introduce decomposition techniques, which underlie Chapter 3 and Chapter 4, and available software. Background material and relevant references regarding conic optimization are presented in Section 5.2.

2.1 Structure exploitation in generic MIP solvers

Unsurprisingly, generic MIP solvers internally perform some (limited) structure detection, typically at presolve. These include the detection of disconnected components in Gurobi [8], and the automatic Benders decomposition in CPLEX [11]. Furthermore, major MILP solvers support certain types of constraints whose formulation do not always fall under the generic MIP paradigm. Examples include Special-Ordered Set constraints and indicator constraints, whose explicit knowledge can improve the resolution process as it enables the use of specific branching strategies.

Callbacks provide the ability to interact with a solver's resolution process, albeit in a controlled capacity. Supported features are generally limited to providing heuristics solutions and cutting planes, with a handful of solvers allowing branching decisions as well. Only limited information can be queried from a callback, e.g., the current fractional or integer solution, and bound information. Lower-level components are generally not available. Callback APIs and functionalities vary widely between solvers and, to the best of our knowledge, JuMP [12] is the only modeling language that offers (limited) support for callbacks, in a solver-agnostic way. These are restricted to submitting heuristic solutions and cutting planes, provided that the underlying solver supports such features.

2.2 Decomposition techniques and available software

Two of the most popular decomposition techniques in MIP are the Dantzig-Wolfe (DW) decomposition [13] and the Benders decomposition [14]. We only present the main principles behind each method, and refer to [15] and [14, 16] for a thorough overview of Dantzig-Wolfe decomposition and column generation, and of Benders decomposition, respectively.

2.2.1 Dantzig-Wolfe decomposition

The DW decomposition principle first partitions a problem's constraints into *linking* and *local* constraints. Then, the original variables are replaced by a combination of extreme points and extreme rays of the convex hull of the set defined by local constraints alone. This yields an extended formulation with fewer constraints, but more numerous –exponentially many in general– variables. The resulting problem is then solved by branch-and-price, a branch-and-bound algorithm where lower bounds at each node are computed by column generation.

The DW decomposition, and the corresponding branch-and-price scheme, are particularly advantageous when solving problems that involve the operation of multiple systems, each with its own constraints, whose operation is coupled by a set of linking constraints. In particular, the DW reformulation typically yields improved dual bounds and tends to alleviate a number of symmetry issues. Column-generation algorithms are routinely used in conjunction with DW decomposition, as well as in a number of routing and scheduling problems, see [15]. In particular, when local constraints comprise independent components, several opportunities arise to exploit parallelism within the method.

2.2.2 Benders decomposition

The Benders decomposition is the dual counterpart of the DW decomposition. In fact, when all variables are continuous, Benders decomposition is equivalent to DW decomposition applied to the problem's dual. The two methods differ when integer variables are present.

The Benders decomposition partitions the problem's variables into *linking* and *local* variables, such that fixing the values of the linking variables yields a somewhat simpler problem. Then, local variables are projected out, yielding a formulation with fewer variables but more numerous –typically exponentially many– constraints. The resulting problem can be solved by branch-and-cut, which makes implementing the Benders decomposition algorithm more straightforward using MIP solver's callback functionalities for cutting planes. A detailed description of the Benders decomposition in CPLEX is given in [11].

The Benders decomposition is often used in stochastic optimization, where it is sometimes referred to as the "L-shape" method. Similar to DW decomposition and column generation, parallelism can be exploited when the Benders sub-problem is separable.

2.2.3 Software

Several software packages exist that automate the decomposition process and/or solution algorithm. Their development is an indicator of the active research being carried out in this area, as well as of practical needs for such tools.

The commercial solver CPLEX [17] integrates an internal Benders decomposition scheme, whose implementation is described in [11]. Block structure can be provided by the user through model annotations, or it can be detected automatically. Substantial speedups are reported on classes of models that are known to benefit from Benders decomposition, most notably facility location problems. Another commercial product, SAS/OR, integrates work by Galati [18] to automate structured detection and the resolution process. The same ideas underlie the open-source Coin-OR DIP[1] framework [19].

The MIP solver SCIP [20], through the GCG plugin [21], supports both DW and Benders decomposition. Similar to CPLEX, the user can annotate the model to convey block structure to the solver, or structure can be detected automatically. Another academic code, BaPCod [22], also supports both DW and Benders decomposition through a user-provided structure annotation. This software is currently being translated into the open-source branch-and-price-and-cut framework Coluna [23]. Coluna is accessible through the modeling language JuMP, and structure information is conveyed by the JuMP extension `BlockDecomposition.jl`. Finally, the open-source DSP software [24], initially developed for the resolution of large stochastic integer programming problems, implements parallel decomposition schemes, including DW and Benders decomposition. DSP can also be accessed through JuMP, via the `StructJuMP.jl` extension, which is intended for block-structured optimization problems.

Although all these software packages employ similar representations and methodological tools, each comes with its own annotating scheme and interface. Nevertheless, extensions to algebraic modeling languages such as `StructJuMP`, `BlockDecomposition` or SML [25], may eventually ease the process of conveying block structure to decomposition-aware solvers.

[1]DIP was formerly known as DECOMP.

CHAPTER 3 ARTICLE 1 - A DECENTRALIZED FRAMEWORK FOR THE OPTIMAL COORDINATION OF DISTRIBUTED ENERGY RESOURCES

Authors: Miguel Anjos, Andrea Lodi and Mathieu Tanneau
Published in *IEEE Transactions on Power Systems*, [26]

Abstract Demand-response aggregators are faced with the challenge of how to best manage numerous and heterogeneous Distributed Energy Resources (DERs). This paper proposes a decentralized methodology for optimal coordination of DERs. The proposed approach is based on Dantzig-Wolfe decomposition and column generation, thus allowing to integrate any type of resource whose operation can be formulated within a mixed-integer linear program. We show that the proposed framework offers the same guarantees of optimality as a centralized formulation, with the added benefits of distributed computation, enhanced privacy, and higher robustness to changes in the problem data. The practical efficiency of the algorithm is demonstrated through extensive computational experiments, on a set of instances generated using data from Ontario energy markets. The proposed approach was able to solve all test instances to proven optimality, while achieving significant speed-ups over a centralized formulation solved by state-of-the-art optimization software.

3.1 Nomenclature

3.1.1 Parameters

$C_d^{\mathbf{th}}$	Thermal capacity of thermal load d.
$E_d^{\mathbf{bat,\ min}}$	Minimum state of charge of battery device d.
$E_d^{\mathbf{bat,\ max}}$	Maximum state of charge of battery device d.
$L_d^{\mathbf{uni}}$	Cycle duration of uninterruptible device d.
$\tilde{P}_{d,t}^{\mathbf{unc}}$	Power consumption of uncontrollable load d during time period t.
$\tilde{P}_{d,t}^{\mathbf{cur}}$	Power consumption of curtailable load d during time period t, in the absence of curtailment.
$\tilde{P}_{d,l}^{\mathbf{uni}}$	Power requirement of uninterruptible load d, during its cycle's phase l.

$P_d^{\textbf{th, min}}$	Minimum power consumption of thermal load d, when that device is on.
$P_d^{\textbf{th, max}}$	Maximum power consumption of thermal load d, when that device is on.
$P_d^{\textbf{ch, min}}$	Minimum charging power of battery device d.
$P_d^{\textbf{ch, max}}$	Maximum charging power of battery device d.
$P_d^{\textbf{dis, min}}$	Minimum discharging power of battery device d.
$P_d^{\textbf{dis, max}}$	Maximum discharging power of battery device d.
$P_r^{\textbf{min}}$	Minimum value of household r's net load.
$P_r^{\textbf{max}}$	Maximum value of household r's net load.
$P_a^{\textbf{min}}$	Minimum value of the aggregated residential net load.
$P_a^{\textbf{max}}$	Maximum value of the aggregated residential net load.
R	Number of resources.
T	Length of the time-horizon.
$\Delta\tau$	Duration of each time period.
$\eta_d^{\textbf{ch}}$	Charging efficiency of battery device d, in $[0, 1)$.
$\eta_d^{\textbf{dis}}$	Discharging efficiency of battery device d, in $[0, 1)$.
$\eta_d^{\textbf{th}}$	Thermal efficiency of thermal load d.
$\Theta_d^{\textbf{min}}$	Minimum temperature requirement of thermal load d.
$\Theta_d^{\textbf{max}}$	Maximum temperature requirement of thermal load d.
$\Theta_{d,t}^{\textbf{ext}}$	Outside temperature for thermal load d during time period t.
$\mu_d^{\textbf{th}}$	Thermal conductivity of thermal load d.
Π_t	Market price of electricity during time period t.

3.1.2 Sets

$\mathcal{D}_r^{unc}$	Set of uncontrollable loads of household r.
$\mathcal{D}_r^{cur}$	Set of curtailable loads of household r.
$\mathcal{D}_r^{uni}$	Set of uninterruptible loads of household r.
$\mathcal{D}_r^{def}$	Set of deferrable loads of household r.
$\mathcal{D}_r^{th}$	Set of thermal loads of household r.
$\mathcal{D}_r^{sto}$	Set of energy storage devices of household r.
$\mathcal{R}$	Set of resources $\mathcal{R} = \{1, ..., R\}$.
$\mathcal{T}$	Time horizon $\mathcal{T} = \{0, ..., T-1\}$.
$\mathcal{X}_r$	Set of feasible operational schedules of resource r.
Γ_r	Set of extreme rays of $conv(\mathcal{X}_r)$.
Ω_r	Set of extreme vertices of $conv(\mathcal{X}_r)$.

3.1.3 Variables

$e_{d,t}^{\mathbf{bat}}$	State of charge of battery device d at the end of time period t.
$p_{a,t}$	Net aggregated load during time period t.
$p_{d,t}$	Algebraic power consumption of device d during time period t.
$p_{d,t}^{\mathbf{ch}}$	Charging power of battery device d during time period t.
$p_{d,t}^{\mathbf{dis}}$	Discharging power of battery device d during time period t.
$p_{r,t}$	Net load of household r during time period t.
$u_{d,t}^{\mathbf{ch}}$	Binary variable that takes value 1 (resp. 0) if battery device d is charging (resp. not charging) during time period t.
$u_{d,t}^{\mathbf{dis}}$	Binary variable that takes value 1 (resp. 0) if battery device d is discharging (resp. not discharging) during time period t.
$u_{d,t}^{\mathbf{cur}}$	Binary variable that takes value 1 (resp. 0) if curtailable load d is on (resp. off) during time period t.

$u^{\mathbf{th}}_{d,t}$	Binary variable that takes value 1 (resp. 0) if thermal load d is on (resp. off) during time period t.
$v^{\mathbf{uni}}_{d,t}$	Binary variable that takes value 1 if uninterruptible load d's cycle is started at time t, and 0 otherwise.
$\mathbf{x}_r$	Vector of decision variables (operational schedule) of resource r.
$\mathbf{y}$	Vector of decision variables for the aggregator.
$\theta^{\mathbf{th}}_{d,t}$	Temperature of thermal load d during time period t.
λ	Vector of primal variables in the Master Problem.
π	Vector of dual variables (shadow prices).
σ	Vector of dual variables (shadow marginal costs).

3.2 Introduction

Smart grids hold the promise of more reliable and sustainable power grids, through the integration of communication technologies, advanced computation and intelligent controls. Among smart grid-enabled paradigms, Demand Response (DR) programs [27] enable end-users to dynamically adjust their electricity consumption, in response to price signals or incentives. The present work focuses on the DR potential of Distributed Energy Resources (DERs) that include, among others, flexible loads, distributed generation, and distributed energy storage [28].

Because of their small size, individual resources have a negligible marginal impact at the grid level. This has motivated the introduction of aggregators [29], that act as intermediaries between the grid and resources, thus enabling the latters to participate in traditional energy markets. Therefore, aggregators must address the challenges associated with coordinating numerous and heterogeneous resources, whose operation may involve discrete decisions. To that end, various coordination mechanisms have been investigated in the literature [30–32], and essentially break down into two main categories: price-based and incentive-based mechanisms.

In price-based mechanisms, price signals are communicated to users, either periodically, e.g. in Time-of-Use rates, or dynamically as in the case of Real-Time Pricing [32]. Users then adapt their consumption by optimizing their own individual objective, hopefully reducing electricity consumption when prices are higher. Therefore, the efficiency of a price-based DR mechanism is contingent on how price signals are computed and how users react to them. In [33–36], the DR problem is formulated as a Stackelberg game between an electricity retailer (leader) and electricity consumers (followers). This results in a bi-level problem wherein the retailer sets prices so as to maximize its own payoff, while accounting for consumers' optimal response to those prices. While authors in [34] and [36] focus on Time-Of-Use rates, Dynamic pricing is also considered in [33] and [35]. These works perform a reformulation of the bi-level program as a single-level Mixed-Integer Linear Program, which can then be solved using standard optimization software. Nevertheless, this approach suffers from poor scalability, and does not handle the presence of discrete variables in the consumers' operation. Another methodology, employed in [37–40], formulates the DR problem as a social welfare maximization problem. In [37] and [38], the total cost of electricity over a finite time-horizon is minimized, while authors in [39] and [40] consider individual utility functions for electricity consumers. All of [37–40] reduce to solving a convex (continuous) optimization problem, where dual information (i.e., Lagrange multipliers) is used to compute the price signals. However, this methodology does not apply when discrete variables are present, which is the setting we consider in this work.

On the other hand, incentive-based mechanisms reward users who are willing to participate in DR programs, for example by offering discounts on electricity bills [27], or on a per-event basis [41]. Incentive-based mechanisms include Direct Load Control schemes, which is the approach considered in this paper. Specifically, we focus on jointly coordinating the operation of a (large) number of DERs, so as to optimize a global objective. This can be formulated as a centralized optimization problem, as was explored in [42–45], wherein a central controller manages each individual resource. Although a centralized formulation offers the strongest optimality guarantees, it quickly becomes intractable when the number of resources increases, both due to memory requirements and the presence of discrete variables. Furthermore, the disclosure of private information by the resources raises privacy concerns, and puts additional burden on communication requirements.

Decentralized methods, on the other hand, distribute the computational effort among resources by leveraging local computing power. This typically results in better scalability, reduced communication overheads, and improved privacy. Several distributed heuristics were proposed, for example in [46–49], but provide weaker guarantees of optimality, since heuristic methods converge at best to a local optimum. Nevertheless, classical decomposition techniques allow to solve the centralized problem in a distributed way, thus offering the same guarantees of optimality, with the added benefit of decentralized computation. Therefore, in this work, we focus on exact, decomposition-based methods.

A large body of literature has focused on dual decomposition and Lagrangian-based methods. Dual decomposition yields a separable structure, which in turn enables distributed implementations. The related works [50–55] thus differ mainly in which algorithm is used to optimize the dual Lagrangian. In [50] and [51], the authors consider an augmented Lagrangian-based relaxation, which is formulated as a consensus problem and solved with the Alternating Direction Method of Multipliers (ADMM). However, ADMM does not handle discrete variables, that are used to model on-off constraints. In a similar fashion, standard Lagrangian relaxation is used in [52–55]. A classical sub-gradient algorithm is investigated in [52] and cutting-planes methods are studied in [53], but discrete variables were not considered. Although the works in [54] and [55] do consider mixed-integer variables, they rely on recovery heuristics to obtain feasible solutions. A bundle method is used in [54], while a double smoothing of the dual Lagrangian is applied in [55], allowing the use of a more efficient gradient-based algorithm. Overall, the main drawback of Lagrangian-based approaches is the recovery of feasible solutions when strong duality does not hold, which is generally the case for mixed-integer problems.

Alternatively, Dantzig-Wolfe (DW) decomposition-based approaches were investigated in [56–59]. In [56] and [57], DW decomposition is applied to demand-response problems related to peak-load management and, in [58], to the charging of electric vehicles. However, only a few types of devices were considered in these works, with no discrete variables involved. Finally, a column generation-based heuristic is used in [59] to schedule residential heating systems. Nevertheless, this heuristic approach does not provide a guarantee of optimality.

3.2.1 Contribution and outline

In this paper, we propose a scalable and exact methodology for the coordination of DERs whose operation involves discrete decisions, i.e., mixed-integer variables. Our proposed framework is based on DW decomposition and Column Generation, and has several advantageous features compared to existing approaches, which are either heuristic or poorly scalable. First, DW decomposition provides a systematic and efficient treatment of discrete variables. This yields a technology-agnostic formulation as shown in Section 3.4.1. Second, our column-generation algorithm naturally distributes among resources, which makes it highly scalable, as demonstrated in Section 3.5. Third, this decentralization offers enhanced privacy to the resources. Fourth, our formulation is robust to numerical changes in the problem data, with empirical evidence reported in Section 3.5.5.

The rest of the paper is organized as follows. In Section 3.3, we introduce operational models, and formulate the aggregation problem as a (centralized) Mixed-Integer Linear Program. We then present our methodology in Section 3.4, including the Dantzig-Wolfe reformulation in 3.4.1 and our column-generation algorithm in 3.4.2 and 3.4.3. Computational results are reported in Section 3.5, and Section 3.6 concludes the paper.

Unless specified otherwise, durations are expressed in hours (h), power and energy quantities in kilowatt (kW) and kilowatt-hour (kWh) respectively, and temperatures in degrees Celsius. Energy prices are given in dollars per kilowatt-hour ($/kWh).

3.3 Problem formulation

We consider a set of households that interact with an aggregator, in order to minimize the total cost of purchasing electricity from the grid. The aggregator and households are assumed to behave rationally, and households do not interact directly with each other. Finally, the environment is assumed to be deterministic. This last assumption is further discussed in Section 3.5.

Operational models are presented in 3.3.1 and 3.3.2, and the aggregation problem is formulated in 3.3.3.

3.3.1 Device constraints

Following the classification in [60], devices are grouped into six classes: uncontrollable loads (unc), curtailable loads (cur), uninterruptible loads (uni), deferrable loads (def), thermal loads (th), and energy storage (bat). The sequence $\mathbf{p}_d = (p_{d,0}, \cdots, p_{d,T-1})$ is called the device's load profile. A negative consumption, i.e., $p_{d,t} < 0$, indicates the device generates power.

Uncontrollable loads include devices whose operation cannot be altered. The load profile of such a device d is thus

$$p_{d,t} = \tilde{P}^{\text{unc}}_{d,t}. \qquad \forall t \in \mathcal{T} \tag{3.1}$$

Curtailable loads refer to devices whose operation at a given time t can be altered, independently of past and future operations. The operation of a curtailable load d is

$$p_{d,t} - u^{\text{cur}}_{d,t} \tilde{P}^{\text{cur}}_{d,t} = 0, \qquad \forall t \in \mathcal{T} \tag{3.2}$$

$$u^{\text{cur}}_{d,t} \in \{0, 1\}. \qquad \forall t \in \mathcal{T} \tag{3.3}$$

On-off curtailment is modelled through binary variables $u^{\text{cur}}_{d,t}$. Continuous curtailment is modelled by relaxing the integrality constraint on $u^{\text{cur}}_{d,t}$. This model can be extended to include several operation modes. Finally, a penalty is typically incurred when demand is curtailed. However, we do not explicitly formulate it here because, as will be detailed in Section 3.5, we do not consider in our experiments any demand that can be curtailed.

Uninterruptible loads are comprised of devices such as dishwashers and clothes dryers, whose operation is typically composed of a finite number of cycles. A cycle's start-up time is flexible but, once started, it cannot be interrupted. For simplicity, we consider a single cycle. The device's operation is then defined by

$$p_{d,t} - \sum_{l=0}^{L-1} v^{\text{uni}}_{d,t-l} \tilde{P}^{\text{uni}}_{d,l} = 0, \qquad \forall t \in \mathcal{T} \tag{3.4}$$

$$\sum_{t=0}^{T-L} v^{\text{uni}}_{d,t} = 1, \tag{3.5}$$

$$v^{\text{uni}}_{d,t} \in \{0,1\}, \qquad \forall t \in \mathcal{T} \tag{3.6}$$

where constraint (3.5) ensures the cycle is started exactly once, and $v^{\text{uni}}_{d,t} = 0, \forall t < 0$. This framework naturally extends to several cycles with precedence constraints as proposed in [61].

Deferrable loads include devices such as an electric vehicle's (EV) charger, whose consumption may be shifted earlier or later in time. Deferrable loads are modelled as follows:

$$E^{\text{def, min}}_{d} \leq \Delta\tau \sum_{t\in\mathcal{T}} p_{d,t} \leq E^{\text{def, max}}_{d}, \tag{3.7}$$

$$u^{\text{def}}_{d,t} P^{\text{def, min}}_{d} \leq p_{d,t} \leq u^{\text{def}}_{d,t} P^{\text{def, max}}_{d}, \qquad \forall t \in \mathcal{T} \tag{3.8}$$

$$u^{\text{def}}_{d,t} \in \{0,1\}. \qquad \forall t \in \mathcal{T} \tag{3.9}$$

On-off constraints are modelled by (3.8)-(3.9).

Thermal loads encompass devices like space heaters and air conditioners, that aim at keeping a system's (e.g., a room) temperature within a certain range. Their operation is modelled by

$$\Theta^{\min}_{d} \leq \theta^{\text{th}}_{d,t} \leq \Theta^{\max}_{d}, \qquad \forall t \in \mathcal{T} \tag{3.10}$$

$$\frac{\mu^{\text{th}}_{d}}{C^{\text{th}}_{d}}(\Theta^{\text{ext}}_{d,t} - \theta^{\text{th}}_{d,t}) + \frac{\eta^{\text{th}}_{d}}{C^{\text{th}}_{d}} p_{d,t} = \frac{\theta^{\text{th}}_{d,t+1} - \theta^{\text{th}}_{d,t}}{\Delta\tau}, \qquad \forall t \in \mathcal{T} \tag{3.11}$$

$$u^{\text{th}}_{d,t} P^{\text{th, min}}_{d} \leq p_{d,t} \leq u^{\text{th}}_{d,t} P^{\text{th, max}}_{d}, \qquad \forall t \in \mathcal{T} \tag{3.12}$$

$$u^{\text{th}}_{d,t} \in \{0,1\}. \qquad \forall t \in \mathcal{T} \tag{3.13}$$

The evolution of the system's temperature in (3.11) is given by the first-order approximation of a thermodynamic model [62]. The device's on-off constraints are modelled in (3.12)-(3.13).

Energy storage devices can store energy and release it later. For simplicity, we consider the case of batteries, for which an operational model (adapted from [63]) is

$$E_d^{\text{bat, min}} \leq e_{d,t}^{\text{bat}} \leq E_d^{\text{bat, max}}, \qquad \forall t \in \mathcal{T} \tag{3.14}$$

$$p_{d,t} - p_{d,t}^{\text{ch}} + p_{d,t}^{\text{dis}} = 0, \qquad \forall t \in \mathcal{T} \tag{3.15}$$

$$\Delta\tau \left(\eta_d^{\text{ch}} p_{d,t}^{\text{ch}} - \frac{1}{\eta_d^{\text{dis}}} p_{d,t}^{\text{dis}} \right) = e_{d,t}^{\text{bat}} - e_{d,t-1}^{\text{bat}}, \qquad \forall t \in \mathcal{T} \tag{3.16}$$

$$u_{d,t}^{\text{ch}} P_d^{\text{ch, min}} \leq p_{d,t}^{\text{ch}} \leq u_{d,t}^{\text{ch}} P_d^{\text{ch, max}}, \qquad \forall t \in \mathcal{T} \tag{3.17}$$

$$u_t^{\text{dis}} P_d^{\text{dis, min}} \leq p_{d,t}^{\text{dis}} \leq u_{d,t}^{\text{dis}} P_d^{\text{dis, max}}, \qquad \forall t \in \mathcal{T} \tag{3.18}$$

$$u_{d,t}^{\text{ch}} + u_{d,t}^{\text{dis}} \leq 1, \qquad \forall t \in \mathcal{T} \tag{3.19}$$

$$u_{d,t}^{\text{ch}}, u_{d,t}^{\text{dis}} \in \{0, 1\}. \qquad \forall t \in \mathcal{T} \tag{3.20}$$

The internal dynamics of the battery are captured by (3.16), and on-off constraints (3.17)-(3.20) ensure that the battery cannot be simultaneously charged and discharged. This model can be further extended to include constraints on ramping and cycling, see, e.g. [63].

Finally, renewable generation can be modelled as a negative load. Depending on systems' specifications, it may be either uncontrollable or curtailable.

3.3.2 Household and aggregator constraints

We now consider a given household $r \in \mathcal{R}$. Additional constraints at the household level are formulated as

$$P_r^{\min} \leq p_{r,t} \leq P_r^{\max}, \qquad \forall t \in \mathcal{T} \tag{3.21}$$

$$p_{r,t} = \sum_{d \in \mathcal{D}_r} p_{d,t}. \qquad \forall t \in \mathcal{T} \tag{3.22}$$

corresponding to that house's circuit breaker's operating range.

Similarly, the aggregator must ensure that the aggregated load $p_{a,t}$ remains within physical limitations, i.e.,

$$P_a^{\min} \leq p_{a,t} \leq P_a^{\max}, \qquad \forall t \in \mathcal{T} \tag{3.23}$$

$$p_{a,t} = \sum_{r \in \mathcal{R}} p_{r,t}. \qquad \forall t \in \mathcal{T} \tag{3.24}$$

If households were operated independently, i.e., without interacting with the aggregator, constraints (3.23) may be violated. In practice, this could result in equipment damage, or localised blackouts.

3.3.3 Aggregation problem

The aggregation problem consists here in minimizing the total cost of purchasing energy from the grid, while satisfying all operational constraints. It is formulated as the following Mixed-Integer Linear Program (MILP):

$$\begin{aligned}
\min \quad & \sum_{t \in \mathcal{T}} \Pi_t \times \Delta\tau \times p_{a,t} && (3.25)\\
\text{s.t.} \quad & (3.1), \quad \forall r, \forall d \in \mathcal{D}_r^{unc}\\
& (3.2) - (3.3), \quad \forall r, \forall d \in \mathcal{D}_r^{cur}\\
& (3.4) - (3.6), \quad \forall r, \forall d \in \mathcal{D}_r^{uni}\\
& (3.7) - (3.9), \quad \forall r, \forall d \in \mathcal{D}_r^{def}\\
& (3.10) - (3.13), \quad \forall r, \forall d \in \mathcal{D}_r^{th}\\
& (3.14) - (3.20), \quad \forall r, \forall d \in \mathcal{D}_r^{sto}\\
& (3.21) - (3.22), \quad \forall r\\
& (3.23) - (3.24).
\end{aligned}$$

We now introduce a more general and compact notation for the aggregation problem (3.25). For each household r, let $\mathbf{x}_r$ denote its vector of decision variables, i.e., the vector obtained by concatenating all decision variables specific to that household. Here, $\mathbf{x}_r$ would be composed of a household's net load, plus the decision variables of each device in that household. In what follows, we refer to $\mathbf{x}_r$ as the operational schedule of household r. Define $\mathcal{X}_r$ as the set of all feasible operational schedules for household r, so that operational constraints for that household are written in the compact form $\mathbf{x}_r \in \mathcal{X}_r$.. Therefore, $\mathcal{X}_r$ is defined by a finite number of linear constraints and integrality requirements, so that $conv(\mathcal{X}_r)$ is a polyhedron. Finally, a household's operating cost is written $c_r^T x_r$ for a given cost vector c_r. In the present case, we have $c_r = 0$ for every r, since no household-specific cost is considered. Nevertheless, we keep the objective term c_r in the formulation for generalization purposes.

Similarly, let $\mathbf{y}$ denote the vector of decision variables that are specific to the aggregator. For the case at hand, $\mathbf{y}$ corresponds to the aggregated load profile $\mathbf{p}_a$. The aggregator's operating cost is written $q^T\mathbf{y}$, while constraints (3.23)-(3.24) are written $M\mathbf{y} + \sum_r A_r\mathbf{x}_r = b$, without loss of generality.

The aggregation problem can therefore be written in the general compact form

$$\min_{\mathbf{y},\mathbf{x}_1,\dots,\mathbf{x}_R} \quad q^T\mathbf{y} + \sum_{r\in\mathcal{R}} c_r^T\mathbf{x}_r \tag{3.26}$$

$$\text{s.t.} \quad M\mathbf{y} + \sum_{r\in\mathcal{R}} A_r\mathbf{x}_r = b \tag{3.27}$$

$$\mathbf{x}_r \in \mathcal{X}_r. \qquad \forall r \tag{3.28}$$

Constraints (3.27) induce a coupling between the households, and are thus referred to as linking constraints. Conversely, constraints (3.28) are separable by household, and are referred to as local constraints. We emphasize that this formulation is not restricted to the aforementioned loads, but allows to integrate any type of DER whose operation can be formulated within a MILP.

Mixed-integer linear programs such as (3.26)-(3.28) can be solved to proven optimality using standard optimization software. Although MILPs are NP-hard in general, practical instances can often be solved efficiently. Indeed, several of our test instances, with up to hundreds of thousands of variables, were solved to optimality in a few minutes with a state-of-the art solver. Nevertheless, a centralized formulation obviously becomes intractable when dealing with large systems, both due to memory requirements and the presence of discrete variables. Furthermore, as will be demonstrated in Section 3.5.5, the nature of the DERs may have a significant impact on the formulation's integrality gap, and therefore on computing time.

3.4 Distributed Column-Generation framework

We now present a decentralized framework for solving the aggregation problem to global optimality. To underline the generality of the proposed methodology, we use the notation introduced in Section 3.3.3.

3.4.1 Dantzig-Wolfe decomposition

We begin by introducing a Dantzig-Wolfe reformulation of the aggregation problem (3.26)-(3.28). First, for each resource $r \in \mathcal{R}$, let Ω_r (resp. Γ_r) denote the set of extreme vertices (resp. extreme rays) of $conv(\mathcal{X}_r)$. Both sets are well-defined and finite since, as mentioned in Section 3.3.3, $conv(\mathcal{X}_r)$ is a polyhedron. For simplicity, we assume that $\mathcal{X}_r$ is bounded, so that $\Gamma_r = \emptyset$.

We then apply the Dantzig-Wolfe decomposition principle [13] to the aggregation problem (3.26)-(3.28), which yields the extended formulation

$$\begin{aligned}
\min_{\mathbf{y},\lambda} \quad & q^T\mathbf{y} + \sum_{r\in\mathcal{R},\omega\in\Omega_r} c_{r,\omega}\lambda_{r,\omega} && && (3.29)\\
\text{s.t.} \quad & M\mathbf{y} + \sum_{r\in\mathcal{R},\omega\in\Omega_r} \lambda_{r,\omega}a_{r,\omega} = b, && && (3.30)\\
& \sum_{\omega\in\Omega_r} \lambda_{r,\omega} = 1, && \forall r && (3.31)\\
& \lambda \geq 0, && && (3.32)\\
& \sum_{\omega\in\Omega_r} \lambda_{r,\omega}\omega = \mathbf{x}_r, && \forall r && (3.33)\\
& \mathbf{x}_r \in \mathcal{X}_r, && \forall r && (3.34)
\end{aligned}$$

with the notation $c_{r,\omega} = c_r^T\omega$ and $a_{r,\omega} = A_r\omega$. The objective (3.29) and linking constraints (3.30) are simply re-writing of (3.26) and (3.27), respectively. Thereby, each extreme point $\omega \in \Omega_r$ is associated to a variable $\lambda_{r,\omega}$, and to a column $a_{r,\omega}$ that corresponds to a load schedule for resource r. The extended formulation (3.29)-(3.34) is equivalent to the compact formulation (3.26)-(3.28), and is most typically solved using a branch-and-price algorithm. In what follows, we focus on the linear relaxation of the extended formulation, which is given by (3.29)-(3.32) and referred to as the Master Problem (MP).

Unlike the centralized formulation (3.26)-(3.28), the structure of the MP is independent of the nature of the resources. Indeed, the linking constraints only involve variables corresponding to a resource's net load. In particular, the MP is always a (continuous) linear program, even if a resource's operation involves discrete variables. Therefore, using DW decomposition yields a technology-agnostic formulation. This feature is highly advantageous, since it allows to integrate new types of resources without any major impact on performance.

3.4.2 Distributed column generation

Since the MP contains exponentially many variables, it is solved by a Column-Generation (CG) algorithm. We refer to [15] for a thorough overview of column generation and branch-and-price algorithms, as well as the relation between Lagrangian relaxation and Dantzig-Wolfe decomposition.

Consider the Restricted Master Problem (RMP)

$$\min_{\mathbf{y},\lambda} \quad q^T y + \sum_{r \in \mathcal{R}, \omega \in \bar{\Omega}_r} c_{r,\omega} \lambda_{r,\omega} \tag{3.35}$$

$$\text{s.t.} \quad M\mathbf{y} + \sum_{r \in \mathcal{R}, \omega \in \bar{\Omega}_r} \lambda_{r,\omega} a_{r,\omega} = b, \tag{3.36}$$

$$\sum_{\omega \in \bar{\Omega}_r} \lambda_{r,\omega} = 1, \qquad \forall r \tag{3.37}$$

$$\lambda \geq 0, \tag{3.38}$$

where, for each resource r, $\bar{\Omega}_r \subset \Omega_r$ denotes the subset of columns that are currently considered. The RMP is initialized with a small number of columns, some of which may be artificial to ensure feasibility.

At the beginning of each iteration, the RMP is solved to optimality. Let π denote the vector of dual variables associated to linking constraints (3.36), and σ_r the dual variable associated to convexity constraint (3.37) for resource r. For given r and $\omega \in \Omega_r$, the reduced cost of $\lambda_{r,\omega}$ is then

$$\bar{c}_{r,\omega} = c_r^T \omega - \pi^T A_r \omega - \sigma_r. \tag{3.39}$$

Therefore, a variable λ_{r,ω^*} with smallest reduced cost is given by the pricing step

$$\omega^* \in \arg\min_{\omega \in \Omega_r} \left(c_r^T \omega - \pi^T A_r \omega - \sigma_r \right). \tag{3.40}$$

However, explicitly iterating over all Ω_r is prohibitively expensive. Nevertheless, since each $\omega \in \Omega_r$ is an extreme point of $\text{conv}(\mathcal{X}_r)$, performing the pricing step (3.40) is equivalent to solving

$$(SP_r) \quad \omega^* \in \arg\min_{\mathbf{x}_r} \quad (c_r^T - \pi^T A_r)\mathbf{x}_r - \sigma_r \tag{3.41}$$

$$\text{s.t.} \quad \mathbf{x}_r \in \mathcal{X}_r. \tag{3.42}$$

The pricing sub-problem (3.41)-(3.42) is a small MILP, which we assume can be solved efficiently. If the identified variable λ_{r,ω^*} has negative reduced cost, i.e., $\bar{c}_{r,\omega^*} < 0$, it is added to the RMP. Otherwise, $\bar{c}_{r,\omega^*} \geq 0$ and all variables $\lambda_{r,\omega}, \omega \in \Omega_r$ have non-negative reduced cost. Optimality in the MP is reached when this is the case for all resources, i.e., all variables in the master problem have non-negative reduced cost. The pseudo-code of our CG algorithm is given in Algorithm 1.

```
Input: Initial RMP
 1: while stopping criterion not met do
 2:     Solve RMP and obtain optimal dual variables $(\pi, \sigma)$
 3:     // Pricing step (distributed)
 4:     for all $r \in \mathcal{R}$ do
 5:         Solve $SP_r$ with the query point $(\pi, \sigma_r)$, and obtain $\omega^*$ such that $\lambda_{r,\omega^*}$ has most negative
            reduced cost
 6:         if $\bar{c}_{r,\omega^*} < 0$ then
 7:             Add corresponding column to the RMP
 8:         end if
 9:     end for
10:     // Stopping criterion
11:     if no column added to RMP then
12:         STOP
13:     end if
14: end while
```

Algorithm 1 Column-Generation algorithm

All the sub-problems are independent, and therefore can be solved in a decentralized fashion. This offers greater privacy to the resources. Indeed, each sub-problem SP_r is solved locally by resource r. Consequently, the aggregator only has limited information about each resource's operation, in the form of the columns that are generated. Furthermore, this decentralized setting is more robust to communication failure. If a resource r fails to communicate with the aggregator, a locally feasible operation can always be used, e.g. the last column that was generated. Similarly, the aggregator and other resources can carry on the optimization process, considering for resource r only the last column it generated. If communication is later restored, the quality of the final solution will not be affected, although the total number of iterations may obviously increase.

3.4.3 Branch-and-price

The MP is the root node of the branching tree in a branch-and-price algorithm. Solving the MP yields a primal optimal solution $(\bar{y}, \bar{\lambda})$ and using it, we define, for each $r \in \mathcal{R}$

$$\bar{\mathbf{x}}_r := \sum_{\omega \in \Omega_r} \bar{\lambda}_{r,\omega} \omega. \tag{3.43}$$

Since the MP is a relaxation of the extended formulation, the $\bar{\mathbf{x}}_r$ are generally fractional. For example, $\bar{\mathbf{x}}_r$ may represent an operational schedule where a battery is charging and discharging simultaneously, which would violate constraint (3.20). An optimal integer solution is obtained by branch-and-price.

We implemented a heuristic diving procedure wherein several branching decisions are taken simultaneously. At each successive branching node, a random subset of resources $\tilde{\mathcal{R}}$ is selected. Then, for each $r \in \tilde{\mathcal{R}}$, a feasible $\tilde{\mathbf{x}}_r$ is computed by projecting the current fractional solution $\bar{\mathbf{x}}_r$ onto the feasible set $\mathcal{X}_r$. This is done by solving the MILP

$$\tilde{\mathbf{x}}_r \in \arg\min_{\mathbf{x}_r} \quad \|\mathbf{x}_r - \bar{\mathbf{x}}_r\|_\infty \tag{3.44}$$

$$\text{s.t.} \quad \mathbf{x}_r \in \mathcal{X}_r. \tag{3.45}$$

Branching decisions are taken by fixing all integer variables to their value in $\tilde{\mathbf{x}}_r$, and are enforced in the sub-problem SP_r directly. The column corresponding to $\tilde{\mathbf{x}}_r$ is added to the RMP, and all conflicting columns, i.e., columns that do not satisfy the branching decisions, are removed from the formulation. Enforcing branching decisions in the sub-problems results in no conflicting column being generated afterwards. Finally, the relaxation of the newly obtained node is solved by column generation, and the procedure is repeated until all integer variables are fixed. The pseudo-code of this diving procedure is given in Algorithm 2.

```
Input: Solution $\bar{\mathbf{x}}$ of the MP, $\bar{\mathcal{R}} := \mathcal{R}$
1: while stopping criterion not met do
2:     // Branching decisions
3:     Select a random subset of resources $\tilde{\mathcal{R}} \subset \bar{\mathcal{R}}$
4:     for all $r \in \tilde{\mathcal{R}}$ do
5:         Compute $\tilde{\mathbf{x}}_r$ and add corresponding column to the RMP
6:         Remove conflicting columns
7:         Enforce branching decisions in $SP_r$
8:     end for
9:     Solve node relaxation using Algorithm 1, obtain new $\bar{\mathbf{x}}$
10:    $\bar{\mathcal{R}} := \bar{\mathcal{R}} - \tilde{\mathcal{R}}$
11:    // Stopping criterion
12:    if $\bar{\mathcal{R}} = \emptyset$ then
13:        STOP
14:    end if
15: end while
```

Algorithm 2 Diving heuristic procedure

This heuristic was run right after solving the MP and, in all test instances, it was able to find what turned out to be an optimal integer solution. More precisely, a feasible integer solution was always found by the heuristic, and its optimality was proven using the lower bound provided by the MP. Therefore, no branching was needed, which we emphasize may not be true in general.

3.5 Computational results

We now report computational results for the proposed column-generation algorithm (CG), using the centralized formulation (MILP) as a baseline. Tests are carried out on a set of 1120 instances, generated using data from Ontario energy markets. We first describe our test methodology in 3.5.1 and 3.5.2. Numerical results are analysed in 3.5.3 and 3.5.4, and the robustness of the formulation is further assessed in 3.5.5.

For reproducibility, we make our code for generating instances and numerical data publicly available[1].

[1]https://github.com/mtanneau/DER_instances

3.5.1 Numerical instantiation

For all simulations, the time-horizon begins at 5am Monday, January 18^{th} 2016, and one time-period is always $\Delta\tau = 1\text{h}$. We consider four different values for T and seven for R, which are indicated in Figure 3.1 and Table 3.2, respectively. Ontario's provincial load, production and pricing data are obtained from [64].

The set of considered devices and their ownership rates are given in Table 3.1 (adapted from [60]). The ownership rate of a device is interpreted as the probability that that device be present in a given household. Devices that are not explicitly considered in this work are aggregated into one uncontrollable load for each household. In addition, we consider correlation among devices. It is assumed that only households with a clothes washer own a clothes dryer. Because available data regarding the penetration of EVs, batteries and rooftop solar in Ontario is very scarce, we assumed that households own either none or the three of them. Several deployment scenarios are considered, corresponding to ownership rates $\xi^{\text{dep}} = 0\%, 33\%, 66\%$ and 100%.

Dishwashers, clothes washers and clothes dryers are modelled as uninterruptible loads. Each appliance's operation consists of one cycle per day, with a constant power consumption of 1kW. Cycles' durations are 2h for dishwashers, 2h for clothes washers, and 3h for clothes dryers.

For electric heating systems, the outside temperature Θ^{ext} is obtained by perturbing a reference profile Θ^{ref} that consists of hourly readings from a weather station in the Toronto area [65]

$$\Theta^{\text{ext}}_{d,t} = \Theta^{\text{ref}}_{t} + 0.5 \times \varepsilon_{d,t}, \qquad \forall t \in \mathcal{T}$$

Table 3.1 Devices classification and ownership rates

Device	Classification	Own. rate (%)
Dishwasher	Uninterruptible load	65
Clothes washer	Uninterruptible load	90
Clothes dryer	Uninterruptible load	75
Electric heating	Thermal load	60
Electric vehicle	Deferrable load	ξ^{dep}
Home battery	Energy storage	ξ^{dep}
Rooftop solar	Curtailable load	ξ^{dep}
Others	Uncontrollable load	100

with $\varepsilon \sim \mathcal{N}(0,1)$. Numerical parameters for the thermodynamics model are identical among households: $\eta = 1$, $\mu = 0.2$, $C = 3$, and $P^{\text{th, min}} = 0$, $P^{\text{th, max}} = 10$. The inside temperature must be kept in the range $[\Theta_d^{\min}, \Theta_d^{\max}] = [18, 22]$.

The charging of electric vehicles is modelled as a deferrable load with a daily energy requirement of $E_d^{\min}, E_d^{\max} = 10$ as reported in [60]. Charging must happen between 8pm and 5am, and charger limitations are $P_d^{\min} = 1.1$, $P_d^{\max} = 7.7$.

Battery specifications are based on the Tesla powerwall [66]. Energy capacity is $E_d^{\text{bat, max}} = 13.5$, with minimum state of charge $E_d^{\text{bat, min}} = 0$. Charging and discharging limitations are $P_d^{\text{ch, min}}, P_d^{\text{dis, min}} = 0$ and $P_d^{\text{ch, max}}, P_d^{\text{dis, max}} = 5$. Efficiencies are $\eta_d^{\text{ch}}, \eta_d^{\text{dis}} = 0.95$, yielding a 90% round-trip efficiency.

Rooftop solar is modelled as a (negative) curtailable load. The output of a PV system d is given by

$$\tilde{P}_{d,t}^{\text{PV}} = \gamma_r \times \tilde{Q}_t^{\text{PV}} \times \zeta_{d,t}, \qquad \forall t \in \mathcal{T}$$

where the normalized output $\tilde{Q}_t^{\text{PV}}$ is Ontario's hourly PV output Q_t^{PV}, divided by its average value over the considered time period. The household-specific scaling factor γ_r is drawn from a uniform distribution $\mathcal{U}(0.5, 1.5)$, and $\zeta \sim \mathcal{U}(0,1)$ is a random noise.

For each household r, the corresponding uncontrollable load d has the following load profile:

$$\tilde{P}_{d,t} = \gamma_r \times \max\left(0, \tilde{Q}_t^{\text{Ont}} + 0.05 \times \varepsilon_{d,t}\right), \qquad \forall t \in \mathcal{T}$$

where $\tilde{Q}^{\text{Ont}}$ is the normalized Ontario hourly provincial load, and $\varepsilon \sim \mathcal{N}(0,1)$ is a white noise.

Finally, we set $P_r^{\min} = 0$, $P_r^{\max} = 10$ for the households' net load constraints. Similarly, the total aggregated load is bounded by $P_a^{\min} = 0$ and $P_a^{\max} = 7.5 \times R$. The price of electricity Π_t is the Hourly Ontario Energy Price (HOEP).

In practice, since neither uncontrollable loads, nor outside temperature, nor PV output, are deterministic, one may use forecasts for these quantities instead. How to obtain such forecasts is beyond the scope of this paper. To handle forecast errors, a receding horizon scheme can be employed, wherein forecasts are updated periodically. When forecasts are updated, e.g. every hour, the aggregation problem is updated accordingly and solved again over the entire time-horizon, e.g. the next 24h. We emphasize that the performance of our algorithm makes such an implementation tractable.

3.5.2 Implementation details

Experiments were performed on a 2×Xeon E5-2650V4 2.2Ghz, 256GB RAM computer running Linux. Our implementation was coded in Python 2.7, with CPLEX 12.7 as the linear solver. All CPLEX runs used default parameters and a single thread. Accordingly, an instance is considered solved to optimality when the reported optimality gap is smaller than 10^{-4}, which is the default threshold for CPLEX.

In order to smooth out performance variations, we generated ten different instances for each combination of (R, T, ξ^{dep}). This resulted in a testbed of $(4{\times}7{\times}4){\times}10{=}1120$ instances, and we compare the results obtained by the proposed method (CG), and the centralized formulation (MILP). For the MILP, we used CPLEX with default parameters, a single thread and a time limit of one hour. All instances were found to be feasible. Due to limited computing resources, the column-generation sub-problems were solved serially rather than in parallel. Nevertheless, the duration of each iteration in the distributed setting is given by the RMP computing time, plus the maximum solving time among sub-problems, plus computation overheads. We emphasize that, in practice, each sub-problem would indeed be solved locally by the corresponding household.

Finally, the RMP is initialized by computing, for each resource, a column corresponding to a minimum-peak load schedule. In order to ensure feasibility in the RMP, artificial slack and surplus variables are added to the linking constraints. These artificial variables implement an l_1 penalty with sufficiently large cost, and are thus automatically set to zero once a feasible solution is found. Furthermore, we used a partial pricing strategy that consists in adding at most $0.1 \times R$ columns to the master at each iteration. The 0.1 ratio was found to achieve good performance across a wide range of instances. Similarly, for the recovery heuristic, a random 10% of the resources are selected at each step.

3.5.3 Performance analysis

Performance statistics are presented for MILP and CG in Tables 3.2 and 3.3 respectively. Since both methods behaved consistently across the different values of ξ^{dep}, we only report results for the case $\xi^{\text{dep}} = 0.66$. For the MILP formulation, Table 3.2 reports the number of binary and continuous variables (Bin. and Cont. respectively, in millions) of the problem, the proportion of instances solved to proven optimality, total and root computing times (in seconds), the number of nodes in the branch-and-bound tree, and the root gap (in %). For the CG method, Table 3.3 reports the number of columns generated (Col., in thousands), the total number of CG iterations to reach optimality (Iter.), the proportion of instances solved to proven optimality, computing times (in seconds), and the root gap (in %). Root gaps are given by

$$\text{gap} = \frac{|z^* - \underline{z}|}{|z^*|},$$

where z^* is the value of the best known integer solution found by either algorithm, and $\underline{z}$ is the value of the root relaxation. Finally, all reported averages are geometric means.

As expected, CPLEX solved the MILP formulation for smaller instances, but systematically reached the time limit for the larger ones. More specifically, CPLEX failed to find a feasible solution for 162 instances, roughly corresponding to the instances with more than two million binary variables. Note that the quality of the lower bound for MILP is not an issue here. On the opposite, the solver's capability of exploring the branch-and-bound nodes in a reasonable amount of time, in order to provide good feasible solutions, is affected by the problem's size. In comparison, CG was able to solve all 1120 instances to proven optimality. Furthermore, computing times for both methods are displayed in Figure 3.1. For all values of T, CG exhibits a more scalable behaviour than MILP, and achieves speed-ups of up to two orders of magnitude. These results confirm that, even if it may be tractable for small-size systems, a centralized approach fails to handle large numbers of resources.

To further analyse the scalability of CG, Figure 3.2 shows the number of CG iterations to reach convergence, corresponding to the number of times the RMP is solved. Here, the number of iterations is more relevant than raw computation times, since the latter depends on machine specifications and on the solver's performance. On one hand, an increase in the length of the time-horizon T leads, as expected, to an increase in the number of iterations. On the other hand, as was hinted at in Table 3.3, the number of iterations appears to be independent of the number of households. This remarkable behaviour is due to similarities between sub-problems. This explanation is corroborated by the fact that larger values of ξ^{dep}, which only affects the distribution of households, and thus of sub-problems, resulted in more iterations. Overall, CG is sensitive to the distribution of resources, rather than their number.

Finally, the proportions of time spent solving the master problem and sub-problems, respectively, are displayed in Figure 3.3. Computation overheads can be significant, but they are highly dependent on the implementation. Therefore, we factor them out of the plots. Clearly, as the number of resources increases, most of the time is spent solving the RMP. Indeed, since sub-problems are solved in parallel, the number of resources has little influence on the duration of the pricing step. This is consistent with computing times reported in

Table 3.2 MILP statistics for $T = 24$, $\xi^{\text{dep}} = 0.66$

R	Variables (M)		Solved	Time (s)		B&B	Root
	Bin.	Cont.	(%)	Root	Total	nodes	% gap
1024	0.13	0.19	100	9.6	40.1	1.5	0.00
1536	0.19	0.29	100	20.6	132.9	11.5	0.00
2048	0.26	0.39	100	30.2	291.0	108.1	0.00
3072	0.39	0.57	100	65.4	649.8	394.4	0.00
4096	0.52	0.77	100	147.3	1263.7	562.8	0.00
6144	0.78	1.15	70	314.8	2907.1	550.1	0.00
8192	1.04	1.54	60	534.6	2937.4	664.3	0.00

Table 3.3 CG statistics for $T = 24$, $\xi^{\text{dep}} = 0.66$

R	Col.	Iter.	Solved	Time (s)			Root
	(k)		(%)	Master	Pricing	Total	% gap
1024	5.3	45.7	100	0.6	1.7	3.8	0.00
1536	8.0	46.0	100	1.1	1.7	5.1	0.00
2048	10.6	46.1	100	1.8	1.8	6.8	0.00
3072	16.2	46.6	100	3.8	1.9	10.6	0.00
4096	21.6	47.2	100	6.4	2.0	15.0	0.00
6144	31.7	46.3	100	13.5	2.0	25.2	0.00
8192	43.4	47.6	100	25.1	2.1	40.6	0.00

Table 3.3. Conversely, the size of the RMP is given by the number of columns generated. Therefore, as R increases, so does the size of the RMP and the associated computational cost. Consequently, solving the RMP is a computation bottleneck for CG, and currently constitutes the main limitation to the algorithm's scalability.

3.5.4 Evolution of the aggregated load

The evolution of the aggregated load is displayed in Figure 3.4, together with the market price of electricity over the considered time horizon. The normalized load takes value 1 (resp. 0) when the aggregated load reaches its upper bound $P_a^{\max}$ (resp. lower bound $P_a^{\min}$).

The initial load profile (Iter. 0) corresponds to each resource computing its minimum peak

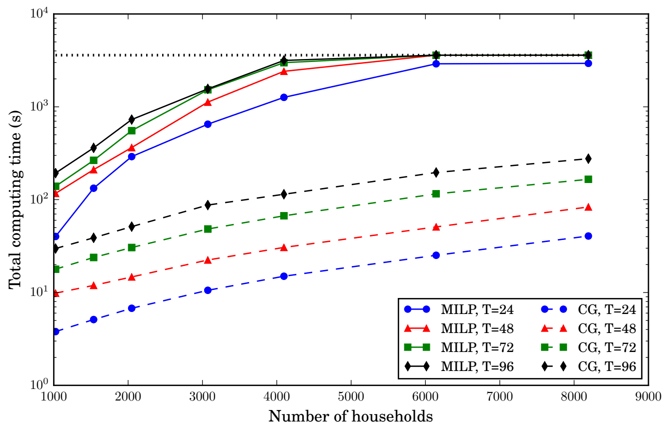


Figure 3.1 Average computing times for CG and MILP formulations, with $\xi^{\text{dep}} = 0.66$. The horizontal dotted line depicts the one-hour time limit for MILP.

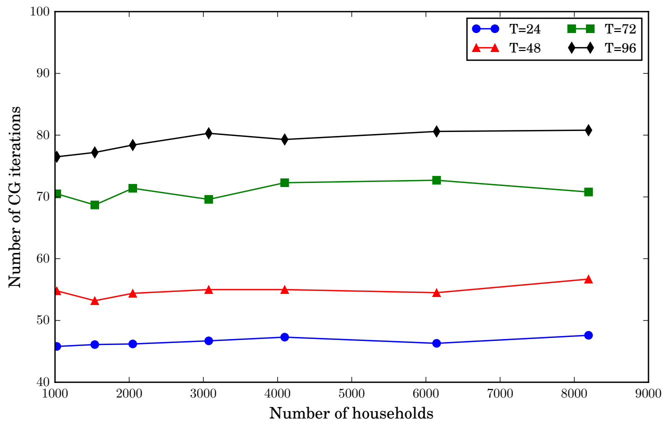


Figure 3.2 Average number of CG iterations, including the diving heuristic. $\xi^{\text{dep}} = 0.66$.

load schedule. Since the objective is to minimize the cost of purchasing electricity from the grid, we expect consumption to be shifted to periods when electricity is cheapest, while periods of high price should result in low demand. The evolution of the load depicted in Figure 3.4 shows that this is indeed the case. At the optimum (Iter. 46), consumption is highest at periods $t = 0, 5, 16, 17, 18$ and 22, which correspond to periods of lower prices.

Finally, the upper bound on the aggregated load is reached, e.g. for $t = 16, 17, 18$ at the optimum, but never violated. This demonstrates that the algorithm is effectively enforcing the linking constraints during the optimization process. This last feature is desirable, since a feasible solution will be available if the algorithm is stopped prematurely, e.g. due to time restrictions.

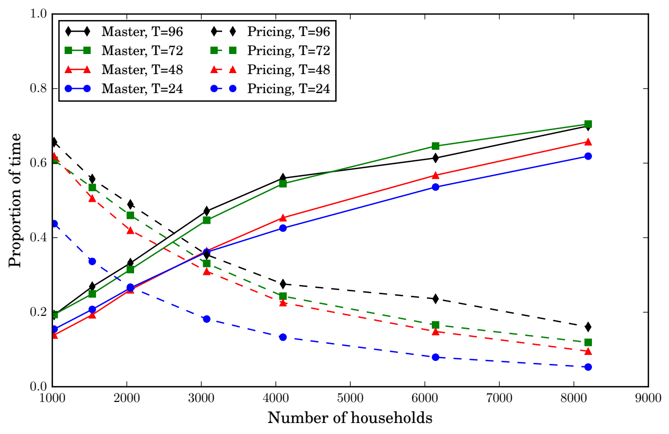


Figure 3.3 Time spent solving the master problem and pricing sub-problems, as a fraction of total computing time. $\xi^{\text{dep}} = 0.66$.

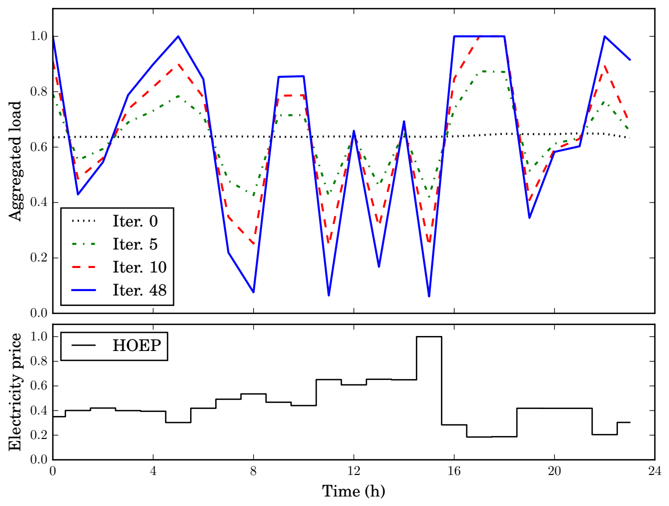


Figure 3.4 Normalized Hourly Ontario Electricity Price (HOEP, lower graph) and normalized aggregated load (upper graph) over the time horizon. The aggregated load is displayed at iterations 0, 5, 10 and 48 (last iteration) of the CG algorithm. $R = 1024, T = 24, \xi^{dep} = 0.66$.

3.5.5 Further discussion

As shown in Tables 3.2 and 3.3, root gaps for CG and MILP were often found to be lower than 10^{-4}, meaning both formulations have essentially zero integrality gap. This observation carries over to the rest of the dataset: for MILP, root gaps were always less than 0.02%, and smaller than 0.01% in 843 instances out of 1120. For CG, the largest recorded root gap was 0.008%. Moreover, root gaps tended to be smaller as the number of resources increased.

We now assess whether low integrality gaps reflect intrinsic properties of the problem at hand, or arise from our numerical data. To that end, we increased the batteries' minimum power ratios $P^{\text{ch, min}}/P^{\text{ch, max}}$ (resp. $P^{\text{dis, min}}/P^{\text{dis, max}}$) from 0, as in our initial tests, to 90%. This is done by setting the value of $P^{\text{ch, min}}$ (resp. $P^{\text{dis, min}}$) accordingly. Figure 3.5 displays the resulting evolution of computing time (left axis) and integrality gap (right axis). Results are reported for $R = 1024$, $T = 24$ and $\xi^{\text{dep}} = 0.66$, and similar behaviour were observed for other settings. Figure 3.5 shows that the MILP integrality gaps increases considerably, from 0.005% to over 2%. This resulted in an increase in the number of branching nodes, which we do not report for lack of space, and in computing time. On the other hand, although the CG gap increased, it remained below 0.01%. Moreover, the increase in computing time for CG is caused only by longer solving times for sub-problems. The number of CG iterations did not increase, nor did solving time for the RMP. Overall, CG appears to be more robust than MILP. This robustness is explained by the fact that changes in the resources' operation only affect sub-problems for CG while, for MILP, the entire problem structure may be affected.

3.6 Conclusion

In this paper, we have considered the problem of coordinating the operation of multiple DERs, and focused on the challenges raised by the presence of discrete decisions in the resources' operation, such as on-off constraints. We showed that this problem can be formulated as a centralized MILP, which is however intractable for large systems.

We have proposed an exact methodology for solving this MILP efficiently, based on Dantzig-Wolfe decomposition and Column Generation. In particular, we have shown that the proposed formulation is technology agnostic and can be implemented in a decentralized fashion, thus yielding high scalability while providing enhanced privacy to the resources. We have also reported on extensive computational results, which demonstrate the efficiency and robustness of our approach.

Future work will focus on integrating demand and price uncertainty in the formulation, as well as developing acceleration strategies to improve practical performance.

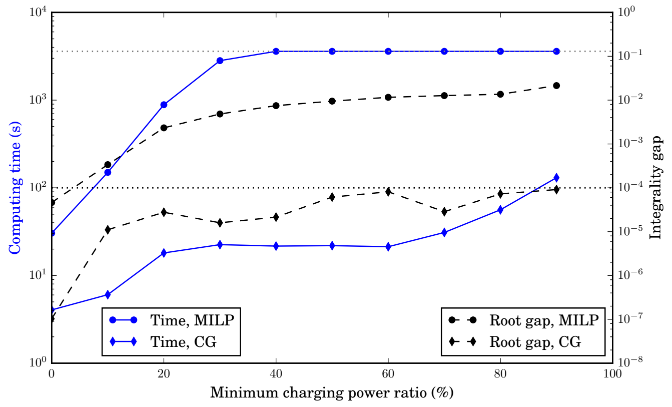


Figure 3.5 Average computing time (blue, left-hand axis scale) and root gap (black, right-hand axis scale) with respect to minimum charging power ratio. Dotted horizontal lines denote the one-hour time limit for MILP (top, in gray), and the 10^{-4} optionality threshold for integrality gaps (bottom, in black). Results obtained with $R = 1024$, $T = 24$ and $\xi^{\text{dep}} = 0.66$.

CHAPTER 4 ARTICLE 2 - DESIGN AND IMPLEMENTATION OF A MODULAR INTERIOR-POINT SOLVER FOR LINEAR OPTIMIZATION

Authors: Miguel Anjos, Andrea Lodi and Mathieu Tanneau
Submitted to *Mathematical Programming Computation.* Preprint: [67].

Abstract This paper introduces the algorithmic design and implementation of Tulip, an open-source interior-point solver for linear optimization. It implements a regularized homogeneous interior-point algorithm with multiple centrality corrections, and therefore handles unbounded and infeasible problems. The solver is written in Julia, thus allowing for a flexible and efficient implementation: Tulip's algorithmic framework is fully disentangled from linear algebra implementations and from a model's arithmetic. In particular, this allows to seamlessly integrate specialized routines for structured problems. Extensive computational results are reported. We find that Tulip is competitive with open-source interior-point solvers on the H. Mittelmann's benchmark of barrier linear programming solvers. Furthermore, we design specialized linear algebra routines for structured master problems in the context of Dantzig-Wolfe decomposition. These routines yield a tenfold speedup on large and dense instances that arise in power systems operation and two-stage stochastic programming, thereby outperforming state-of-the-art commercial interior point method solvers. Finally, we illustrate Tulip's ability to use different levels of arithmetic precision by solving problems in extended precision.

4.1 Introduction

Linear programming (LP) algorithms have been around for over 70 years, and LP remains a fundamental paradigm in optimization. Indeed, although nowadays most real-life applications involve discrete decisions or nonlinearities, the methods employed to solve them often rely on LP as their workhorse. Besides algorithms for mixed-integer linear programming (MILP), these include cutting-plane and outer-approximation algorithms that substitute a nonlinear problem with a sequence of iteratively refined LPs [68–70]. Furthermore, LP is at the heart of classical decomposition methods such as Dantzig-Wolfe and Benders decompositions [13, 14]. Therefore, efficient and robust LP technology is instrumental to our ability to solve more involved optimization problems.

Over the past few decades, interior-point methods (IPMs) have become a standard and efficient tool for solving LPs [71, 72]. While IPMs tend to outperform Dantzig's simplex algorithm on large-scale problems, the latter is well-suited to solving sequences of closely related LPs, by taking advantage of an advanced basis. Nevertheless, beyond sheer performance, it is now well recognized that a number of LP-based algorithms can further benefit from IPMs, despite their limited ability to warm start. In cutting plane algorithms, stronger cuts are often obtained by cutting off an interior point rather than an extreme vertex [70,73,74]. Similarly, IPMs have been successfully employed in the context of decomposition methods [75–79], wherein well-centered interior solutions typically provide a stabilization effect [80–82], thus reducing tailing-off and improving convergence.

4.1.1 Exploiting structure in IPMs

The remarkable performance of IPMs stems from both strong algorithmic foundations and efficient linear algebra. Indeed, the main computational effort of IPMs resides in the solution, at each iteration, of a system of linear equations. Therefore, the efficiency of the underlying linear algebra has a direct impact of the method's overall performance. Remarkably, while most IPM solvers employ general-purpose sparse linear algebra routines, substantial speedups can be obtained by exploiting a problem's specific structure. Nevertheless, successfully doing so requires (i) identifying a problem's structure and associated specialized linear algebra, (ii) integrating these custom routines within an IPM solver, and (iii) having a convenient and flexible way for the user to convey structural information to the solver. The main contribution of our work is to simplify the latter two points.

Numerous works have studied structure-exploiting IPMs, e.g., [83–91]. For instance, block-angular matrices typically arise in stochastic programming when using scenario decomposition. In [83] and later in [87], the authors thus design specialized factorization techniques that outperform generic implementations. Schultz et al. [85] design a specialized IPM for block-angular problems; therein, linking constraints are handled separately, thus allowing to decompose the rest of the problem. Gondzio [88] observed that the master problem in Dantzig-Wolfe decomposition possesses a block-angular structure. Similar approaches have been explored for network flow problems [86], multi-commodity flow problems [89], asset management problems [90], and for solving facility location problems [84, 91].

The aforementioned works focus on devising specialized linear algebra for a particular structure or application. On the other hand, a handful of IPM codes that accommodate various linear algebra implementations have been developed. The OOQP software, developed by Gertz and Wright [92], uses object-oriented design so that data structures and linear algebra routines can be tailored to specific applications. Motivated by large-scale stochastic programming, PIPS [93] incorporates a large share of OOQP's codebase, alongside specialized linear solvers for block-angular matrices. Nevertheless, to the best of the authors' knowledge, OOQP is no longer actively maintained, while current development on PIPS focuses on nonlinear programming.[1] In a similar fashion, OOPS [89, 90, 94] implements custom linear algebra that can exploit arbitrary block matrix structures. We also note that both PIPS and OOPS are primarily intended for massive parallelism on high-performance computing infrastructure. Furthermore, the BlockIP software [95] is designed for block-angular convex optimization problems, and solves linear systems with a combination of Cholesky factorization and preconditioned conjugate gradient. Both OOPS and BlockIP can be accessed through SML [25] –which requires AMPL, and are distributed under a closed-source proprietary license.

Finally, while nowadays most optimization solvers are written in C or C++, users are increasingly turning to higher-level programming languages such as Python, Matlab or Julia, alongside a variety of modeling tools, e.g, Pyomo [96], CVXPY [97], YALMIP [98], JuMP [12], to mention a few. Thus, users of high-level languages often have to switch to a low-level language in order to implement performance-critical tasks such as linear algebra. This situation, commonly referred to as the "two-language problem", hinders code development, maintenance, and usability.

4.1.2 Contributions and outline

In this paper, we describe the design and implementation of a modular interior-point solver, Tulip. The solver is written in Julia [99], which offers several advantages. First, Julia combines both high-level syntax and fast performance, thus addressing the two-language problem. In particular, it offers built-in support for linear algebra, with direct access to dense and sparse linear algebra libraries such as BLAS, LAPACK and SuiteSparse [100]. Second, the Julia ecosystem for optimization comprises a broad range of tools, from solvers' wrappers to modeling languages, alongside a growing and dynamic community of users. Finally, Julia's multiple dispatch feature renders Tulip's design fully flexible, thus allowing to disentangle the IPM algorithmic framework from linear algebra implementations, and to solve problems in arbitrary precision arithmetic.

The remainder of the paper is structured as follows. In Section 4.2, we introduce some notations and relevant definitions.

In Section 4.3, we describe the homogeneous self-dual embedding, and Tulip's regularized homogeneous interior-point algorithm. This feature contrasts with most IPM LP codes,

[1] Personal communication with PIPS developers.

namely, those that implement the almost-ubiquitous infeasible primal-dual interior-point algorithm [101]. The main advantage of the homogeneous algorithm is its ability to return certificates of primal or dual infeasibility. It is therefore better suited for use within cutting-plane algorithms or decomposition methods, wherein one may encounter infeasible or unbounded LPs.

In Section 4.4, we highlight the resolution of linear systems within Tulip, which builds on black-box linear solvers. This modular design leverages Julia's multiple dispatch, thereby facilitating the integration of custom linear algebra with no performance loss due to using external routines.

The presolve procedure is described in Section 4.5 and, in Section 4.6, we provide further implementation details of Tulip, such as the treatment of variable bounds, default values of parameters, and default linear solvers. Tulip is publicly available [102] under an open-source license. It can be used as a stand-alone package in Julia, and through the solver-independent interface `MathOptInterface` [103].

In Section 4.7, we report on three sets of computational experiments. First, we compare Tulip to several open-source and commercial IPM solvers on a benchmark set of unstructured LP instances. We observe that, using generic sparse linear algebra, Tulip is competitive with open-source IPM solvers. Second, we demonstrate Tulip's flexible design. We consider block-angular problems with dense linking constraints from two column-generation applications, for which we design specialized linear algebra routines. This implementation yields a tenfold speedup, thereby outperforming commercial solvers on large-scale instances. Third, we show how extended precision can alleviate numerical difficulties, thus illustrating Tulip's ability to work in arbitrary precision arithmetic.

Finally, Section 4.8 concludes the paper and highlights future research directions.

4.2 Notations

We consider LPs in primal-dual standard form

$$
\begin{array}{rlrl}
(P) & \displaystyle\min_{x} & c^T x & \\
 & s.t. & Ax & = b, \\
 & & x & \geq 0,
\end{array}
\qquad
\begin{array}{rlrl}
(D) & \displaystyle\max_{y,s} & b^T y & \\
 & s.t. & A^T y + s & = c, \\
 & & s & \geq 0,
\end{array}
\tag{4.1}
$$

where $c, x, s \in \mathbb{R}^n$, $b, y \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$ is assumed to have full row rank. We follow the usual notations from interior-point literature, and write X (resp. S) the diagonal matrix whose diagonal is given by x (resp. s), i.e., $X := Diag(x)$ and $S := Diag(s)$.

We denote I the identity matrix and e the vector with all coordinates equal to one; their respective dimensions are always obvious from context. The norm of a vector is written $\|\cdot\|$ and, unless specified otherwise, it denotes the ℓ_2 norm.

A primal solution x is feasible if $Ax = b$ and $x \geq 0$. A strictly feasible (or interior) solution is a primal feasible solution with $x > 0$. Similarly, a dual solution (y, s) is feasible if $A^T y + s = c$ and $s \geq 0$, and strictly feasible if, additionally, $s > 0$. Finally, a primal-dual solution (x, y, s) is optimal for (4.1) if x is primal-feasible, (y, s) is dual-feasible, and their objective values are equal, i.e., $c^T x = b^T y$.

A solution (x, y, s) with $x, s \geq 0$ is strictly complementary if

$$\forall i \in \{1, \dots, n\}, \Big(x_i s_i = 0 \text{ and } x_i + s_i > 0 \Big). \tag{4.2}$$

The complementarity gap is defined as $x^T s$. When (x, y, s) is primal-dual feasible, the complementarity gap equals the classical optimality gap, i.e., we have $x^T s = c^T x - b^T y$.

For ease of reading, we assume, without loss of generality, that all primal variables are required to be non-negative. The handling of free variables and of variables with finite upper bound will be detailed in Section 4.6.

4.3 Regularized homogeneous interior-point algorithm

In this section, we describe the homogeneous self-dual formulation and algorithm. Our implementation largely follows the algorithmic framework of [104] and [105], combined with the primal-dual regularization scheme of [106]. Consequently, we focus on the algorithm's main components, and refer to [104–106] for convergence proofs and theoretical results. Specific implementation details will be further discussed in Section 4.6.

4.3.1 Homogeneous self-dual embedding

The simplified homogeneous self-dual form was introduced in [104]. It consists in reformulating the primal-dual pair (4.1) as a single, self-dual linear program, which writes

$$
\begin{aligned}
(HSD) \quad \min_{x,y,\tau} \quad & 0 && (4.3)\\
s.t. \quad & -A^T y + c\tau \geq 0, && (4.4)\\
& Ax - b\tau = 0, && (4.5)\\
& -c^T x + b^T y \geq 0, && (4.6)\\
& x, \tau \geq 0, && (4.7)
\end{aligned}
$$

where τ is a scalar variable. Let s, κ be the non-negative slacks associated to (4.4) and (4.6), respectively. A solution (x, y, s, τ, κ) is strictly complementary if

$$x_i s_i = 0, x_i + s_i > 0, \text{ and } \tau\kappa = 0, \tau + \kappa > 0.$$

Problem (HSD) is always feasible, has empty interior and, under mild assumptions, possesses a strictly complementary feasible solution [104].

Let $(x^*, y^*, s^*, \tau^*, \kappa^*)$ be a strictly complementary feasible solution for (HSD). If $\tau^* > 0$, then $(\frac{x^*}{\tau^*}, \frac{y^*}{\tau^*}, \frac{s^*}{\tau^*})$ is an optimal solution for the original problem (4.1). Otherwise, we have $\kappa^* > 0$ and thus $c^T x^* - b^T y^* < 0$. In that case, the original problem (P) is infeasible or unbounded. If $c^T x^* < 0$, then (P) is unbounded and x^* is an unbounded ray. If $-b^T y^* < 0$, then (P) is infeasible and y^* is an unbounded dual ray. The latter is also referred to as a Farkas proof of infeasibility. Finally, if both $c^T x^* < 0$ and $-b^T y^* < 0$, then both (P) and (D) are infeasible.

4.3.2 Regularized formulation

Friedlander and Orban [106] introduce an exact primal-dual regularization scheme for convex quadratic programs, which we extend to the HSD form. The benefits of regularizations will be further detailed in Section 4.4. Importantly, rather than viewing (HSD) as a generic LP to which the regularization procedure of [106] is applied, we exploit the fact that (HSD) is a self-dual embedding of (P)–(D), and formulate the regularization in the original primal-dual space.

Thus, we consider a *single*, regularized, self-dual problem

$$
\begin{aligned}
(rHSD) \quad \min_{x,y,\tau} \quad & \rho_p(x-\bar{x})^T x + \rho_d(y-\bar{y})^T y + \rho_g(\tau-\bar{\tau})\tau && (4.8)\\
s.t. \quad & -A^T y + c\tau + \rho_p(x-\bar{x}) \geq 0, && (4.9)\\
& Ax - b\tau + \rho_d(y-\bar{y}) = 0, && (4.10)\\
& -c^T x + b^T y + \rho_g(\tau-\bar{\tau}) \geq 0, && (4.11)\\
& x, \tau, \geq 0, && (4.12)
\end{aligned}
$$

where ρ_p, ρ_d, ρ_g are positive scalars, and $\bar{x} \in \mathbb{R}^n, \bar{y} \in \mathbb{R}^m, \bar{\tau} \in \mathbb{R}$ are given estimates of an optimal solution of (HSD). We denote by s, κ the non-negative slack variables of constraints (4.9) and (4.11), respectively. The first-order Karush-Kuhn-Tucker (KKT) conditions for $(rHSD)$ can then be expressed in the following form:

$$
\begin{aligned}
\rho_p x - A^T y - s + c\tau &= \rho_p \bar{x}, && (4.13)\\
Ax + \rho_d y - b\tau &= \rho_d \bar{y}, && (4.14)\\
-c^T x + b^T y + \rho_g \tau - \kappa &= \rho_g \bar{\tau}, && (4.15)\\
x_j s_j &= 0, \qquad j = 1, ..., n && (4.16)\\
\tau\kappa &= 0, && (4.17)\\
x, s, \tau, \kappa &\geq 0. && (4.18)
\end{aligned}
$$

The correspondence between $(rHSD)$ and [106] follows from the fact that, up to a constant term, the objective function (4.8) equals

$$
\frac{1}{2}\left(\rho_p \|x-\bar{x}\|^2 + \rho_d \|y-\bar{y}\|^2 + \rho_g \|\tau-\bar{\tau}\|^2 + \rho_p \|x\|^2 + \rho_d \|y\|^2 + \rho_g \|\tau\|^2\right).
$$

Note that, for $\rho_p = \rho_d = \rho_g = 0$, the regularized problem $(rHSD)$ reduces to (HSD). Furthermore, Theorem 1 shows that, for positive ρ_p, ρ_d, ρ_g, the regularization is exact.

Theorem 1. *Assume $\rho_p, \rho_d, \rho_g > 0$. Let (x^*, y^*, τ^*) be a complementary optimal solution of (HSD), and let $(\bar{x}, \bar{y}, \bar{\tau}) = (x^*, y^*, \tau^*)$ in the definition of $(rHSD)$. Then, (x^*, y^*, τ^*) is the unique optimal solution of $(rHSD)$.*

Proof. The uniqueness of the optimum is a direct consequence of $(rHSD)$ being a convex problem with strictly convex objective.

Next, we show that any feasible solution of $(rHSD)$ has non-negative objective. Let (x, y, s, τ, κ) be a feasible solution of $(rHSD)$. Substituting Eq. (4.9)-(4.11) into the objective (4.8), one obtains

$$\begin{aligned}Z &= \rho_p(x-\bar{x})^T x + \rho_d(y-\bar{y})^T y + \rho_g(\tau-\bar{\tau})\tau \\ &= (A^T y + s - c\tau)^T x + (b\tau - Ax)^T y + (c^T x - b^T y + \kappa)\tau \\ &= x^T s + \tau\kappa \geq 0.\end{aligned}$$

Then, (x^*, y^*, τ^*) is trivially feasible for $(rHSD)$, and its objective value is $(x^*)^T s^* + \tau^*\kappa^* = 0$. Thus, it is optimal for $(rHSD)$, which concludes the proof. $\square$

4.3.3 Regularized homogeneous algorithm

We now describe the regularized homogeneous interior-point algorithm. Similar to [106], we apply a single Newton iteration to a sequence of problems of the form $(rHSD)$ where, at each iteration, $\bar{x}, \bar{y}, \bar{\tau}$ are chosen to be the current primal-dual iterate.

Let (x, y, s, τ, κ) denote the current primal-dual iterate, with $(x, s, \tau, \kappa) > 0$, and define the residuals

$$r_p = b\tau - Ax, \tag{4.19}$$

$$r_d = c\tau - A^T y - s, \tag{4.20}$$

$$r_g = c^T x - b^T y + \kappa, \tag{4.21}$$

and the barrier parameter

$$\mu = \frac{x^T s + \tau\kappa}{n+1}.$$

For given $\bar{x}, \bar{y}, \bar{\tau}$, a search direction $(\delta_x, \delta_y, \delta_s, \delta_\tau, \delta_\kappa)$ is computed by solving a Newton system of the form

$$-\rho_p\delta_x + A^T\delta_y + \delta_s - c\delta_\tau = \eta\left(c\tau - A^T y - s + \rho_p(\bar{x} - x)\right), \tag{4.22}$$

$$A\delta_x + \rho_d\delta_y - b\delta_\tau = \eta\left(b\tau - Ax - \rho_d(y - \bar{y})\right), \tag{4.23}$$

$$-c^T\delta_x + b^T\delta_y + \rho_g\delta_\tau - \delta_\kappa = \eta\left(c^T x - b^T y + \kappa - \rho_g(\tau - \bar{\tau})\right), \tag{4.24}$$

$$S\delta_x + X\delta_s = -XSe + \gamma\mu e, \tag{4.25}$$

$$\kappa\delta_\tau + \tau\delta_\kappa = -\tau\kappa + \gamma\mu, \tag{4.26}$$

where γ and η are non-negative scalars whose values will be specified in Section 4.3.3. We evaluate the Newton system at $(\bar{x}, \bar{y}, \bar{\tau}) = (x, y, \tau)$, which yields

$$\begin{bmatrix} -\rho_p I & A^T & I & -c & 0 \\ A & \rho_d I & 0 & -b & 0 \\ -c^T & b^T & 0 & \rho_g & -1 \\ S & 0 & X & 0 & 0 \\ 0 & 0 & 0 & \kappa & \tau \end{bmatrix} \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_s \\ \delta_\tau \\ \delta_\kappa \end{bmatrix} = \begin{bmatrix} \eta r_d \\ \eta r_p \\ \eta r_g \\ -XSe + \gamma \mu e \\ -\tau\kappa + \gamma\mu \end{bmatrix}. \tag{4.27}$$

System (4.27) is identical to the Newton system obtained when solving (HSD) (see, e.g., [105]), except for the regularization terms that appear in the left-hand side. In particular, the right-hand side remains unchanged.

Starting point

We choose the following default starting point

$$(x^0, y^0, s^0, \tau^0, \kappa^0) = (e, 0, e, 1, 1).$$

This initial point was proposed in [104]. Besides its simplicity, it has well-balanced complementarity products, which are all equal to one.

Search direction

At each iteration, a search direction is computed using Mehrotra's predictor-corrector technique [101], combined with Gondzio's multiple centrality corrections [107]. Following [105], we adapt the original formulas of [101, 107] to account for the homogeneous embedding.

First, the affine-scaling direction $(\delta_x^{\text{aff}}, \delta_y^{\text{aff}}, \delta_s^{\text{aff}}, \delta_\tau^{\text{aff}}, \delta_\kappa^{\text{aff}})$ is obtained by solving the Newton system

$$-\rho_p \delta_x^{\text{aff}} + A^T \delta_y^{\text{aff}} + \delta_s^{\text{aff}} - c\delta_\tau^{\text{aff}} = r_d, \tag{4.28}$$

$$A\delta_x^{\text{aff}} + \rho_d \delta_y^{\text{aff}} - b\delta_\tau^{\text{aff}} = r_p, \tag{4.29}$$

$$-c^T \delta_x^{\text{aff}} + b^T \delta_y^{\text{aff}} + \rho_g \delta_\tau^{\text{aff}} - \delta_\kappa^{\text{aff}} = r_g, \tag{4.30}$$

$$S\delta_x^{\text{aff}} + X\delta_s^{\text{aff}} = -XSe, \tag{4.31}$$

$$\kappa\delta_\tau^{\text{aff}} + \tau\delta_\kappa^{\text{aff}} = -\tau\kappa, \tag{4.32}$$

which corresponds to (4.27) with $\eta = 1$ and $\gamma = 0$. Taking a full step ($\alpha = 1$) would thus reduce both infeasibility and complementarity gap to zero. However, doing so is generally not possible, due to the non-negativity requirement on (x, s, τ, κ).

Consequently, a corrected search direction is computed, as proposed in [101]. The corrected direction hopefully enables one to make longer steps, thus reducing the total number of IPM iterations. Let $\eta = 1 - \gamma$, where

$$\gamma = (1 - \alpha^{\text{aff}})^2 \min\Big(\gamma_{min}, (1 - \alpha^{\text{aff}})\Big) \tag{4.33}$$

for some $\gamma_{min} > 0$, and

$$\alpha^{\text{aff}} = \max\Big\{0 \leq \alpha \leq 1 \mid (x, s, \tau, \kappa) + \alpha(\delta_x^{\text{aff}}, \delta_s^{\text{aff}}, \delta_\tau^{\text{aff}}, \delta_\kappa^{\text{aff}}) \geq 0\Big\}. \tag{4.34}$$

The corrected search direction is then given by

$$\begin{aligned}
-\rho_p \delta_x + A^T \delta_y + \delta_s - c\delta_\tau &= \eta r_d, && (4.35)\\
A\delta_x + \rho_d \delta_y - b\delta_\tau &= \eta r_p, && (4.36)\\
-c^T \delta_x + b^T \delta_y + \rho_g \delta_\tau - \delta_\kappa &= \eta r_g, && (4.37)\\
S\delta_x + X\delta_s &= -XSe + \gamma\mu e - \Delta_x^{\text{aff}} \Delta_s^{\text{aff}} e, && (4.38)\\
\kappa\delta_\tau + \tau\delta_\kappa &= -\tau\kappa + \gamma\mu - \delta_\tau^{\text{aff}} \delta_\kappa^{\text{aff}}, && (4.39)
\end{aligned}$$

where $\Delta_x^{\text{aff}} = Diag(\delta_x^{\text{aff}})$ and $\Delta_s^{\text{aff}} = Diag(\delta_s^{\text{aff}})$.

Additional centrality corrections

Additional centrality corrections aim at improving the centrality of the new iterate, i.e., to keep the complementary products well balanced. Doing so generally allows to make longer steps, thus reducing the total number of IPM iterations. We implement Gondzio's original technique [107], with some modifications introduced in [105].

Let $\delta = (\delta_x, \delta_y, \delta_s, \delta_\tau, \delta_\kappa)$ be the current search direction, α^{max} the corresponding maximum step size, and define

$$(\bar{x}, \bar{y}, \bar{s}, \bar{\tau}, \bar{\kappa}) := (x, y, s, \tau, \kappa) + \bar{\alpha}(\delta_x, \delta_y, \delta_s, \delta_\tau, \delta_\kappa), \tag{4.40}$$

where $\bar{\alpha} := \min(1, 2\alpha^{max})$ is a tentative step size.

First, a soft target in the space of complementarity products is computed as

$$t_j = \begin{cases} \mu_l - \bar{x}_j \bar{s}_j & \text{if } \bar{x}_j \bar{s}_j < \mu_l \\ 0 & \text{if } \bar{x}_j \bar{s}_j \in [\mu_l, \mu_u] \\ \mu_u - \bar{x}_j \bar{s}_j & \text{if } \bar{x}_j \bar{s}_j > \mu_u \end{cases}, \quad j = 1, \dots, n, \tag{4.41}$$

$$t_0 = \begin{cases} \mu_l - \bar{\tau} \bar{\kappa} & \text{if } \bar{\tau} \bar{\kappa} < \mu_l \\ 0 & \text{if } \bar{\tau} \bar{\kappa} \in [\mu_l, \mu_u] \\ \mu_u - \bar{\tau} \bar{\kappa} & \text{if } \bar{\tau} \bar{\kappa} > \mu_u \end{cases}, \tag{4.42}$$

where $\mu_l = \gamma \mu \beta$ and $\mu_u = \gamma \mu \beta^{-1}$, for a fixed $0 < \beta \leq 1$. Then, define

$$v = t - \frac{e^T t + t_0}{n+1} e, \tag{4.43}$$

$$v_0 = t_0 - \frac{e^T t + t_0}{n+1}. \tag{4.44}$$

A correction is obtained by solving the linear system

$$-\rho_p \delta_x^c + A^T \delta_y^c + \delta_s^c - c \delta_\tau^c = 0, \tag{4.45}$$

$$A \delta_x^c + \rho_d \delta_y^c - b \delta_\tau^c = 0, \tag{4.46}$$

$$-c^T \delta_x^c + b^T \delta_y^c + \rho_g \delta_\tau^c - \delta_\kappa^c = 0, \tag{4.47}$$

$$S \delta_x^c + X \delta_s^c = v, \tag{4.48}$$

$$\kappa \delta_\tau^c + \tau \delta_\kappa^c = v_0, \tag{4.49}$$

which yields a corrected search direction

$$(\delta_x, \delta_y, \delta_s, \delta_\tau, \delta_\kappa) + (\delta_x^c, \delta_y^c, \delta_s^c, \delta_\tau^c, \delta_\kappa^c).$$

The corrected direction is accepted if it results in an increased step size.

Finally, additional centrality corrections are computed only if a sufficient increase in the step size is observed. Specifically, as suggested in [105], an additional correction is computed only if the new step size α satisfies

$$\alpha \geq 1.10 \times \alpha^{\max}. \tag{4.50}$$

Regularizations

Following [106], the regularizations are updated as follows. Let $\rho_p^k, \rho_d^k, \rho_g^k$ denote the regularization terms at iteration k. We set $\rho_p^0 = \rho_d^0 = \rho_g^0 = 1$, and use the update rule

$$\rho_p^{k+1} = \max\left(\sqrt{\epsilon}, \frac{\rho_p^k}{10}\right), \tag{4.51}$$

$$\rho_d^{k+1} = \max\left(\sqrt{\epsilon}, \frac{\rho_d^k}{10}\right), \tag{4.52}$$

$$\rho_g^{k+1} = \max\left(\sqrt{\epsilon}, \frac{\rho_g^k}{10}\right), \tag{4.53}$$

where ϵ denotes the machine precision, e.g., $\epsilon \simeq 10^{-16}$ for double-precision floating point arithmetic.

Further details on the role of regularizations in the resolution of the Newton system are given in Section 4.4. Let us only mention here that ρ_p, ρ_d, ρ_g may become too small to ensure that the Newton system is properly regularized, e.g., for badly scaled problems. When this is the case, we increase the regularizations by a factor of 100, and terminate the algorithm if three consecutive increases fail to resolve the numerical issues.

Step size

Once the final search direction has been computed, the step size α is given by

$$\alpha = 0.9995 \times \alpha^{max}, \tag{4.54}$$

where

$$\alpha^{max} = \max\left\{0 \leq \alpha \leq 1 \mid (x, s, \tau, \kappa) + \alpha(\delta_x, \delta_s, \delta_\tau, \delta_\kappa) \geq 0\right\}.$$

Stopping criteria

The algorithm stops when, up to numerical tolerances, one of the following three cases holds: the current iterate is optimal, the primal problem is proven infeasible, the dual problem is proven infeasible (unbounded primal).

The problem is declared solved to optimality if

$$\frac{\|r_p\|_\infty}{\tau(1+\|b\|_\infty)} < \varepsilon_p, \tag{4.55}$$

$$\frac{\|r_d\|_\infty}{\tau(1+\|c\|_\infty)} < \varepsilon_d, \tag{4.56}$$

$$\frac{|c^T x - b^T y|}{\tau + |b^T y|} < \varepsilon_g, \tag{4.57}$$

where $\varepsilon_p, \varepsilon_d, \varepsilon_g$ are positive parameters. The above criteria are independent of the magnitude of τ, and correspond to primal feasibility, dual feasibility and optimality, respectively.

Primal or dual infeasibility is detected if

$$\mu < \varepsilon_i, \tag{4.58}$$

$$\frac{\tau}{\kappa} < \varepsilon_i, \tag{4.59}$$

where ε_i is a positive parameter. When this is the case, a complementary solution with small τ has been found. If $c^T x < -\varepsilon_i$, the problem is declared dual infeasible (primal unbounded), and x is an unbounded ray. If $-b^T y < -\varepsilon_i$, the problem is declared primal infeasible (dual unbounded), and y is a Farkas dual ray.

Finally, premature termination criteria such as numerical instability, time limit or iteration limit are discussed in Section 4.6.

4.4 Solving linear systems

Search directions and centrality corrections are obtained by solving several Newton systems such as (4.28)-(4.32), all with identical left-hand side matrix but different right-hand side. Specifically, each Newton system has the form

$$\begin{bmatrix} -\rho_p I & A^T & I & -c & \\ A & \rho_d I & & -b & \\ -c^T & b^T & & \rho_g & -1 \\ S & & X & & \\ & & & \kappa & \tau \end{bmatrix} \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_s \\ \delta_\tau \\ \delta_\kappa \end{bmatrix} = \begin{bmatrix} \xi_d \\ \xi_p \\ \xi_g \\ \xi_{xs} \\ \xi_{\tau\kappa} \end{bmatrix}, \tag{4.60}$$

where $\xi_p, \xi_d, \xi_g, \xi_{xs}, \xi_{\tau\kappa}$ are appropriate right-hand side vectors. The purpose of this section is to provide further details on the techniques used for the resolution of (4.60), and their implementation in Tulip.

4.4.1 Augmented system

First, we eliminate δ_s and δ_κ as follows:

$$\delta_s = X^{-1}(\xi_{xs} - S\delta_x), \tag{4.61}$$

$$\delta_\kappa = \tau^{-1}(\xi_{\tau\kappa} - \kappa\delta_\tau), \tag{4.62}$$

which yields

$$\begin{bmatrix} -(\Theta^{-1} + \rho_p I) & A^T & -c \\ A & \rho_d I & -b \\ -c^T & b^T & \tau^{-1}\kappa + \rho_g \end{bmatrix} \begin{bmatrix} \delta_x \\ \delta_y \\ \delta_\tau \end{bmatrix} = \begin{bmatrix} \xi_d - X^{-1}\xi_{xs} \\ \xi_p \\ \xi_g + \tau^{-1}\xi_{\tau\kappa} \end{bmatrix}, \tag{4.63}$$

where $\Theta = XS^{-1}$.

As outlined in [71, 105], a solution to (4.63) is obtained by first solving

$$\begin{bmatrix} -(\Theta^{-1} + \rho_p I) & A^T \\ A & \rho_d I \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} c \\ b \end{bmatrix}, \tag{4.64}$$

and

$$\begin{bmatrix} -(\Theta^{-1} + \rho_p I) & A^T \\ A & \rho_d I \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \xi_d - X^{-1}\xi_{xs} \\ \xi_p \end{bmatrix}. \tag{4.65}$$

Linear systems of the form (4.64) and (4.65) are referred to as *augmented systems*. Then, $\delta_x, \delta_y, \delta_\tau$ are computed as follows:

$$\delta_\tau = \frac{\xi_g + \tau^{-1}\xi_{\tau\kappa} + c^T u + b^T v}{\tau^{-1}\kappa + \rho_g - c^T p + b^T q}, \tag{4.66}$$

$$\delta_x = u + \delta_\tau p, \tag{4.67}$$

$$\delta_y = v + \delta_\tau q. \tag{4.68}$$

Note that (4.64) does not depend on the right-hand side ξ. Thus, it is only solved once per IPM iteration, and its solution is reused when solving subsequent Newton systems.

Finally, as pointed in [106], the augmented system's structure motivates the following observations. First, the use of primal-dual regularizations controls the effective condition number of the augmented system, which, in turn, improves the algorithm's numerical behavior. Second, the augmented system's matrix is symmetric quasi-definite. This allows the use of efficient symmetric indefinite factorization techniques, which only require one symbolic analysis at the beginning of the optimization. In particular, dual regularizations ensure that this quasi-definite property is retained even when A does not have full rank. Third, directly solving the augmented system implicitly handles dense columns in A, which make the system of normal equations dense [71]. We have also found this approach to be more numerically stable than a normal equations system-based approach.

4.4.2 Black-box linear solvers

The augmented system may be solved using a number of techniques, with direct methods –namely, symmetric factorization techniques– being the most popular choice. Importantly, the algorithm itself is unaffected by *how* the augmented system is solved, provided that it is solved accurately. Our implementation leverages Julia's multiple dispatch feature and built-in support for linear algebra, thus allowing to disentangle the algorithmic framework from the linear algebra implementation.

First, the interior-point algorithm is defined over abstract linear algebra structures. Namely, the constraint matrix A is treated as an `AbstractMatrix`, whose concrete type is only known once the model is instantiated. Julia's standard library includes extensive support for linear algebra, thus removing the need for a custom abstract linear algebra layer.

Second, while the reduction from the Newton system to the augmented system is performed explicitly, the latter is solved by a black-box linear solver. Specifically, we design an `AbstractKKTSolver` type, from which concrete linear solver implementations inherit. The `AbstractKKTSolver` interface is deliberately minimal, and consists of three functions:[2] `setup`, `update!`, and `solve!`.

A linear solver is instantiated at the beginning of the optimization using the `setup` function. Custom options can be passed to `setup` so that the user can select a linear solver of their choice. At the beginning of each IPM iteration, the linear solver's state is updated by calling the `update!` function. For instance, if a direct method is used, this step corresponds to updating the factorization. Following the call to `update!`, augmented systems can be solved through the `solve!` function. Default, generic, linear solvers are described in Section 4.6.3, and an example of specialized linear solver is given in Section 4.7.2. Specific details are provided in Tulip's online documentation.[3]

Finally, specialized methods are automatically dispatched based on the (dynamic) type of A. These include matrix-vector and matrix-matrix product, as well as matrix factorization

[2]In Julia, a `!` is appended to functions that mutate their arguments.

[3]`https://ds4dm.github.io/Tulip.jl/dev/`

routines. We emphasize that the dispatch feature is a core component of the Julia programming language, and is therefore entirely transparent to the user. Consequently, one can easily define custom routines that exploit certain properties of A, so as to speed-up computation or reduce memory overheads. Furthermore, this customization is entirely independent of the interior-point algorithm, thus allowing to properly assess the impact of different linear algebra implementations.

4.5 Presolve

Tulip's presolve module performs elementary reductions, all of which are described in [108] and [109]. Therefore, in this section, we only outline the presolve procedure; further implementation details are given in Section 4.6.

4.5.1 Presolve

We only perform reductions that do not introduce any additional non-zero coefficients, i.e., fill-in, to the problem. The presolve procedure is outlined in Algorithm 3, and proceeds as follows.

First, we ensure all bounds are consistent, remove all empty rows and columns, and identify all row singletons, i.e., rows that contain with a single non-zero coefficient. Then, a series of passes is performed until no further reduction is possible. At each pass, the following reductions are applied: empty rows and columns, fixed variables, row singletons, free and implied free column singletons, forcing and dominated rows, and dominated columns. The presolve terminates if infeasibility or unboundedness is detected, in which case an appropriate primal or dual ray is constructed. If all rows and columns are eliminated, the problem is declared solved, and a primal-dual optimal solution is constructed.

Finally, to improve the numerical properties of the problem, rows and columns are re-scaled as follows:

$$\tilde{A} = D^{(r)} \times A \times D^{(c)}, \tag{4.69}$$

where $\tilde{A}$ is the scaled matrix, A is the constraint matrix of the reduced problem, and $D^{(r)}$,

$D^{(c)}$ are diagonal matrices with coefficients

$$D_i^{(r)} = \frac{1}{\sqrt{\|A_{i,\cdot}\|}}, \quad \forall i, \tag{4.70}$$

$$D_j^{(c)} = \frac{1}{\sqrt{\|A_{\cdot,j}\|}}, \quad \forall j. \tag{4.71}$$

Column and row bounds, as well as the objective, are scaled appropriately.

```
Input: Initial LP
  Remove empty rows
  Remove empty columns
  repeat
    Check for bounds inconsistencies
    Remove empty columns

    Remove row singletons
    Remove fixed variables

    Remove row singletons
    Remove forcing/dominated rows

    Remove row singletons
    Remove free columns singletons

    Remove row singletons
    Remove dominated columns
  until No reduction is found
  Scale rows and columns
```

Algorithm 3 Presolve procedure

4.5.2 Postsolve

A primal-dual solution to the presolved problem is computed using the interior-point algorithm described in Section 4.3. A solution to the original problem is then constructed in a postsolve phase, whose algorithmic details are detailed in [108, 109]. Note that, in general, the postsolve solution is *not* an interior point with respect to the original problem, e.g., some variables may be at their upper or lower bound.

4.6 Implementation details

Tulip is an officially registered Julia package, and is publicly available[4] under an open-source license. The entire source code comprises just over $4,000$ lines of Julia code, which makes it easy to read and to modify. The code is single-threaded, however external linear algebra libraries may exploit multiple threads.

We provide an interface to `MathOptInterface` [103], a solver-agnostic abstraction layer for optimization. Thus, Tulip is readily available through both `JuMP` [12], an open-source algebraic modeling language embedded in Julia, and the convex optimization modeling framework `Convex` [110].

Finally, Tulip supports arbitrary precision arithmetic, thus allowing, for instance, to solve problems in quadruple (128 bits) precision. This functionality is available from Tulip's direct API and through the `MathOptInterface` API; it is illustrated in Section 4.7.3.

4.6.1 Bounds on variables

Tulip stores LP problems in the form

$$\begin{array}{llcccl} (LP) & \displaystyle\min_{x} & & c^T x & + c_0 & \\ & s.t. & l_i^b \leq & \sum_j a_{i,j} x_j & \leq u_i^b, & \forall i = 1, ..., m, \\ & & l_j^x \leq & x_j & \leq u_j^x, & \forall j = 1, ..., n, \end{array} \tag{4.72}$$

where $l_{i,j}^{b,x}, u_{i,j}^{b,x} \in \mathbb{R} \cup \{-\infty, +\infty\}$, i.e., some bounds may be infinite. Before being passed to the interior-point optimizer, the problem is transformed into standard form. This transformation occurs after the presolve phase, and is transparent to the user. In particular, primal-dual solutions are returned with respect to formulation (4.72).

Free variables are an outstanding issue for interior-point methods, see, e.g. [71, 111], and are not supported explicitly in Tulip. Instead, free variables are automatically split into the difference of two non-negative variables, with the knowledge that this reformulation may introduce some numerical instability.

Although finite upper bounds may be treated as arbitrary constraints, it is more efficient to handle them separately. Let $\mathcal{I}$ denote the set of indices of upper-bounded variables.

[4]Source code is available at `https://github.com/ds4dm/Tulip.jl`, and online documentation at `https://ds4dm.github.io/Tulip.jl/dev/`

Upper-bound constraints then write

$$x_i \leq u_i, \quad \forall i \in \mathcal{I}, \tag{4.73}$$

which we write in compact form $Ux \leq u$, where $U \in \mathbb{R}^{|\mathcal{I}| \times n}$ and

$$U_{i,j} = \begin{cases} 1 & \text{if } i = j \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases}.$$

Therefore, internally, Tulip solves linear programs of the form

$$\begin{array}{llllll} (P) & \displaystyle\min_{x,w} & c^T x & (D) & \displaystyle\max_{y,s,z} & b^T y - u^T z \\ & s.t. & Ax = b, & & s.t. & A^T y + s - U^T z = c, \\ & & Ux + w = u & & & s, z \geq 0. \\ & & x, w \geq 0, & & & \end{array} \tag{4.74}$$

Let us emphasize that handling upper bounds separately only affects the underlying linear algebra operations, not the interior-point algorithm.

The Newton system (4.60) then writes

$$\begin{bmatrix} -\rho_p I & & A^T & I & -U^T & -c & \\ A & & \rho_d I & & & -b & \\ U & I & & & & -u & \\ -c^T & & b^T & & -u^T & \rho_g & -1 \\ S & & & X & & & \\ & Z & & & W & & \\ & & & & & \kappa & \tau \end{bmatrix} \begin{bmatrix} \delta_x \\ \delta_w \\ \delta_y \\ \delta_s \\ \delta_z \\ \delta_\tau \\ \delta_\kappa \end{bmatrix} = \begin{bmatrix} \xi_d \\ \xi_p \\ \xi_u \\ \xi_g \\ \xi_{xs} \\ \xi_{wz} \\ \xi_{\tau\kappa} \end{bmatrix}, \tag{4.75}$$

and it reduces, after performing diagonal substitutions, to solving two augmented systems of the form

$$\begin{bmatrix} -(\tilde{\Theta}^{-1} + \rho_p I) & A^T \\ A & \rho_d I \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} \tilde{\xi}_d \\ \tilde{\xi}_p \end{bmatrix}, \tag{4.76}$$

where $\tilde{\Theta} = \left(X^{-1}S + U^T(W^{-1}Z)U \right)^{-1}$. Note that $\tilde{\Theta}$ is a diagonal matrix with positive diagonal. Therefore, system (4.76) has the same size and structure as (4.64). Furthermore, $\tilde{\Theta}$ can be computed efficiently using only vector operations, i.e., without any matrix-matrix nor matrix-vector product.

4.6.2 Solver parameters

The default values for numerical tolerances of Section 4.3.3 are

$$\begin{aligned}\varepsilon_p &= \sqrt{\epsilon},\\ \varepsilon_d &= \sqrt{\epsilon},\\ \varepsilon_g &= \sqrt{\epsilon},\\ \varepsilon_i &= \sqrt{\epsilon},\end{aligned}$$

where ϵ is the machine precision, which depends on the arithmetic. For instance, double precision (64 bits) floating point arithmetic corresponds to $\epsilon_{64} \simeq 10^{-16}$, while quadruple precision (128 bits) corresponds to $\epsilon_{128} \simeq 10^{-34}$.

When computing additional centrality corrections, we use the following default values:

$$\begin{aligned}\gamma_{min} &= 10^{-1},\\ \beta &= 10^{-1}.\end{aligned}$$

The default maximum number of centrality corrections is set to 5.

Finally, the maximum number of IPM iterations is set to a default of 100. A time limit may be imposed by the user, in which case it is checked at the beginning of each IPM iteration.

4.6.3 Default linear solvers

Several generic linear algebra implementations are readily available in Tulip, and can be selected without requiring any additional implementation.

The default settings are as follows. First, A is stored in a `SparseMatrixCSC` struct, i.e., in compressed sparse column format. Elementary linear algebra operations, e.g., matrix-vector products, employ Julia's standard library `SparseArrays`. Augmented systems are then solved by a direct method, namely, an LDL^T factorization of the quasi-definite augmented system. Sparse factorizations use either the CHOLMOD module of SuiteSparse [100], or the `LDLFactorizations` package [112], a Julia translation of SuiteSparse's LDL^T factorization code that supports arbitrary arithmetic. Tulip uses the former for double precision floating point arithmetic, and the latter otherwise. Finally, the solver's log indicates: the model's arithmetic, the linear solver's backend, e.g., CHOLMOD, and the linear system being solved, i.e., either the augmented system of the normal equations system.

As mentioned in Section 4.4, custom options for linear algebra can be passed to the solver.

Specifically, the `MatrixOptions` parameter lets the user select a matrix implementation of their choice, and the `KKTOptions` parameter is used to specify a choice of linear solver. Their usage is depicted in Figure 4.1.

In Figure 4.1a, the default settings are used. The model is instantiated at line 1; the `Model{Float64}` syntax indicates that `Float64` arithmetic is used. Then, the problem is read from the `problem.mps` file at line 2, and the model is solved at line 4. Figures 4.1b, 4.1c, 4.1d are identical, but select different linear algebra implementations by setting the appropriate `MatrixOptions` and `KKTOptions` parameters.

Figure 4.1b illustrates the use of dense linear algebra. Line 5 indicates that A should be stored as a dense matrix. Then, at line 6, a dense linear solver is selected through the `SolverOptions(Dense_SymPosDef)` setting. In this case, the augmented system is reduced to the (dense) normal equations systems, and a dense Cholesky factorization is applied; BLAS/LAPACK routines are automatically called when using single and double precision floating point arithmetic, otherwise Julia's generic routines are called.

In the example of Figure 4.1c, linear systems are reduced to the normal equations system, and CHOLMOD's sparse Cholesky factorization is applied. Note that a single dense column in A results in a fully dense normal equations systems. Thus, in the absence of a mechanism for handling dense columns, this approach may be impractical for some large problems. Finally, in Figure 4.1d, the augmented system is solved using an LDL^T factorization, computed by `LDLFactorizations`.

4.7 Computational results

In this section, we compare Tulip to several open-source and commercial solvers, focusing on those that are available to Julia users. Let us emphasize that our goal is *not* to perform a comprehensive benchmark of interior-point LP solvers.

We evaluate Tulip's performance and robustness in the following three settings. First, in Section 4.7.1, we consider general LP instances from H. Mittelmann's benchmark,[5] which are solved using generic sparse linear algebra. Then, in Section 4.7.2, we consider structured instances that arise in decomposition methods, for which we develop specialized linear algebra. Finally, in Section 4.7.3, we illustrate Tulip's ability to use different levels of arithmetic precision by solving problems in higher precision.

[5] `http://plato.asu.edu/ftp/lpbar.html`

```julia
model = Tulip.Model{Float64}()              # Instantiate model
Tulip.load_problem!(model, "problem.mps")   # Read problem

Tulip.optimize!(model)                      # Solve the problem
```

(a) Sample code using default linear algebra settings

```julia
model = Tulip.Model{Float64}()              # Instantiate model
Tulip.load_problem!(model, "problem.mps")   # Read problem

# Select dense linear algebra
model.params.MatrixOptions = MatrixOptions(Matrix)
model.params.KKTOptions = SolverOptions(Dense_SymPosDef)

Tulip.optimize!(model)                      # Solve the problem
```

(b) Sample code using dense linear algebra

```julia
model = Tulip.Model{Float64}()              # Instantiate model
Tulip.load_problem!(model, "problem.mps")   # Read problem

# Solve the normal equations with CHOLMOD
model.params.KKTOptions = SolverOptions(CholmodSolver, normal_equations=true)

Tulip.optimize!(model)                      # Solve the problem
```

(c) Sample code using CHOLMOD to solve the normal equations system

```julia
model = Tulip.Model{Float64}()              # Instantiate model
Tulip.load_problem!(model, "problem.mps")   # Read problem

# Solve the augmented system with LDLFactorizations
model.params.KKTOptions = SolverOptions(LDLFact_SymQuasDef)

Tulip.optimize!(model)                      # Solve the problem
```

(d) Sample code using `LDLFactorizations`

Figure 4.1 Code examples for reading and solving a problem with various linear algebra implementations.

4.7.1 Results on general LP instances

We select all instances from H. Mittelmann's benchmark of barrier LP solvers, except `qap15` and `L1_sixm1000obs`. The former is identical to `nug15`, and the latter could not be solved by any solvers in the prescribed time limit. This yields a testset of 43 medium to large-scale instances. We compare the following open-source and commercial solvers: Clp 1.17 [113], GLPK 4.64 [114], ECOS 2.0 [115], Tulip 0.5.0, CPLEX 12.10 [17], Gurobi 9.0 [116] and Mosek 9.2 [117]. All are accessed through their respective Julia interface. We run the interior-point algorithm of each solver with a single thread, no crossover, and a $10,000$s time limit. For Tulip, the maximum number of IPM iterations is increased from the default 100 to 500. All other parameters are left to their default values.

Experiments are carried out on a cluster of machines equipped with dual Intel Xeon 6148-2.4GHz CPUs, and varying amounts of RAM. Each job is run with a single thread and 16GB of memory. Scripts for running these experiments are available online,[6] together with the logfiles of each solver.

Computational results are displayed in Table 4.1. For each solver, we report the total number of instances solved, the mean runtime, and individual runtimes for each instance. Segmentation faults are indicated by `seg`, timeouts by `t`, other failures by `f`, and reduced accuracy solutions by `r`. The time to read in the data is not included.

Mean runtimes are shifted geometric means

$$\mu_\delta(t_1, ..., t_N) = \left(\prod_{i=1}^{N}(t_i + \delta)\right)^{\frac{1}{N}} - \delta = \exp\left[\frac{1}{N}\sum_{i=1}^{N}\log(t_i + \delta)\right] - \delta,$$

with $\delta = 10$ seconds.

First, the three commercial solvers CPLEX, Gurobi and Mosek display similar performance and robustness, and outperform open-source alternatives by one to two orders of magnitude. While CPLEX and Gurobi encountered numerical issues on a few instances, we found that these were resolved by activating crossover.

Second, Clp displays a worse performance than expected, solving only 25 problems with an average runtime about two times larger than Tulip's. In fact, out of 43 instances, we recorded 5 segmentation faults, 8 unidentified errors, with the $10,000$s time limit being reached on the remaining 10 unsolved instances. A more detailed analysis of the log suggests that segmentation faults and some unknown errors are caused by memory-related issues, i.e., large Cholesky factors that do not fit in memory. We note that those errors do not occur

[6]`https://github.com/mtanneau/LPBenchmarks`

Table 4.1 Results on the Mittelmann test set

Problem	Clp	CPLEX	ECOS	GLPK	Gurobi	Mosek	Tulip
Solved	25	39	26	6	41	43	33
Average	1607.5	52.4	2344.7	7092.8	42.5	32.0	604.6
L1_sixm250obs	seg	23.7	f	f	f	148.8	f
Linf_520c	seg	25.9	f	seg	30.4	22.7	t
brazil3	2.3	0.2	f	f	0.6	0.6	2.3
buildingenergy	f	18.3	362.0	f	19.9	16.6	39.1
chrom1024-7	seg	0.3	96.8	f	1.5	3.2	3.6
cont1	2322.4	5.2	138.0	f	5.9	12.6	26.9
cont11	626.8	f	174.7	f	14.8	12.7	51.4
dbic1	118.5	9.6	332.9	f	9.2	9.4	30.2
degme	t	235.5	f	f	308.3	253.0	t
ds-big	237.1	29.6	331.6r	f	31.1	16.5	95.2
ex10	t	6.3	f	f	46.8	21.6	5427.5
fome13	413.3	19.0	1284.0	f	17.8	16.9	538.2
irish-e	363.6	21.1	f	f	16.5	20.3	35.4
karted	6509.1	89.1	5801.8	9334.9	115.7	44.4	3786.9
neos	433.4	26.7	1201.0r	seg	39.5	33.1	419.5
neos1	f	4.9	167.2	f	6.3	4.1	130.4
neos2	f	4.0	129.6	f	4.7	3.4	462.1
neos3	seg	26.5	1282.9	f	33.7	17.1	1358.2
neos5052403	2067.9	43.6	3489.4r	f	28.1	13.1	475.4
ns1644855	t	334.2	f	f	437.0	468.0	t
ns1687037	t	f	f	f	19.6	12.3	330.4
ns1688926	t	25.7	f	f	f	2.0	f
nug08-3rd	t	3.4	f	f	2.8	55.0	f
nug15	352.2	12.7	1871.4	f	0.9	17.8	998.6
pds-100	t	168.2	f	f	106.2	152.7	f
pds-40	1303.7	25.6	f	f	22.1	32.9	5000.6
psched3-3	t	86.6	f	f	147.5r	148.5	t
rail02	t	208.9	f	f	134.3	195.8	t
rail4284	5407.8	72.2	8578.3r	f	133.5	81.0	1049.5
s100	1587.5	29.7	f	f	38.1	30.2	894.2
s250r10	263.5	17.8	f	f	25.4	30.8	257.2
savsched1	183.9	27.1	2355.4	f	25.9	55.5	138.8
self	21.5	3.2	162.3	45.1	4.0	3.1	13.3
shs1023	286.2	f	f	f	48.2	74.6	371.6
square41	202.8	3.4	1703.3r	f	4.9	32.7	134.0
stat96v1	164.9	f	163.6r	f	32.2r	6.3	41.3
stat96v4	1.4	1.0	31.4	261.5	1.0	2.2	1.8
stormG2_1000	seg	64.2	9593.0	f	122.6	131.9	216.3
stp3d	1864.1	31.6	1363.3	f	31.9	39.2	529.7
support10	f	17.1	6210.2	f	19.3	27.2	3553.1
tp-6	7872.1	150.9	f	f	203.8	312.3	5543.0
ts-palko	1757.1	35.9	1109.3	2315.7	50.7	31.7	841.1
watson_2	65.0	24.4	f	111.0	26.6	30.1	f

seg: segmentation fault; r: reduced accuracy solution; t: time limit; f: other failure
All times in seconds.

when running Clp through its command-line executable: the executable performs additional checks to decide whether the model should be dualized; this can yield smaller linear systems and thus avoid memory issues. Nevertheless, given that the `dualize` option is not available in Clp's C interface[7], on which Clp's Julia wrapper is built, the present results best represent the behavior that Julia users would encounter.

Third, among open-source solvers, Tulip is the top performer with 33 instances solved and a mean runtime of 604.6s, while GLPK has the worst performance with only 6 instances reportedly solved. Tulip's 5 failures include 3 instances that out of memory; for the remaining 2, i.e., `ns1688926` and `watson_2`, Tulip fails to reach the prescribed accuracy due to numerical issues. A possible remedy to the latter will be discussed in Section 4.7.3. Finally, out of the 26 instances reported as solved by ECOS, 6 were solved to reduced accuracy. This situation typically corresponds to ECOS encountering numerical issues close to optimality, but a feasible or close-to-feasible solution is still available.

4.7.2 Results on structured LP instances

We now compare Tulip to state-of-the-art commercial solvers on a collection of structured problems, for which we design specialized linear algebra routines. Specifically, we consider the context of Dantzig-Wolfe (DW) decomposition [13] in conjunction with a column-generation (CG) algorithm; we refer to [15] for a thorough overview of DW decomposition and CG algorithms. Here, we focus on the resolution of the master problem, i.e., we consider problems of the form

$$
\begin{aligned}
(MP) \quad \min_{\lambda} \quad & \sum_{r=1}^{R} \sum_{j=1}^{n_r} c_{r,j} \lambda_{r,j} + c_0^T \lambda_0 && (4.77)\\
s.t. \quad & \sum_{j=1}^{n_r} \lambda_{r,j} = 1, \quad r = 1, ..., R, && (4.78)\\
& \sum_{r=1}^{R} \sum_{j=1}^{n_r} a_{r,j} \lambda_{r,j} + A_0 \lambda_0 = b_0, && (4.79)\\
& \lambda \geq 0, && (4.80)
\end{aligned}
$$

where R is the number of sub-problems, m_0 is the number of linking constraints, n_r is the number of columns from sub-problem r, $A_0 \in \mathbb{R}^{m_0 \times n_0}$, and $\forall (r,j), a_{r,j} \in \mathbb{R}^{m_0}$. Let $M = R + m_0$ and $N = n_0 + n_1 + \cdots + n_R$ be the number of constraints and variables in (MP), respectively. In what follows, we focus on the case where (i) R is large, typically in

[7]See discussion in `https://github.com/coin-or/Clp/issues/151`

the thousands or tens of thousands, (ii) m_0 is not too large, typically in the hundreds, and (iii) the vectors $a_{r,j} \in \mathbb{R}^{m_0}$ and A_0 are dense.

Instance collection

We build a collection of master problems from two sources. First, we generate instances of Distributed Energy Resources (DER) coordination from [26]. We select a renewable penetration rate $\xi = 0.33$, a time horizon $T = \{24, 48, 96\}$, and a number of resources $R = \{1024, 2048, 4096, 8192, 16384, 32768\}$. Second, we select all two-stage stochastic programming (TSSP) problems from [79] that have at least $1,000$ scenarios. This yields 18 DER instances, and 27 TSSP instances.

Then, each instance is solved by column generation; master problems are solved with Gurobi's barrier (with crossover) and sub-problems are solved with Gurobi's default settings. In the case of DER instances, which contain mixed-integer variables, only the root node of a branch-and-price tree is solved. Finally, at every tenth CG iteration and the last, the current master problem is saved. Thus, we obtain a dataset of 153 master problems of varying sizes.

CG algorithms benefit from sub-optimal, well-centered interior solutions from the master problem [80], which are typically obtained by simply relaxing an IPM solver's optimality tolerance. These provide the double benefit of stabilizing the CG procedure, thus reducing the number of CG iterations, and speeding-up the resolution of the master problem by stopping the IPM early. Importantly, this approach requires *feasible*, but sub-optimal, dual solutions from the master problem. While in classical primal-dual IPMs, feasibility is generally reached earlier than optimality, in the homogeneous algorithm, infeasibilities and complementarity are reduced at the same rate [105]. As a consequence, for IPM solvers that implement the homogeneous algorithm, such as Mosek, ECOS and Tulip, relaxing optimality tolerances yields no computational gain. Nevertheless, let us formally restate that our present goal is *not* to implement a state-of-the-art column-generation solver, but to quantify the benefits of specialized linear algebra in that context; in particular, specialized linear algebra would equally benefit classical primal-dual IPMs, since the approach of [80] does not affect the master problem's structure. Therefore, we only implement a vanilla CG procedure, which is described in Appendix A. In particular, we do not make use of any acceleration technique beyond the use of partial pricing.

Table 4.2 and Table 4.3 display some statistics for DER and TSSP instances, respectively. For each instance, we report: the number of sub-problems R, the number of CG iterations (Iter), total time spent solving the master problem (Master) and pricing sub-problems (Pricing) during the CG procedure and, for the final (MP): the number of linking constraints (m_0), the number of variables (N), and the proportion of non-zero coefficients in the linking constraints (%nz). From the two tables, we see that `DER`, `4node` and `4node-base` instances display relatively dense linking rows, with 35 to 90% coefficients being non-zeros, and a modest number of linking constraints. Other instances are either sparser, e.g., the `env` and `env-diss` instances whose linking rows are only 13% dense, or have few linking constraints, e..g, `phone`. Therefore, we expect that our specialized implementation will yield larger gains for the former instances.

Specialized linear algebra

We now describe a specialized Cholesky factorization that exploits the block structure of the master problem. First, the constraint matrix of (MP) is unit block-angular, i.e., it has the form

$$A = \begin{bmatrix} e^T & & & 0 \\ & \ddots & & \vdots \\ & & e^T & 0 \\ A_1 & \cdots & A_R & A_0 \end{bmatrix}, \tag{4.81}$$

where

$$A_r = \begin{pmatrix} | & & | \\ a_{r,1} & \dots & a_{r,n_r} \\ | & & | \end{pmatrix} \in \mathbb{R}^{m_0 \times n_r}. \tag{4.82}$$

Let us recall that the normal equations system writes

$$\Big(A(\Theta^{-1} + \rho_p I)^{-1} A^T + \rho_d I\Big)\, \delta_y = \xi, \tag{4.83}$$

where $\delta_y \in \mathbb{R}^M$, and $\Theta \in \mathbb{R}^{N \times N}$ is a diagonal matrix with positive diagonal. Let S denote the left-hand matrix of (4.83), and define

$$\tilde{\Theta} = (\Theta^{-1} + \rho_p I)^{-1} = \begin{pmatrix} \tilde{\Theta}_1 & & & \\ & \ddots & & \\ & & \tilde{\Theta}_R & \\ & & & \tilde{\Theta}_0 \end{pmatrix}, \tag{4.84}$$

and $\tilde{\theta}_r = \tilde{\Theta}_r e \in \mathbb{R}^{n_r}$, for $r = 0, ..., R$. Consequently, the normal equations system has the

Table 4.2 Column-generation statistics - DER instances

		CG statistics			MP statistics		
Instance	R	Iter	Master(s)	Pricing(s)	m_0	N	%nz
DER-24	1024	43	4.5	16.6	24	6493	89.7
	2048	40	9.7	40.8	24	12152	89.0
	4096	41	24.0	86.5	24	24559	89.4
	8192	40	74.9	155.9	24	48668	89.7
	16384	42	195.8	419.9	24	95845	90.1
	32768	40	585.7	826.3	24	192039	89.6
DER-48	1024	49	10.8	25.7	48	7440	87.0
	2048	49	24.6	50.7	48	14736	88.0
	4096	49	60.8	103.2	48	29328	88.3
	8192	50	148.1	212.3	48	59536	88.5
	16384	48	355.4	418.3	48	114832	88.5
	32768	47	853.8	870.2	48	225424	88.4
DER-96	1024	64	49.0	67.9	96	9504	86.7
	2048	56	90.8	117.0	96	16672	87.8
	4096	53	191.7	220.1	96	31520	88.2
	8192	60	603.4	529.9	96	69920	88.5
	16384	57	1248.7	993.0	96	133408	89.0
	32768	54	3657.2	2163.7	96	254240	88.7

Table 4.3 Column-generation statistics - TSSP instances

		CG statistics			MP statistics		
Instance	R	Iter	Master(s)	Pricing(s)	m_0	N	%nz
4node	1024	24	3.4	2.9	60	5997	41.5
	2048	24	7.9	7.2	60	11614	38.8
	4096	22	14.4	14.0	60	22034	38.0
	8192	23	43.8	28.1	60	44691	37.3
	16384	23	114.8	58.0	60	87569	37.9
	32768	21	248.7	95.8	60	158895	36.5
4node-base	1024	26	4.5	2.8	60	6197	59.4
	2048	27	11.9	5.5	60	12968	60.2
	4096	25	35.8	10.5	60	24153	60.3
	8192	22	46.6	17.4	60	43399	59.4
	16384	25	143.8	45.0	60	95792	60.4
	32768	23	321.6	79.8	60	179472	60.2
assets	37500	6	2.1	6.2	13	77928	38.5
env	1200	6	0.1	0.5	85	2860	12.5
	1875	6	0.1	0.7	85	4283	12.9
	3780	6	0.2	1.5	85	8357	13.3
	5292	6	0.2	2.1	85	11541	13.5
	8232	6	0.4	3.3	85	17664	13.6
	32928	6	2.5	13.2	85	69783	13.8
env-diss	1200	13	0.2	0.7	85	4439	12.3
	1875	15	0.4	1.3	85	7435	12.7
	3780	15	1.0	2.6	85	15168	12.9
	5292	15	1.5	3.6	85	20745	13.0
	8232	15	2.5	5.7	85	31752	13.0
	32928	14	14.8	21.8	85	123892	13.3
phone	32768	5	1.4	7.6	9	65553	83.3
stormG2	1000	21	7.9	10.9	306	6075	23.9

form

$$\begin{bmatrix} d_1 & & & (A_1\tilde{\theta}_1)^T \\ & \ddots & & \vdots \\ & & d_R & (A_R\tilde{\theta}_R)^T \\ A_1\tilde{\theta}_1 & \cdots & A_R\tilde{\theta}_R & \Phi \end{bmatrix} \begin{bmatrix} (\delta_y)_1 \\ \vdots \\ (\delta_y)_R \\ (\delta_y)_0 \end{bmatrix} = \begin{bmatrix} \xi_1 \\ \vdots \\ \xi_R \\ \xi_0 \end{bmatrix}, \tag{4.85}$$

where

$$d_r = e^T\tilde{\theta}_r + \rho_d, \quad r = 1, ..., R, \tag{4.86}$$

$$\Phi = \sum_{r=0}^{R} A_r\tilde{\Theta}_r A_r^T + \rho_d I. \tag{4.87}$$

Then, define

$$l_r = \frac{1}{d_r} A_r\tilde{\theta}_r \in \mathbb{R}^{m_0}, \quad r = 1, ..., R, \tag{4.88}$$

$$C = \Phi - \sum_{r=1}^{R} \frac{1}{d_r}(A_r\tilde{\theta}_r)(A_r\tilde{\theta}_r)^T \in \mathbb{R}^{m_0 \times m_0}. \tag{4.89}$$

Given that both S and its upper-left block are positive definite, so is the Schur complement C. Therefore, its Cholesky factorization exists, which we denote $C = L_C D_C L_C^T$. It then follows that a Cholesky factorization of S is given by

$$S = \underbrace{\begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ l_1 & \cdots & l_R & L_C \end{bmatrix}}_{L} \times \underbrace{\begin{bmatrix} d_1 & & & \\ & \ddots & & \\ & & d_R & \\ & & & D_C \end{bmatrix}}_{D} \times \underbrace{\begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ l_1 & \cdots & l_R & L_C \end{bmatrix}^T}_{L^T}. \tag{4.90}$$

Finally, once the Cholesky factors L and D are computed, the normal equations (4.85) are solved as follows:

$$(\delta_y)_0 = (L_C D_C L_C^T)^{-1} \left(\xi_0 - \sum_{r=1}^{R} \xi_r l_r \right), \tag{4.91}$$

$$(\delta_y)_r = \frac{1}{d_r}\xi_r - l_r^T(\delta_y)_0, \quad r = 1, ..., R. \tag{4.92}$$

Exploiting the structure of A yields several computational advantages. First, the factors L and D can be computed directly from A and Θ, i.e., the matrix S does not need to be explicitly formed nor stored, thus saving both time and memory. Second, the sparsity structure of L is known beforehand. Specifically, the lower blocks $l_1, \dots, l_R$ are all dense column vectors, and the Schur complement C is a dense $m_0 \times m_0$ matrix. Therefore, one does not need a preprocessing phase wherein a sparsity-preserving ordering is computed, thus saving time and making memory allocation fully known in advance. Third, since most heavy operations are performed on dense matrices, efficient cache-exploiting kernels for dense linear algebra can be used, further speeding-up the computations. Finally, note that most operations such as forming the Cholesky factors and performing the backward substitutions, are amenable to parallelization.

Experimental setup

We implement the specialized routines described above in Julia.[8] Specifically, we define a `UnitBlockAngularMatrix` type, together with specialized matrix-vector product methods, and a `UnitBlockAngularFactor` type for computing factorizations and solving linear systems. Dense linear algebra operations are performed by BLAS/LAPACK routines directly, and the entire implementation is less than 250 lines of code.

This specialized implementation is passed to the solver by setting the `MatrixOptions` and `KKTOptions` parameters accordingly, as illustrated in Figure 4.2. A `Model` object is first created at line 4, and the problem data is imported at line 5. At line 11, we set the `MatrixOptions` parameter to specify that the constraint matrix is of the `UnitBlockAngularMatrix` type with $m_0 = 24$ linking constraints, $n_0 = 72$ linking variables, $n = 6421$ non-linking variables, and $R = 1024$ unit blocks. Then, at line 16, we select the `UnitBlockAngularFactor` type as a linear solver. Finally, the correct matrix and linear solver are instantiated within the `optimize!` call at line 20. Importantly, let us emphasize that no modification was made to Tulip's source code: the correct methods are automatically selected by Julia's multiple dispatch feature, with no performance loss for calling an external function.

Experiments are carried out on an Intel Xeon E5-2637@3.50GHz CPU, 128GB RAM machine running Linux; scripts and data for running these experiments are available online.[9] We compare the following IPM solvers: CPLEX 12.10 [17], Gurobi 9.0 [116], Mosek 9.2.5 [117], Tulip 0.5.0 with generic linear algebra, and Tulip 0.5.0 with specialized linear algebra; the latter

[8]`https://github.com/mtanneau/UnitBlockAngular.jl`

[9]Code for generating DER instances is available at `https://github.com/mtanneau/DER_experiments` and for TSPP instances at `https://github.com/mtanneau/TSSP`

is denoted Tulip*. We run each solver on a single thread, and no crossover. Presolve may alter the structure of A in several ways by, e.g., reducing the number of linking constraints, eliminating variables –possibly some entire blocks– or modifying the unit blocks during scaling. Therefore, since we are interested in comparing the per-iteration cost among solvers, we also deactivate presolve. Finally, none of the selected IPM solvers have any warm-start capability, i.e., in a CG algorithm, master problems would effectively be solved from scratch at each CG iteration. Thus, solving master problems independently of one another, as is done here, does not invalidate our analysis.

Results

Results are reported in Table 4.4 and Table 4.5 for the DER and TSSP instances, respectively; for conciseness, only the final master problem of each CG instance is included here. Results for the entire collection can be found in Table B.1, Appendix B. For each instance and solver, we report total CPU time (T), in seconds, and the number of IPM iterations (Iter). In Table B.1, the number of CG iterations (at which the instance was obtained) is also displayed.

We begin by comparing Tulip with and without specialized linear algebra. First, the number of IPM iterations is almost identical between the two, with differences never exceeding 6 IPM iterations. The differences are caused by small numerical discrepancies between the linear algebra implementations, which remain negligible until close to the optimum. Second, using specialized linear algebra results in a significant speedup, especially on larger and denser instances. Indeed, on large `DER` and `4node` instances, we typically observe a tenfold speedup. For smaller and sparser instances, e.g., the `env` instances, or with very few linking constraints such as `phone`, using specialized linear algebra still brings a moderate performance improvement.

Next, we compare Tulip with specialized linear algebra, Tulip*, against state-of-the-art commercial solvers. Given CPLEX's poorer relative performance on this test set, in the following we mainly discuss the results of Tulip* in comparison with Mosek and Gurobi. First, our specialized implementation is able to outperform commercial codes on the larger and denser instances, while remaining within a reasonable factor on smaller and sparse instances. The largest performance improvement is observed on the `DER-48` instance with $R = 32,768$, for which Tulip* achieves a 30% speedup over the fastest commercial alternative. This demonstrates that, when exploiting structure, open-source solvers can compete with state-of-the-art commercial codes. Second, Tulip's iteration count is typically 50 to 100% larger than that of Mosek and Gurobi. When comparing average per-iteration times on the denser instances, we observe that Tulip is generally 1.5 to 3 times faster than Gurobi and Mosek.

```julia
import Tulip
using UnitBlockAngular

model = Tulip.Model{Float64}()
Tulip.load_problem!(model, "DER_24_1024_43.mps")  # read file

# Deactivate presolve
model.params.Presolve = 0

# Select matrix options
model.params.MatrixOptions = Tulip.TLA.MatrixOptions(
    UnitBlockAngularMatrix,
    m0=24, n0=72, n=6421, R=1024
)
# Select custom linear solver
model.params.KKTOptions = Tulip.KKT.SolverOptions(
    UnitBlockAngularFactor
)

Tulip.optimize!(model)  # solve the problem
```

Figure 4.2 Sample Julia code illustrating the use of a custom `UnitBlockAngularMatrix` type and specialized factorization.

Table 4.4 Performance comparison of IPM solvers - DER instances

		CPLEX		Gurobi		Mosek		Tulip		Tulip*	
Problem	R	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter
DER-24	1024	0.2	33	0.2	27	**0.2**	21	1.1	33	0.5	33
	2048	0.4	48	**0.4**	36	0.4	27	2.5	47	0.6	47
	4096	1.0	40	1.0	32	**0.8**	26	4.7	38	1.0	38
	8192	4.3	79	2.7	46	**2.4**	38	19.0	67	2.6	68
	16384	10.8	93	5.3	48	**5.3**	42	49.8	86	5.3	83
	32768	33.9	148	19.4	85	12.3	43	103.6	91	**11.4**	86
DER-48	1024	0.5	37	0.4	19	**0.3**	21	1.8	28	0.4	28
	2048	1.6	40	1.0	21	**0.8**	25	5.7	37	0.9	37
	4096	4.1	44	2.0	25	2.0	27	14.1	39	**1.7**	39
	8192	9.7	51	4.2	20	4.5	24	37.0	46	**3.4**	47
	16384	22.3	64	9.9	29	9.3	28	89.7	60	**7.4**	57
	32768	57.1	85	21.6	32	21.1	33	178.8	59	**14.2**	54
DER-96	1024	3.3	38	1.2	19	0.9	22	6.6	31	**0.9**	31
	2048	7.9	45	2.4	20	1.7	21	18.2	38	**1.7**	37
	4096	16.3	51	5.5	24	5.2	28	42.6	40	**3.2**	40
	8192	51.7	75	15.5	29	11.1	31	137.6	60	**8.8**	57
	16384	107.8	86	31.9	31	24.4	39	260.0	55	**17.3**	59
	32768	291.9	119	102.9	54	**55.5**	47	753.7	89	65.4	86

Table 4.5 Performance comparison of IPM solvers - TSSP instances

		CPLEX		Gurobi		Mosek		Tulip		Tulip*	
Problem	R	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter
4node	1024	15.7	21	0.4	43	**0.2**	25	1.3	30	0.5	32
	2048	0.7	38	0.6	27	**0.6**	25	2.1	36	0.9	36
	4096	1.1	27	1.7	37	**0.7**	17	4.5	28	1.2	28
	8192	2.7	30	2.3	29	**1.8**	24	12.3	35	2.7	33
	16384	5.8	29	10.7	53	**4.0**	22	26.4	33	4.6	33
	32768	17.0	57	18.7	55	14.6	41	74.8	56	**14.2**	59
4node-base	1024	17.0	17	1.0	60	**0.3**	27	1.4	28	0.6	27
	2048	1.0	35	2.6	72	**0.8**	33	3.7	32	0.9	33
	4096	2.3	38	5.3	72	**1.5**	34	9.1	34	1.8	34
	8192	3.8	29	3.7	27	**2.6**	25	19.7	36	2.8	36
	16384	13.5	53	26.2	74	8.0	37	63.4	53	**7.0**	47
	32768	20.3	37	29.0	43	14.9	30	107.7	48	**12.9**	50
assets	37500	1.6	21	**0.6**	12	1.1	20	2.0	13	1.0	13
env	1200	0.0	21	**0.0**	12	0.1	16	0.3	16	0.3	16
	1875	0.1	22	**0.0**	12	0.1	13	0.4	16	0.4	16
	3780	0.1	25	**0.1**	12	0.1	14	0.7	17	0.5	17
	5292	0.2	27	**0.1**	13	0.1	13	0.7	17	0.7	17
	8232	0.3	26	**0.2**	13	0.3	14	1.1	18	1.2	18
	32928	1.7	26	**0.9**	13	1.3	17	5.1	21	4.4	21
env-diss	1200	0.1	15	**0.0**	15	0.1	17	0.4	23	0.4	23
	1875	0.1	17	**0.1**	18	0.1	18	0.6	22	0.5	22
	3780	0.2	20	**0.1**	18	0.2	18	1.0	22	1.0	22
	5292	0.3	22	**0.3**	23	0.3	22	1.3	25	1.5	25
	8232	1.0	31	0.6	29	**0.5**	23	3.2	35	2.6	35
	32928	4.8	28	**2.1**	22	2.5	19	10.0	27	7.7	27
phone	32768	0.5	15	**0.4**	8	0.6	8	1.9	10	0.7	10
stormG2	1000	1.6	37	0.8	18	**0.5**	22	4.0	29	1.7	28

Recall that the cost of an individual IPM iteration depends not only on problem size and the efficiency of the underlying linear algebra, but also on algorithmic features such as the number of corrections, which we cannot measure directly. Nevertheless, the performance difference is significant enough to suggest that algorithmic improvements aimed at reducing the number of IPM iterations would substantially improve Tulip's performance.

4.7.3 Solving problems in extended precision

Almost all optimization solvers perform computations in double precision (64 bits) floating-point arithmetic, denoted by `double` and `Float64` in C and Julia, respectively. Julia's parametric type system and multiple dispatch allow to write generic code: in the present case, this results in Tulip's code can be used with *arbitrary* arithmetic. We now illustrate this functionality for solving problems in higher precision.

The ability to use extended precision is useful is various contexts. First, while typical numerical tolerances for most LP solvers range from 10^{-6} to 10^{-8}, one may *require* levels of precision that exceed what double-precision arithmetic can achieve. For instance, in [118], the authors consider problems where variations of order 10^{-6} to 10^{-10} are meaningful. One remedy to this issue is to use, e.g., quadruple-precision arithmetic. Second, even with "standard" tolerances, solvers may encounter numerical issues for badly scaled problems, sometimes resulting in the optimization being aborted. These issues may be alleviated by using higher precision, thereby allowing to solve a given challenging instance, albeit at a performance cost. Finally, in the course of developing a new optimization software or algorithmic technique, identifying whether inconsistencies are due to numerical issues, mathematical errors, or software bugs, can be a daunting and time-consuming task. In that context, the ability to easily switch between different arithmetics enables one to factor out rounding errors and related issues, thereby identifying –or ruling out– other sources of errors.

Let us note that a handful of simplex-based solvers have the capability to compute extended-precision or exact solutions to LP problems, either by performing computations in exact arithmetic, solving a sequence of LPs with increasing precision, or using iterative refinement techniques; the reader is referred to [119] for an overview of such approaches and available software. We are not aware of any existing interior-point solver with this capability. As pointed out in [119], performing all computations in the prescribed arithmetic, as is the case in Tulip, is intractable for large problems. Consequently, Tulip should not be viewed as a competitive tool for solving LPs in extended precision. Rather, the main advantage of our implementation is its simplicity and flexibility: it required no modification of the source code, runs the same algorithm regardless of the arithmetic, and its use is straightforward. Indeed, as Figure 4.3 illustrates, besides loading the appropriate packages, the user only needs to specify the arithmetic when creating a model; the rest of the code is identical. Therefore, using Tulip with higher-precision arithmetic is best envisioned as a prototyping tool, or to occasionally solve a numerically challenging problem.

As an example of this use case, we consider the 6 instances from Section 4.7.1 that required more than 100 IPM iterations; this generally indicates numerical issues. Each instance is solved with Tulip in quadruple-precision arithmetic. We use the `Double64` type from the `DoubleFloats` Julia package, which implements the so-called “double-double" arithmetic, wherein a pair of double-precision numbers is used to approximate one quadruple-precision number. This implementation allows to exploit fast, hardware-implemented, double-precision arithmetic, while achieving similar precision as 128 bits floating point arithmetic. Experiments were carried on the same cluster of machines as in Section 4.7.1. Besides the different arithmetic, we increase the time limit to $40,000$s and set tolerances to 10^{-8}, that is, the problems are solved up to usual double-precision tolerances. All other settings are left identical to those of Section 4.7.1.

Results are displayed in Table 4.6. For each instance and arithmetic, we report the total solution time (CPU) in seconds, the number of IPM iterations (Iter), and the solver's result status (Status). We first note that, when using `Double64` arithmetic, all instances are solved to optimality. This validates the earlier finding that instances `ns1688926` and `watson_2` did encounter numerical issues. Second, we observe a drastic reduction in the number of IPM iterations from `Float64` to `Double64`, with decreases in iteration counts ranging from 40% to over 90% in the case of `neos2` and `ns1688926`. Third, while the per-iteration cost of `Double64` is typically 8x larger than that of `Float64`, overall computing times do not increase as much due to the reduction in IPM iterations. In fact, in the extreme cases of `ns1688926`, solving the problem in `Double64` is significantly faster than solving it in `Float64`. Finally, the results of Table 4.6 suggest that Tulip would most benefit from greater numerical stability on instances such as `neos2`, `ns1688926`, `stat96v1` and `watson_2`. This may include, for instance, the use of iterative refinement when solving Newton systems. On the other hand, similar iterations counts for both arithmetics, would have suggested algorithmic issues, e.g., short steps being taken due to the iterates being far from the central path.

4.8 Conclusion

In this paper, we have described a regularized homogeneous interior-point algorithm and its implementation in Tulip, an open-source linear optimization solver. Our solver is written in Julia, and leverages some of the language's features to propose a flexible and easily-customized

```julia
import Tulip

model = Tulip.Model{Float64}()  # Float64 arithmetic
Tulip.load_problem!(tlp, "neos2.mps")  # read file

Tulip.optimize!(model)  # solve the problem
```

(a) Using `Float64` arithmetic.

```julia
import Tulip
using DoubleFloats

model = Tulip.Model{Double64}()  # Double64 arithmetic
Tulip.load_problem!(tlp, "neos2.mps")  # read file

Tulip.optimize!(model)  # solve the problem
```

(b) Using `Double64` arithmetic.

Figure 4.3 Sample Julia code illustrating the use of different arithmetics.

Table 4.6 Problematic instances from the Mittelmann benchmark

	`Float64`			`Double64`		
Instance	CPU (s)	Iter	Status	CPU(s)	Iter	Status
`neos2`	462.1	460	Optimal	265.1	37	Optimal
`ns1688926`	1007.7	500	Iterations	142.8	18	Optimal
`s250r10`	257.2	169	Optimal	1385.0	93	Optimal
`shs1023`	371.6	266	Optimal	968.8	105	Optimal
`stat96v1`	41.3	275	Optimal	30.4	42	Optimal
`watson_2`	295.7	500	Iterations	243.4	67	Optimal

implementation. Most notably, Tulip's algorithmic framework is fully disentangled from linear algebra implementations and the choice of arithmetic.

The performance of the code has been evaluated on generic instances from H. Mittelmann's benchmark testset, on two sets of structured instances for which we developed specialized linear algebra routines, and on numerically problematic instances using higher-precision arithmetic. The computational evaluation has shown three main results. First, when solving generic LP instances, Tulip is competitive with open-source IPM solvers that have a Julia interface. Second, when solving structured problems, the use of custom linear algebra routines yields a tenfold speedup over generic ones, thereby outperforming state-of-the-art commercial IPM solvers on larger and denser instances. These results demonstrate the benefits of being able to seamlessly integrate specialized linear algebra within an interior-point algorithm. Third, in a development context, Tulip can be conveniently used in conjunction with higher-precision arithmetic, so as to alleviate numerical issues.

Finally, future developments will consider the use of iterative methods for solving linear systems, the development of more general structured linear algebra routines and their multi-threaded implementation, and more efficient algorithmic techniques for solving problems in extended precision. Because of the way in which Tulip has been designed, all those developments do not require any significant rework of the code structure.

CHAPTER 5 ARTICLE 3 - DISJUNCTIVE CUTS IN MIXED-INTEGER CONIC OPTIMIZATION

Authors: Andrea Lodi, Mathieu Tanneau and Juan-Pablo Vielma

Submitted to *Mathematical Programming.* Preprint: [120].

Abstract This paper studies disjunctive cutting planes in Mixed-Integer Conic Programming. Building on conic duality, we formulate a cut-generating conic program for separating disjunctive cuts, and investigate the impact of the normalization condition on its resolution. In particular, we show that a careful selection of normalization guarantees its solvability and conic strong duality. Then, we highlight the shortcomings of separating conic-infeasible points in an outer-approximation context, and propose conic extensions to the classical lifting and monoidal strengthening procedures. Finally, we assess the computational behavior of various normalization conditions in terms of gap closed, computing time and cut sparsity. In the process, we show that our approach is competitive with the internal lift-and-project cuts of a state-of-the-art solver.

5.1 Introduction

Mixed-Integer Convex Optimization (MI-CONV) is a fundamental class of Mixed-Integer Nonlinear Optimization problems with applications such as risk management, nonlinear physics (e.g., power systems and chemical engineering) and logistics, just to mention a few. Because of such a relevance, classical algorithms for Mixed-Integer Linear Optimization (MILP) have been successfully extended to MI-CONV, like Branch and Bound [121] or Benders decomposition [122]; others like the Outer Approximation scheme [123] have been designed specifically for MI-CONV. In addition, several software tools are available for solving general MI-CONV problems, see, e.g., the recent comparison in [124]. Finally, some specific classes of MI-CONV problems, like Mixed-Integer (Convex) Quadratically Constrained Quadratic Optimization (MIQCQP) problems are now supported by the major commercial solvers.

Conic optimization is viewed as a more numerically stable and tractable alternative to general convex optimization [125]. Both classes are equivalent: conic optimization problems are convex, and any convex optimization problem can be written as a conic optimization problem [126]. Modeling tools such as disciplined convex optimization [127] can provide conic formulations for most –if not all– convex optimization problems that arise in practice [128].

In particular, [129] recently showed that all convex instances in MINLPLib can be formulated as Mixed-Integer Conic Optimization (MI-CONIC) problems using only a handful of cones.

Nevertheless, the intrinsic difference between convex and conic optimization lies in a problem's algebraic description: in the former, constraints are formulated as $f(x) \leq 0$, where f is a convex function, whereas, in the latter, they are expressed using conic inequalities of the form $Ax - b \in \mathcal{K}$, where A is a matrix, b is a vector and $\mathcal{K}$ is a cone (see [125] and Section 5.2). In particular, conic formulations enable the use of conic duality theory, which underlies a number of theoretical insights and practical tools. Major commercial solvers have supported Mixed-Integer Second Order Cone Programming (MISOCP) for some time, and more general MI-CONIC problems are now supported by a number of solvers, e.g., Mosek and Pajarito [129–131].

This paper builds on two specific aspects that we consider fundamental for solving MI-CONV problems. First, given that cutting planes are instrumental to solving MILP, a number of authors have looked at various approaches to compute cuts for MI-CONV problems and, nowadays, linear cutting planes are part of the arsenal of some MI-CONV solvers. Despite this (partial) success, some fundamental questions in this area are left unanswered. Second, recent experience has shown that *conic* formulations of MI-CONV problems display enviable properties that make them preferable, from the solving viewpoint, to generic MI-CONV formulations.

Therefore, building on *(i)* cutting planes and *(ii)* conic formulations, we answer the (somehow) natural question of what one can gain in terms of cutting planes by using a problem's *conic* structure. In the process of doing so, we answer several questions left open by previous works on the topic. Motivated by the success of disjunctive cuts in MILP, the paper focuses on computational aspects of disjunctive cuts for MI-CONIC problems. In the remainder of this section, we review the literature on the subject and outline our main contributions.

5.1.1 Disjunctive cuts: the MILP case

Disjunctive cuts in MILP go back to Balas' seminal work on disjunctive programming [132] in the 70s, and became widely popular as their integration into branch-and-cut frameworks [133, 134] proved effective. Remarkably, disjunctive cuts, split cuts in particular, encompass several classes of cutting planes, e.g., Chvatal-Gomory, Gomory Mixed-Integer and Mixed-Integer Rounding cuts.

A general approach for separating disjunctive cuts in MILP is the so-called Cut-Generating Linear Program (CGLP) proposed by Balas [132, 133]. The CGLP leverages a characterization of valid inequalities for disjunctive sets using Farkas multipliers, see Theorem 3.1 in [132]. Thus, it is formulated in a higher-dimensional space, whose size is proportional to the number of disjunctive terms: for split cuts, which are two-term disjunctions, the CGLP is roughly double the size of the original problem.

Computational aspects of the CGLP have been studied extensively, some of which we mention here. Given a fractional point $\bar{x}$ to separate, one can project the CGLP onto the support of $\bar{x}$, thereby reducing its size, and recover a valid cut by lifting [133, 134]. Split cuts obtained from solving the CGLP can be improved upon using monoidal strengthening [134, 135]. The normalization condition in the CGLP has been shown to have a major impact on the quality of the obtained cuts, and on overall performance [136–140]. In particular, Balas and Perregard [141], and later Bonami [137], show that, in the case of split disjunctions, the CGLP can in fact be solved in the space of orignal variables only, yielding substantial computational gains. Recent developments include the efficient separation of cuts from multiple disjunctions [142, 143].

5.1.2 Disjunctive cuts: the MI-CONV case

The work on disjunctive cutting planes for MI-CONV (re)started already in the late 90s with two fundamental contributions [144, 145]. More precisely, Ceria and Soares [144] show that disjunctive convex problems can be formulated as a single convex problem in a higher dimensional space, and hint that this could serve to generate cutting planes using sub-gradient information at the optimum. Around the same time, Stubbs and Mehrotra [145] make the separation of disjunctive cuts for MI-CONV explicit by *(i)* solving one Nonlinear Programming (NLP) problem, and *(ii)* identifying a sub-gradient that yields a violated cut. The latter is done by taking a gradient (under regularity assumptions), or by solving a linear system (under the assumption that the objective function of the former problem is polyhedral). Those assumptions and the use of perspective functions lead to differentiability issues that made the results of the computational investigation in [145] numerically disappointing (according to the authors themselves).

The numerical difficulties encountered in [145] have slowed down the development of the area for a number of years –to the exception of [146]– until the renewed interest and the practical approaches of the last decade [147, 148]. More precisely, Kilinc et al. [148] note that *"A simple strategy for generating lift-and-project cuts for a MINLP problem is to solve a CGLP [...] based on a given polyhedral outer approximation of the relaxed feasible region [...]. The key question to be answered [...] is which points to use to define the polyhedral relaxation."* (from [148], Sec. 3).

The distinction in how to answer the above question is the difference between [146], [147], and [148]. Namely, Zhu and Kuno [146] build an outer-approximation through the current fractional solution, and derive a cut by solving the associated CGLP. However, this approach is not guaranteed to find a violated cut if one exists, see Example 1 in [148]. Bonami [147] solves one auxiliary NLP, and uses the solution to get an outer approximation that provably yields a violated cut if any exists. Instead, Kilinc et al. [148] iteratively refine an outer approximation by solving a sequence of LPs until a violated cut, if any, is separated by solving the associated CGLP.

The outer approximation approaches in [147, 148] are, to the best of our knowledge, the state of the art for the implementation of disjunctive cuts for MI-CONV and, especially, for MIQCQPs, see e.g., their implementation in CPLEX starting from version 12.6.2. However, despite the impressive practical improvements with respect to the early attempts [145], questions were left on the table, which we answer in the present paper.

5.1.3 Disjunctive cuts: the MI-CONIC case

Following the support of MISOCP problems by commercial MIP solvers in the 2000s (MISOCP support appeared in CPLEX 9.0 and in Gurobi 5.0), the last decade has seen a flourishing literature on cuts for MI-CONIC problems.

A large share of these works focus on cuts for MISOCP, or, equivalently, for MIQCQP problems. Atamturk and Narayanan [149] introduce conic Mixed-Integer Rounding (MIR) cuts for MISOCP problems. Modaresi et al later show in [150, 151] that conic MIR cuts are in fact linear split cuts in an extended space, and compare the strength of families of conic MIR cuts to that of nonlinear split cuts. In a related work, Andersen and Jensen [152] study intersection cuts in the MISOCP context, and obtain a closed-form formula for the conic quadratic intersection cut. Belotti et al. [153] study the intersection of a convex set and a two-term disjunction. They show that the convex hull is described by a single conic inequality, for which an explicit formula is derived in the conic quadratic case. In a similar fashion, two-term disjunctions on the second-order cone are investigated in [154], and this approach is later extended in [155].

More general approaches, i.e., not restricted to convex quadratic constraints, include [126, 156–159]. In [156], the authors study classes of cutting planes in the MI-CONIC setting, including Chvatal-Gomory cuts and lift-and-project cuts, and report limited experiments on mixed 0-1 semi-definite programming instances. A generic lifting procedure for conic cuts is described in [157]. Dadush et al. [158] show that the split closure of a strictly convex body is defined by a finite number of disjunction, but is not necessarily polyhedral. Minimal valid inequalities are introduced in [126], and are shown to be sufficient to describe the convex hull of a disjunctive conic set. The lack of tractable algebraic representation for minimal inequalities then leads the author to consider the broader class of sublinear inequalities, which are further studied in [160]. Finally, intersection cuts for non-polyhedral sets and certain classes of disjunctions are studied in [159].

Nevertheless, for the most part, these works remain theoretical contributions. Indeed, to the exception of [149,151,156], no computational results were reported for any of these techniques and, to the best of our knowledge, none has been implemented in optimization solvers. In fact, neither Mosek nor Gurobi[1] generate cuts from nonlinear information.

5.1.4 Contribution and outline

In this paper, we study linear disjunctive cutting planes for MI-CONIC problems. Our objective is to derive practical and numerically robust tools for the separation of those cuts, and we show that conic formulations allow us to achieve that goal. Specifically, we do so by extended Balas' CGLP into a Cut-Generating Conic Program (CGCP) (see also [156]). Our contributions are:

1. We study the role of the normalization condition in the CGCP, and propose conic normalizations that guarantee strong duality. In doing so, we answer some concerns that were raised in previous works. Namely,

   - With respect to [147,156], we can select the right normalization to overcome issues associated with potential lack of constraint qualification.
   - With respect to [148], since we use conic formulations, we do not need *(i)* to pay attention at avoiding generating linearization cuts at points outside the domain where the nonlinear functions are known to be convex, *(ii)* to deal with non-differentiable functions, and *(iii)* boundedness assumptions on the value of the constraints and their gradients.

2. We draw attention to the shortcomings of separating of conic-infeasible points in an outer-approximation context, and propose algorithmic strategies to alleviate them.

3. We introduce conic extensions of the lifting procedure for disjunctive cuts, and of monoidal strengthening for split cuts.

[1]Personal communication with Gurobi and Mosek developers.

4. Finally, we provide computational results on the effectiveness of the proposed approach, thereby showing the benefits of the conic representation, and compare the practical effectiveness of several normalization conditions.

The remainder of the paper is structured as follows. In Section 5.2, we introduce some required notation and background material on conic optimization, and state a number of theoretical results on the characterization of valid inequalities for conic and disjunctive conic sets. Section 5.3 formalizes the CGCP and its dual, and the theoretical properties of several normalization conditions for the CGCP are discussed in Section 5.4. The separation of conic-infeasible points is further investigated in Section 5.5, while classical lifting and strengthening techniques from MILP are extended to the conic setting in Section 5.6. In Section 5.7, we analyze the practical behavior of different normalizations, and show that our approach is competitive with CPLEX internal lift-and-project cuts. Some concluding remarks are in Section 5.8.

5.2 Background

In this section, we introduce some notations, and recall a number of results that are needed for our approach. We refer to [161] for a thorough overview of convex analysis, and to [125] and [132] for results on conic optimization and disjunctive programming, respectively.

For $\mathcal{X} \subseteq \mathbb{R}^n$, we denote by $\text{int}(\mathcal{X})$, $\partial(\mathcal{X})$, $\text{cl}(\mathcal{X})$, and $\text{conv}(\mathcal{X})$ the *interior*, *boundary*, *closure*, and *convex hull* of $\mathcal{X}$, respectively. The Minkowski sum of $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^n$ is defined by

$$\mathcal{X} + \mathcal{Y} = \{x + y \mid x \in \mathcal{X}, y \in \mathcal{Y}\}.$$

If $\|\cdot\|$ is a norm on $\mathbb{R}^n$, its *dual norm* $\|\cdot\|_*$ is defined by

$$\forall y \in \mathbb{R}^n, \ \|y\|_* = \sup\left\{y^T x \;\middle|\; \|x\| \leq 1\right\}. \tag{5.1}$$

In all that follows, $\|\cdot\|_2$ denotes the Euclidean norm on $\mathbb{R}^n$. Finally, we denote by e a vector of all ones, and by e_j a vector whose j^{th} coordinate is 1 and all others are 0; the dimension of e and e_j is always obvious from context.

5.2.1 Cones and conic duality

The set $\mathcal{K} \subseteq \mathbb{R}^n$ is a *cone* if $\forall (x, \lambda) \in \mathcal{K} \times \mathbb{R}_+,\ \lambda x \in \mathcal{K}$, and it is *irreducible* if it cannot be written as a cartesian product of irreducible cones. The *dual cone* of $\mathcal{K} \subseteq \mathbb{R}^n$ is

$$\mathcal{K}^* = \left\{ u \in \mathbb{R}^n \;\middle|\; u^T x \geq 0, \forall x \in \mathcal{K} \right\}, \tag{5.2}$$

and $\mathcal{K}$ is *self-dual* if $\mathcal{K} = \mathcal{K}^*$. A cone $\mathcal{K} \subseteq \mathbb{R}^n$ is *pointed* if $\mathcal{K} \cap (-\mathcal{K}) = \{0\}$, i.e., if it does not contain a line that passes through the origin. *Proper* cones are closed, convex, pointed cones with non-empty interior. If $\mathcal{K}$ is a proper cone, then $\mathcal{K}^*$ is also a proper cone, and any $\rho \in \operatorname{int} \mathcal{K}$ induces a norm on $\mathcal{K}^*$, denoted by $|\cdot|_\rho$ and defined by

$$|u|_\rho = \rho^T u, \quad \forall u \in \mathcal{K}^*. \tag{5.3}$$

Examples of proper cones include the non-negative orthant

$$\mathbb{R}^n_+ = \{ x \in \mathbb{R}^n \mid x \geq 0 \},$$

the second-order cone (SOC)

$$\mathcal{L}_n = \left\{ x \in \mathbb{R}^n \;\middle|\; x_1 \geq \sqrt{x_2^2 + ... + x_n^2} \right\},$$

the positive semi-definite (PSD) cone

$$\mathbb{S}^n_+ = \left\{ X \in \mathbb{R}^{n \times n} \;\middle|\; X = X^T, \lambda_{min}(X) \geq 0 \right\},$$

where $\lambda_{min}(X)$ is the smallest eigenvalue of X, and the exponential cone

$$\mathcal{E} = \operatorname{cl} \left\{ (x, y, z) \in \mathbb{R}^3 \;\middle|\; x \exp(x/y) \leq z, y > 0 \right\}.$$

The non-negative orthant, SOC and SDP cone are also self-dual, while the exponential cone is not.

A proper cone $\mathcal{K}$ induces a partial (resp. strict partial) ordering on $\mathbb{R}^n$, denoted $\succeq_\mathcal{K}$ (resp. $\succ_\mathcal{K}$) and defined by

$$\forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n,\ x \succeq_\mathcal{K} y \Leftrightarrow x - y \in \mathcal{K}, \tag{5.4}$$

$$\forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n,\ x \succ_\mathcal{K} y \Leftrightarrow x - y \in \operatorname{int}(\mathcal{K}). \tag{5.5}$$

In all that follows, we refer to $Ax \succeq_{\mathcal{K}} b$ (resp. $Ax \succ_{\mathcal{K}} b$) as a *conic* (resp. *strict conic*) inequality. Consider the system

$$Ax \succeq_{\mathcal{K}} b, \tag{5.6}$$

where $A \in \mathbb{R}^{m \times n}$ and $\mathcal{K} = \mathcal{K}_1 \times ... \times \mathcal{K}_N$ with $\mathcal{K}_i \subset \mathbb{R}^{m_i}$; correspondingly, for $y \in \mathbb{R}^m$, we write $y = (y_1, ..., y_N)$. We follow the terminology of [162], and say that system (5.6) is *feasible* if there exists $x \in \mathbb{R}^n$ such that $Ax - b \in \mathcal{K}$, and *strongly feasible* if there exists $x \in \mathbb{R}^n$ such that $Ax - b \in \mathcal{K}$ and $(Ax - b)_i \in \text{int}(\mathcal{K}_i)$ for all non-polyhedral cones $\mathcal{K}_i$, i.e., such that all non-polyhedral conic inequalities are strictly satisfied. Similarly, system (5.6) is *infeasible* if it does not admit any feasible solution, and *strongly infeasible* if, in addition, there exists $y \in \mathcal{K}^*$ such that $A^T y = 0$ and $b^T y > 0$. Furthermore, we say that system (5.6) is *weakly feasible* if it is feasible but not strongly feasible, and *weakly infeasible* if it is infeasible but not strongly infeasible. Finally, a system is *well-posed* if it is either strongly feasible or strongly infeasible, and *ill-posed* otherwise.

Let us emphasize that well-posedness is an algebraic property, i.e., it is not associated to a geometric set but to its algebraic representation through conic inequalities. For instance, for $n \geq 3$, both $0 \succeq_{\mathbb{R}^n_+} x \succeq_{\mathbb{R}^n_+} 0$ and $0 \succeq_{\mathcal{L}_n} x \succeq_{\mathcal{L}_n} 0$ describe the same set $\{0\}$, however, the former is well-posed and the latter is not. Nevertheless, for brevity, we will refer to the well-posedness of a set $\mathcal{X}$, only if there no ambiguity in its description with conic inequalities.

A conic optimization problem writes, in standard form,

$$(P) \quad \min_{x} \quad c^T x \tag{5.7a}$$
$$s.t. \quad Ax = b, \tag{5.7b}$$
$$x \in \mathcal{K}, \tag{5.7c}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $\mathcal{K}$ is a cone. The strong/weak (in)feasibility and well-posedness of (P) refers to that of the system $(Ax = b, x \in \mathcal{K})$. The optimal value of (P) is $\text{opt}(P) = \inf \left\{ c^T x \;\middle|\; Ax = b, x \in \mathcal{K} \right\}$, and we say that (P) is *bounded* if $\text{opt}(P) \in \mathbb{R}$ and *solvable* if, in addition, there exists a feasible solution x^* such that $c^T x^* = \text{opt}(P)$. The dual of (P) is

$$(D) \quad \max_{y,s} \quad b^T y \tag{5.8a}$$
$$s.t. \quad A^T y + s = c, \tag{5.8b}$$
$$s \in \mathcal{K}^*, \tag{5.8c}$$

and $\text{opt}(D) = \sup \left\{ b^T y \;\middle|\; A^T y + s = c, s \in \mathcal{K}^* \right\}$. In particular, (D) is also a conic optimization problem.

Theorem 2 (Conic duality theorem)**.** *1. [Weak duality]* $\text{opt}(D) \leq \text{opt}(P)$.

2. [Strong duality] If (P) (resp. (D)) is strongly feasible and bounded, then (D) (resp. (P)) is solvable and $\text{opt}(P) = \text{opt}(D)$.

If both (P) and (D) are strongly feasible, then both are solvable with same optimal value.

Proof. See Theorem 1.4.4 in [125]. □

Conic duality extends the classical duality for linear programming, albeit with a number of edge cases that lead to practical difficulties. For instance, there may exist a positive duality gap even though both (P) and (D) are solvable, as illustrated in Example 1.

Example 1 (Example 8.6, [128])**.** *Consider the primal-dual pair*

$$
\begin{aligned}
(P) \quad \min_{x_1,x_2,x_3} \quad & x_3 & \qquad (D) \quad \max_{y_1,y_2} \quad & -y_2 \\
s.t. \quad & x_2 \geq x_1, & s.t. \quad & (y_1, -y_1, 1 - y_2) \in \mathcal{L}_3, \\
& x_3 \geq -1, & & y_1, y_2 \geq 0. \\
& (x_1, x_2, x_3) \in \mathcal{L}_3, & &
\end{aligned}
$$

Primal-feasible solutions are of the form $(x_1, x_1, 0)$*, while dual-feasible solutions are of the form* $(y_1, 1)$*. Thus,* $\text{opt}(P) = 0$ *and* $\text{opt}(D) = -1 < \text{opt}(P)$*.*

5.2.2 Valid inequalities

For $(\alpha, \beta) \in \mathbb{R}^n \times \mathbb{R}$, we say that $\alpha^T x \geq \beta$ is a *valid inequality* for $\mathcal{X} \subseteq \mathbb{R}^n$ if

$$\mathcal{X} \subseteq \left\{ x \in \mathbb{R}^n \;\middle|\; \alpha^T x \geq \beta \right\},$$

and a *supporting hyperplane* if, in addition, $\exists\, \tilde{x} \in \operatorname{cl}\operatorname{conv} \mathcal{X} : \alpha^T \tilde{x} = \beta$. The set of valid inequalities for $\mathcal{X}$ is denoted by $\mathcal{X}^\#$, i.e.,

$$\mathcal{X}^\# = \left\{ (\alpha, \beta) \in \mathbb{R}^n \times \mathbb{R} \;\middle|\; \forall x \in \mathcal{X},\ \alpha^T x \geq \beta \right\}. \tag{5.9}$$

Note that $\mathcal{X}^\#$ is a closed, convex set, and that

$$\left\{ x \in \mathbb{R}^n \;\middle|\; \forall (\alpha, \beta) \in \mathcal{X}^\#,\ \alpha^T x \geq \beta \right\} = \operatorname{cl}\operatorname{conv}(\mathcal{X}). \tag{5.10}$$

Furthermore, for $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^n$, we have

$$\begin{aligned}\mathcal{X} \subseteq \mathcal{Y} &\Rightarrow \mathcal{Y}^{\#} \subseteq \mathcal{X}^{\#},\\ (\mathcal{X} \cup \mathcal{Y})^{\#} &= \mathcal{X}^{\#} \cap \mathcal{Y}^{\#}.\end{aligned}$$

We now focus on the case where $\mathcal{X}$ is described by conic inequalities, and seek an algebraic description of $\mathcal{X}^{\#}$ using a finite number of conic inequalities. Note that, although $\mathcal{X}^{\#}$ is described by an infinite number of linear inequalities, as per Equation (5.9), this semi-infinite representation is not computationally tractable.

Theorem 3 (Conic theorem on alternatives)**.** *Consider the conic system*

$$Ax \succeq_{\mathcal{K}} b, \tag{5.11}$$

where $A \in \mathbb{R}^{m \times n}$ has full column rank and $\mathcal{K}$ is a proper cone.

1. *If there exists $y \in \mathbb{R}^m$ such that*

   $$A^T y = 0, b^T y > 0, y \in \mathcal{K}^*, \tag{5.12}$$

   then (5.11) *has no solution.*

2. *If* (5.12) *has no solution, then* (5.11) *is almost solvable, i.e., for any $\epsilon > 0$, there exists $\tilde{b} \in \mathbb{R}^m$ such that $\|b - \tilde{b}\|_2 < \epsilon$ and the system $Ax \succeq_{\mathcal{K}} \tilde{b}$ is solvable.*

3. (5.12) *is solvable if and only if* (5.11) *is not almost solvable.*

Proof. See Proposition 1.4.2 in [125]. □

Theorem 4 (Valid inequalities)**.** *Let $\mathcal{C} = \{x \mid Ax \succeq_{\mathcal{K}} b\}$ with A of full column rank, and define*

$$\mathcal{F} = \left\{(\alpha, \beta) \;\middle|\; \exists u \in \mathcal{K}^* : (\alpha = A^T u, \beta \leq b^T u)\right\}.$$

Then, $\operatorname{cl} \mathcal{F} \subseteq \mathcal{C}^{\#}$ and, in addition,

1. *if $\mathcal{C} \neq \emptyset$, then $\operatorname{cl} \mathcal{F} = \mathcal{C}^{\#}$;*

2. *if $\mathcal{C}$ is well-posed, then $\mathcal{F} = \mathcal{C}^{\#}$.*

Proof. The inclusion $\mathcal{F} \subseteq C^{\#}$ is immediate from the definition of $\mathcal{K}^*$, and it follows that $\mathrm{cl}(\mathcal{F}) \subseteq \mathrm{cl}(C^{\#}) = C^{\#}$. Case **2** is a direct consequence of conic strong duality.

We now prove **1**. Assume $\mathcal{C} \neq \emptyset$, let $(\alpha, \beta) \in \mathcal{C}^{\#}$, and consider the systems

$$A^T u = \alpha,\ b^T u \geq \beta,\ \ u \in \mathcal{K}^*; \tag{5.13}$$

$$Ax \succeq_{\mathcal{K}} tb,\ \alpha^T x < t\beta,\ \ t \geq 0. \tag{5.14}$$

By Theorem 3, either (5.13) is almost solvable, or (5.14) is solvable. Let us prove that the latter does not hold.

Let (x, t) be a solution to (5.14). On the one hand, if $t > 0$, letting $\bar{x} = t^{-1}x$, we have $A\bar{x} \succeq_{\mathcal{K}} b$, i.e., $\bar{x} \in \mathcal{C}$, but $\alpha^T \bar{x} < \beta$, which contradicts $(\alpha, \beta) \in \mathcal{C}^{\#}$. On the other hand, if $t = 0$, then we have $Ax \succeq_{\mathcal{K}} 0$ and $\alpha^T x < 0$. Thus, for $x_0 \in \mathcal{C}$ and $\tau \geq 0$, we have

$$A(x_0 + \tau x) \succeq_{\mathcal{K}} b,$$

i.e., $(x_0 + \tau x) \in \mathcal{C}$. Furthermore, we have $\alpha^T(x_0 + \tau x) < \beta$ for large enough τ, which also contradicts $(\alpha, \beta) \in \mathcal{C}^{\#}$. Therefore, (5.14) is not solvable and (5.13) is almost solvable.

Thus, for any $\epsilon > 0$, there exists α_ϵ, β_ϵ and $u_\epsilon \in \mathcal{K}^*$ such that and

$$\|\alpha - \alpha_\epsilon\|_2 \leq \epsilon,\ \ \|\beta - \beta_\epsilon\|_2 \leq \epsilon,$$

and

$$\alpha_\epsilon = A^T u_\epsilon,\ \ \beta_\epsilon \leq b^T u_\epsilon,$$

i.e., $(\alpha_\epsilon, \beta_\epsilon) \in \mathcal{F}$. Taking $\epsilon \to 0$, we obtain that $(\alpha, \beta) \in \mathrm{cl}\,\mathcal{F}$. □

We will refer to the multiplier $u \in \mathcal{K}^*$ in Theorem 4 as a (conic) *Farkas multiplier*, and we say that $\alpha^T x \geq \beta$ is obtained by Farkas aggregation if $\alpha = A^T u$ and $\beta \leq b^T u$ for some $u \in \mathcal{K}^*$.

Theorem 4 highlights a fundamental difference between the linear and nonlinear settings. In the linear case, $\mathcal{F}$ is polyhedral, thus, it is always closed and, if $\mathcal{C}$ is non-empty, then all valid inequalities for $\mathcal{C}$ can be obtained by Farkas aggregation. In the conic setting, however, this property may no longer hold, i.e., there may exist valid inequalities that cannot be represented through Farkas aggregation.

Example 2. *Let $\mathcal{C} = \{x \mid Ax \succeq_{\mathcal{K}} 0\}$ where*

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}, \ \mathcal{K} = \mathcal{L}_3,$$

i.e., $\mathcal{C} = \{(x, y) \in \mathbb{R}^2 \mid (x, y, x) \in \mathcal{L}_3\}$. While A has full column rank and $\mathcal{K}$ is proper, the system $Ax \succeq_{\mathcal{K}} 0$ is not well-posed. It then is easy to verify that $\mathcal{C} = \{(x, 0) \mid x \geq 0\}$, and that $y \geq 0$ is a valid inequality for $\mathcal{C}$.

However, for any $u \in \mathcal{K}^* = \mathcal{L}_3$, we have

$$A^T u = \begin{pmatrix} u_1 + u_3 \\ u_2 \end{pmatrix},$$

and $u_1 + u_3 > 0$ unless $u_2 = 0$. Therefore, the system $A^T u = (0, 1), u \in \mathcal{K}^*$ has no solution, i.e., the valid inequality $y \geq 0$ cannot be obtained by Farkas aggregation.

Nevertheless, for $t \geq 0$, let $u_t = (\sqrt{t^2 + 1}, -t, 1)$, which yields the valid inequality $(\sqrt{t^2 + 1} - t)x + y \geq 0$. Then, as $t \to +\infty$, the term $(\sqrt{t^2 + 1} - t)$ becomes negligible and the inequality becomes, in the limit, $y \geq 0$.

5.2.3 Disjunctive inequalities

We now consider disjunctive conic sets, i.e., sets of the form

$$\mathcal{D} = \left\{ x \in \mathbb{R}^n \;\middle|\; \bigvee_{h=1}^{H} D_h x \succeq_{\mathcal{Q}_h} d_h \right\} \tag{5.15}$$

$$= \bigcup_{h=1}^{H} \{x \in \mathbb{R}^n \mid D_h x \succeq_{\mathcal{Q}_h} d_h\}, \tag{5.16}$$

where $H \in \mathbb{Z}_+$ and, $\forall h$, $D_h \in \mathbb{R}^{m_h \times n}$ and $\mathcal{Q}_h$ is a proper cone. We refer to $\operatorname{conv} \mathcal{D}$ as the *disjunctive hull* and, for $(\alpha, \beta) \in \mathcal{D}^{\#}$, we say that $\alpha^T x \geq \beta$ is a *disjunctive inequality.*

We focus on disjunctive conic sets and, again, we seek a tractable algebraic characterization of valid inequalities for such sets. We begin by stating an extension to the conic setting of Balas' representation of the convex hull of a union of polyhedra [132].

Theorem 5 (Characterization of the convex hull)**.** *Let*

$$\mathcal{D} = \bigcup_{h=1}^{H} \underbrace{\{x \in \mathbb{R}^n \mid D_h x \succeq_{\mathcal{Q}_h} d_h\}}_{\mathcal{D}_h},$$

where, $\forall h$, $D_h \in \mathbb{R}^{m_h \times n}$ and $\mathcal{Q}_h$ is a proper cone, and let

$$\mathcal{S} = \left\{ x \in \mathbb{R}^n \;\middle|\; \exists (y_1, ..., y_H, z_1, ..., z_H) : \begin{array}{ll} \sum_h y_h = x, & \\ D_h y_h \succeq_{\mathcal{Q}_h} z_h d_h, & \forall h, \\ z_h \geq 0, & \forall h, \\ \sum_h z_h = 1, & \forall h, \\ y_h \in \mathbb{R}^n, & \forall h \end{array} \right\}.$$

Then $\operatorname{conv}(\mathcal{D}) \subseteq \mathcal{S}$ *and, in addition,*

1. *if $\forall h, \mathcal{D}_h \neq \emptyset$, then $\mathcal{S} \subseteq \operatorname{cl}\operatorname{conv}(\mathcal{D})$;*
2. *if $\forall h, \mathcal{D}_h = \mathcal{X}_h + W$, where $\mathcal{X}_1, ..., \mathcal{X}_H$ are non-empty closed convex sets and W is a closed convex set, then*

$$\operatorname{conv}(\mathcal{D}) = \mathcal{S} = \operatorname{cl}\operatorname{conv}(\mathcal{D}).$$

Proof. See Proposition 2.3.5 in [125]. □

Next, building on Farkas multipliers and the result of Theorem 4, we can extend Balas' characterization of valid disjunctive inequalities (Theorem 3.1 in [132]) to the conic setting.

Theorem 6 (Disjunctive inequalities)**.** *Let*

$$\mathcal{D} = \bigcup_{h=1}^{H} \underbrace{\{x \in \mathbb{R}^n \mid D_h x \succeq_{\mathcal{Q}_h} d_h\}}_{\mathcal{D}_h},$$

where, $\forall h$, $D_h \in \mathbb{R}^{m_h \times n}$ and $\mathcal{Q}_h$ is a proper cone, and

$$\mathcal{F} = \bigcap_{h=1}^{H} \underbrace{\left\{(\alpha, \beta) \in \mathbb{R}^n \times \mathbb{R} \;\middle|\; \exists u_h \in \mathcal{Q}_h^* : (\alpha = A^T u_h, \beta \leq b^T u_h)\right\}}_{\mathcal{F}_h}.$$

Then, $\mathcal{F} \subset \mathcal{D}^{\#}$ and, in addition,

1. *if $\forall h, \mathcal{D}_h \neq \emptyset$, then $\mathcal{D}^{\#} = \bigcap_h \operatorname{cl}\mathcal{F}_h$;*

2. if $\forall h, \mathcal{D}_h$ is well-posed and D_h has full column rank, then $\mathcal{F} = \mathcal{D}^{\#}$.

Proof. Immediate from Theorem 4 and the fact that $\mathcal{D}^{\#} = \bigcap_h D_h^{\#}$. $\square$

Example 3. *Let*

$$\mathcal{D} = \left\{(x, y) \in \mathbb{R}^2 \;\middle|\; (x, y, x) \in \mathcal{L}_3\right\} \cup \left\{(x, y) \in \mathbb{R}^2 \;\middle|\; (-x, y, x) \in \mathcal{L}_3\right\}.$$

Thus, $\mathcal{D} = \{(x, 0) | x \in \mathbb{R}\}$, and valid inequalities for $\mathcal{D}$ are of the form $\pm y \geq \pm\beta$ for $\beta \leq 0$. Building on Example 2, it follows that $\mathcal{F} = \{(0, 0)\} \times \mathbb{R}_-$. Therefore, only the trivial inequality $0 \geq -1$ can be represented using finite Farkas multipliers for each disjunctive term.

5.3 Cut separation

We consider an MI-CONIC problem of the form

$$\begin{aligned} (MICP) \quad \min_x \quad & c^T x && (5.17a)\\ s.t. \quad & Ax = b && (5.17b)\\ & x \in \mathcal{K}, && (5.17c)\\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}, && (5.17d) \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $p \leq n$, and $\mathcal{K}$ is a proper cone. The continuous relaxation of $(MICP)$, denoted by (CP), is given by (5.17a)-(5.17c). The feasible sets of $(MICP)$ and (CP) are denoted by $\mathcal{X}$ and by $\mathcal{C}$, respectively.

Let $\bar{x} \in \mathbb{R}^n$ be a point to separate. Since $\bar{x}$ is typically obtained from solving a relaxation of $(MICP)$, we will assume that $A\bar{x} = b$, i.e., all linear equality constraints are satisfied. In particular, we will not assume that $\bar{x}$ is conic-feasible, i.e., we may have $\bar{x} \notin \mathcal{K}$, for instance when an outer-approximation algorithm is used.

Consider the disjunctive set

$$\mathcal{D} = \bigcup_{h \in H} \mathcal{D}_h = \bigcup_{h \in H} \left\{ x \;\middle|\; \begin{array}{l} Ax = b, x \in \mathcal{K} \\ D_h x \succeq_{\mathcal{Q}_h} d_h \end{array} \right\}, \tag{5.18}$$

where each $\mathcal{Q}_h$ is a proper cone, and $\mathcal{D} \supseteq \mathcal{X}$. Valid inequalities for $\mathcal{D}$ are referred to as *disjunctive inequalities* or, equivalently, as *disjunctive cuts*. For $(\alpha, \beta) \in \mathcal{D}^{\#}$, the inequality $\alpha^T x \geq \beta$ is *trivial* if $(\alpha, \beta) \in \mathcal{C}^{\#}$, and *non-trivial* otherwise. Following the terminology of [130], $\mathcal{K}^*$ *cuts* are trivial inequalities of the form $u^T x \geq 0$ for $u \in \mathcal{K}^*$. Finally, a cut is *violated* if $\alpha^T\bar{x} < \beta$.

It is always possible, e.g., through the use of facial reduction techniques [162], to describe each $\mathcal{D}_h$ by a well-posed system of conic inequalities. In addition, one may assume, after manual inspection, that $\forall h, \mathcal{D}_h \neq \emptyset$. However, in a cutting-plane context, systematically performing such reductions and verifications can quickly become intractable. Therefore, unless stated otherwise, we make no assumption regarding the feasibility nor well-posedness of individual disjunctive terms; we will show in Section 5.4 how to address such shortcomings in a systematic way.

5.3.1 Separation problem

Consider the *cut-generating conic problem* (CGCP)

$$
\begin{aligned}
(CGCP) \quad \min_{\alpha,\beta,u,\lambda,v} \quad & \alpha^T\bar{x} - \beta && && (5.19a)\\
s.t. \quad & \alpha = A^T u_h + \lambda_h + D_h^T v_h, && \forall h, && (5.19b)\\
& \beta \leq b^T u_h + d_h^T v_h, && \forall h, && (5.19c)\\
& (u_h, \lambda_h, v_h) \in \mathbb{R}^m \times \mathcal{K}^* \times \mathcal{Q}_h^*, && \forall h, && (5.19d)
\end{aligned}
$$

which naturally extends Balas' CGLP to the conic setting, see also [156]. In particular, it is a conic programming problem, which can be solved by, e.g., an interior-point algorithm.

First, it follows from Theorem 6 that, if $(\alpha, \beta, u, \lambda, v)$ is feasible for (5.19), then $\alpha^T x \geq \beta$ is a disjunctive inequality. Under the stronger assumption of Case 2. in Theorem 6, every disjunctive inequality corresponds to a feasible solution of the CGCP. In the absence of such assumptions, however, the exact characterization of $\mathcal{D}^\#$ stated in Theorem 6 may not hold. For instance, there may exist disjunctive inequalities that do not correspond to any feasible solution of the CGCP; as illustrated by Example 3, it may even be that all feasible solutions of the CGCP correspond to trivial inequalities of the form $0 \geq \beta$ for some $\beta \leq 0$.

Second, the feasible set of the CGCP is an unbounded cone, which contains the origin. Thus, the CGCP is either unbounded or bounded with objective value zero. In the former case, any unbounded ray yields a violated cut, while in the latter, no violated cut is obtained. Note that unbounded problems can lead to numerical issues for some interior-point algorithms. Therefore, it is common practice to add a normalization condition to the CGCP, whose role is further investigated in Section 5.4.

Third, the CGCP is strongly feasible. Indeed, let $\bar{u}_h = 0, \bar{v}_h \in \text{int } \mathcal{Q}_h^*, \forall h$. Then, since $\mathcal{K}^*$

has non-empty interior, there exists $\bar{\alpha}$ such that

$$\bar{\alpha} \succ_{\mathcal{K}^*} D_h^T \bar{v}_h, \forall h.$$

Finally, letting $\bar{\lambda}_h = \bar{\alpha} - D_h^T \bar{v}_h \in \operatorname{int} \mathcal{K}^*$ and $\bar{\beta} \leq d_h^T \bar{v}_h, \forall h$, it follows that $(\bar{\alpha}, \bar{\beta}, \bar{u}, \bar{\lambda}, \bar{x})$ is strongly feasible for the CGCP.

Fourth, let $(\alpha, \beta, u, \lambda, v)$ be a feasible solution of the CGCP, and let

$$(\tilde{\alpha}, \tilde{\beta}, \tilde{u}, \tilde{\lambda}, \tilde{v}) = (\alpha - A^T u_0, \beta - b^T u_0, u - u_0, \lambda, v), \tag{5.20}$$

where $u_0 \in \mathbb{R}^m$. It is immediate to see that $(\tilde{\alpha}, \tilde{\beta}, \tilde{u}, \tilde{\lambda}, \tilde{v})$ is also feasible for the CGCP, with identical objective value since $A\bar{x} = b$. Therefore, without loss of generality, one of the u_h can be arbitrarily set to zero in the formulation of the CGCP, thereby reducing its size. Furthermore, since $\mathcal{C} \subseteq \{x \mid Ax = b\}$, it follows that

$$\mathcal{C} \cap \left\{ x \;\middle|\; \alpha^T x \geq \beta \right\} = \mathcal{C} \cap \left\{ x \;\middle|\; \tilde{\alpha}^T x \geq \tilde{\beta} \right\}, \tag{5.21}$$

i.e., the two inequalities are equivalent in the sense that both cut off the same portion of the continuous relaxation.

5.3.2 Membership problem

The dual problem of the CGCP is the *membership conic problem* (MCP)

$$\begin{aligned}
(MCP) \quad \max_{y,z} \quad & 0 && && (5.22a)\\
s.t. \quad & \sum_h y_h = \bar{x}, && && (5.22b)\\
& \sum_h z_h = 1, && && (5.22c)\\
& Ay_h = z_h b, && \forall h, && (5.22d)\\
& D_h y_h \succeq_{\mathcal{Q}_h} z_h d_h, && \forall h, && (5.22e)\\
& (y_h, z_h) \in \mathcal{K} \times \mathbb{R}_+, && \forall h, && (5.22f)
\end{aligned}$$

which extends Bonami's membership LP [163] to the conic setting.

A geometrical interpretation of the MCP is provided by Theorem 5. If all disjunctive terms are non-empty and have identical recession cones, then (5.22) is feasible if and only if $\bar{x} \in \operatorname{conv}(\mathcal{D})$. In the general case, however, the exact characterization of $\operatorname{conv} \mathcal{D}$ given by Theorem

5 may no longer hold, and we can only state that, if $\bar{x} \in \operatorname{conv} \mathcal{D}$, then the MCP is feasible.

By weak duality, if the MCP is feasible, then the objective value of the CGCP is bounded below. If, in addition, the MCP is strongly feasible, then both the MCP and the CGCP are solvable with identical objective values.

5.4 The roles of normalization

This section focuses on the roles of the normalization condition in the CGCP. On the one hand, through the lens of conic duality for the CGCP-MCP pair, we investigate the impact of the normalization on the solvability of CGCP. On the other hand, by characterizing optimal solutions of the normalized CGCP, we assess the theoretical properties of the corresponding cuts.

The following normalization conditions are considered:

(i) the α normalization: $\|\alpha\| \leq 1$,

(ii) the *polar* normalization: $\gamma^T \alpha \leq 1$,

(iii) the *standard* normalization: $\sum_h |\lambda_h|_\rho + |v_h|_{\sigma_h} \leq 1$,

(iv) the *trivial* normalization: $\sum_h |v_h|_{\sigma_h} \leq 1$,

(v) the *uniform* normalization: $\sum_h |\lambda_h|_\rho \leq 1$,

where $\gamma \in \mathbb{R}^n$, $\rho \in \operatorname{int} \mathcal{K}$ and $\sigma_h \in \operatorname{int} \mathcal{Q}_h$.

Throughout this section, the strengths and shortcomings of each normalization are illustrated in a simple setting, described in Example 4 below.

Example 4. *For $R > 0$, consider the MICP*

$$\min_x \quad -x_1 - x_2 \tag{5.23}$$
$$s.t. \quad x_0 = R, \tag{5.24}$$
$$x \in \mathcal{L}_3, \tag{5.25}$$
$$x_1, x_2 \in \mathbb{Z}, \tag{5.26}$$

and the split disjunction $(x_1 \leq 0) \vee (x_1 \geq 1)$. Thus, we have

$$\mathcal{D}_1 = \left\{ x \in \mathbb{R}^3 \;\middle|\; x \in \mathcal{L}_3, x_0 = R, x_1 \leq 0 \right\},$$
$$\mathcal{D}_2 = \left\{ x \in \mathbb{R}^3 \;\middle|\; x \in \mathcal{L}_3, x_0 = R, x_1 \geq 1 \right\}.$$

We consider the following three cases:

(a) $R > 1$: $\mathcal{D}_1$ and $\mathcal{D}_2$ are both strongly feasible;

(b) $R = 1$: $\mathcal{D}_1$ is strongly feasible and $\mathcal{D}_2$ is weakly feasible;

(c) $R < 1$: $\mathcal{D}_1$ is strongly feasible and $\mathcal{D}_2 = \emptyset$.

Each of these settings is illustrated in Figure 5.1. Finally, unless specified otherwise, $\bar{x}$ is the solution of the continuous relaxation, i.e., $\bar{x} = (R, \frac{R}{\sqrt{2}}, \frac{R}{\sqrt{2}})$. All CGCPs are solved as conic problems using Mosek 9.2 with default parameters.

5.4.1 Alpha normalization

A straightforward way of bounding the CGCP is to restrict the magnitude of α. This approach was considered in previous work on MI-CONV [144, 145, 148], wherein authors considered restricting the ℓ_1, ℓ_∞ or ℓ_2 norm of α.

For a given norm $\|\cdot\|$, the CGCP then writes

$$\min_{\alpha,\beta,\lambda,u,v} \quad \alpha^T \bar{x} - \beta \tag{5.27a}$$

$$\text{s.t.} \quad \alpha = A^T u_h + \lambda_h + D_h^T v_h, \qquad \forall h, \tag{5.27b}$$

$$\beta \leq b^T u_h + d_h^T v_h, \qquad \forall h, \tag{5.27c}$$

$$(u_h, \lambda_h, v_h) \in \mathbb{R}^m \times \mathcal{K}^* \times \mathcal{Q}_h^*, \qquad \forall h, \tag{5.27d}$$

$$\|\alpha\|_* \leq 1, \tag{5.27e}$$

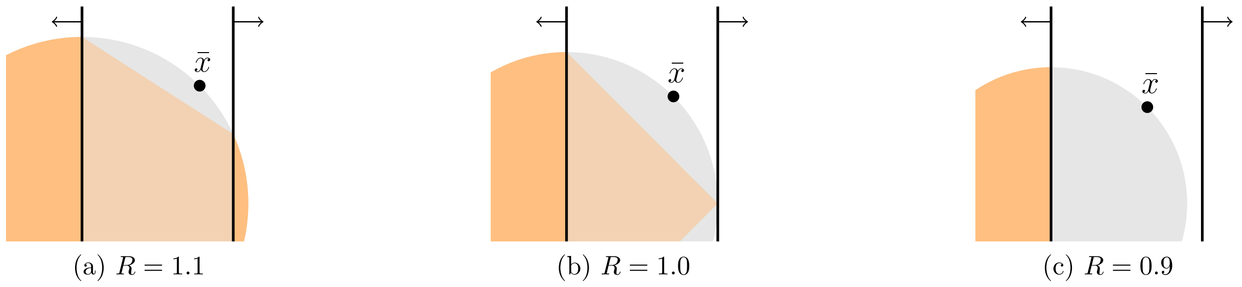


Figure 5.1 The three settings from Example 4, projected onto the $x_0 = 1$ space. The domain of the continuous relaxation is in gray, the split hull in orange, and the split disjunction is indicated in black.

and the corresponding MCP, up to a change of sign in the objective value, is

$$\min_{x,y,z} \quad \|\bar{x} - x\| \tag{5.28a}$$

$$s.t. \quad \sum_h y_h = x, \tag{5.28b}$$

$$\sum_h z_h = 1, \tag{5.28c}$$

$$Ay_h = z_h b, \quad \forall h, \tag{5.28d}$$

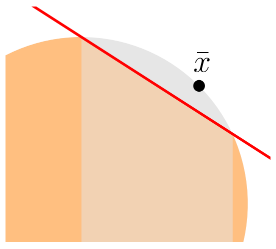
$$D_h y_h \succeq_{\mathcal{Q}_h} z_h d_h, \quad \forall h, \tag{5.28e}$$

$$(y_h, z_h) \in \mathcal{K} \times \mathbb{R}_+, \quad \forall h. \tag{5.28f}$$

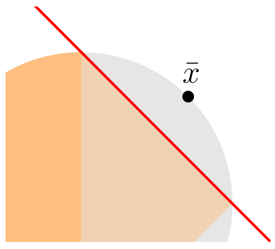
Geometrically, the CGCP (5.27) looks for a deepest cut, i.e., one that maximizes the distance from $\bar{x}$ to the hyperplane $\alpha^T x = \beta$, as measured by $\|\cdot\|$. Correspondingly, the MCP (5.28) computes a projection of $\bar{x}$ onto the set defined by (5.28b)-(5.28f), with respect to $\|\cdot\|$. It is trivially feasible if at least one of the disjunctive terms is non-empty, which is always the case if $\mathcal{X} \neq \emptyset$. This ensures that the CGCP (5.27) is never unbounded. If, in addition, each disjunctive term is strongly feasible, then the MCP (5.28) is strongly feasible and both the MCP and the CGCP are solvable.

Split cuts obtained with the α normalization in the context of Example 4 are illustrated in Figure 5.2; these results are obtained with the (self-dual) ℓ_2 norm in (5.27e). Furthermore, some statistics regarding the resolution of the CGCP are reported in Table 5.1, namely: the number of interior-point iterations (Iter), and the magnitude of α, u, λ, v in the obtained CGCP solution.

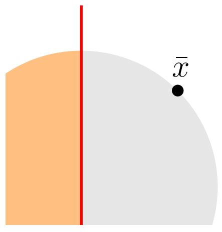
In Example 4(a), the MCP is strongly feasible, no numerical trouble is encountered, and the obtained cut is a supporting hyperplane of the split hull. On the other hand, numerical issues are encountered in Example 4(b). Indeed, as reported in Table 5.1, Mosek terminates due



(a)



(b)



(c)

Figure 5.2 Split cuts (in red) obtained with the α normalization.

Table 5.1 CGCP statistics for Example 4 and α normalization

	Iter	$\|\alpha\|$	$\|u_1\|$	$\|u_2\|$	$\|\lambda_1\|$	$\|\lambda_2\|$	$\|v_1\|$	$\|v_2\|$
(a)	8	1.0	0.8	2.0	1.2	2.9	0.5	1.3
(b)	67*	1.0	0.7	10958.8	1.0	15498.0	0.7	10958.1
(c)	8	1.0	0.0	13.7	0.0	19.3	1.0	12.6

*: slow progress

to slow progress after 67 iterations, albeit with a feasible solution, for which the magnitude of u_2, λ_2, v_2 is very large. The corresponding cut is displayed in Figure 5.2b. Finally, in Example 4(c), although $\mathcal{D}_2 = \emptyset$, the CGCP is solved without issue, thereby showing that it is not necessary for the MCP to be strongly feasible for the CGCP to be solvable.

Example 4(b) illustrates the numerical challenges that may arise when the MCP is not strongly feasible. Indeed, in that case, the deepest cut writes $x_1 + x_2 \leq 1$, up to a positive scaling factor. However, although $x_1+x_2 \leq 1$ is a valid inequality for $\mathcal{D}_2$, it is straightforward to see that it cannot be represented using finite Farkas multipliers: this corresponds to case **1** in Theorem 4. Thus, the CGCP has no optimal solution, and there exists a (diverging) sequence of feasible solutions whose objective value becomes arbitrary close to opt($CGCP$), corresponding to a sequence of valid inequalities that, in the limit, become equivalent to $x_1 + x_2 \leq 1$. Hence, the solver eventually runs into slow progress while the magnitude of u_2, λ_2, v_2 becomes large, as observed in Table 5.1.

Interestingly, all cuts displayed in Figure 5.2 are supporting hyperplanes of the split hull. A slightly more general result is stated in Theorem 7.

Theorem 7. *Assume that the CGCP* (5.27) *and the MCP* (5.28) *are solvable, and let* $(\alpha, \beta, u, \lambda, v)$ *and* (x, y, z) *be corresponding optimal solutions. Then,*

$$\alpha^T x = \beta.$$

Proof. Let δ be the optimal value of the CGCP, i.e., $\delta = \alpha^T \bar{x} - \beta$. Since the CGCP is strongly feasible and bounded, by Theorem 2, strong duality holds. Thus, we have $\delta = -\|\bar{x} - x\|$. The case $\delta = 0$ is trivial, so we assume $\delta < 0$ and, thus, $x \neq \bar{x}$.

Let $w = |\delta|^{-1}(x - \bar{x})$, i.e., $x = \bar{x} + |\delta| w$ and $\|w\| = 1$; in particular, we have $1 \geq \|\alpha\|_* \geq \alpha^T w$.

It follows that

$$\begin{aligned}\alpha^T x - \beta &= \alpha^T \bar{x} + |\delta| \alpha^T w - \beta \\ &= \delta + |\delta| \alpha^T w \\ &= \delta(1 - \alpha^T w) \\ &\leq 0.\end{aligned}$$

Thus, $\alpha^T x \leq \beta$.

Next, we have

$$\begin{aligned}\alpha^T x &= \sum_h \alpha^T y_h \\ &= \sum_h u_h^T A y_h + \lambda_h^T y_h + v_h^T D_h y_h \\ &\geq \sum_h u_h^T (z_h b) + v_h^T (z_h d_h) \\ &\geq \sum_h z_h \beta \\ &= \beta,\end{aligned}$$

which conclude the proof. □

Therefore, if all disjunctive terms are non-empty, then, by Theorem 5, $x \in \operatorname{cl} \operatorname{conv} \mathcal{D}$, and the obtained inequality $\alpha^T x \geq \beta$ is indeed a supporting hyperplane of the disjunctive hull.

5.4.2 Polar normalization

In the MILP setting, Balas and Perregard [142,164] first suggest normalizing the CGLP with a single hyperplane of the form $\alpha^T \gamma = 1$. Doing so ensures that, if the CGLP is feasible and bounded, then there exists an optimal solution for which (α, β) is an extreme ray of $(\operatorname{cl} \operatorname{conv} \mathcal{D})^{\#}$. This approach was then followed in [138, 139], and more recently in [140].

For $\gamma \in \mathbb{R}^n$, the CGCP writes

$$\begin{aligned}
\min_{\alpha,\beta,u,v,\lambda} \quad & \alpha^T \bar{x} - \beta && && (5.29a)\\
s.t. \quad & \alpha = A^T u_h + \lambda_h + D_h^T v_h, && \forall h, && (5.29b)\\
& \beta \leq b^T u_h + d_h^T v_h, && \forall h, && (5.29c)\\
& (u_h, \lambda_h, v_h) \in \mathbb{R}^m \times \mathcal{K}^* \times \mathcal{Q}_h^*, && \forall h, && (5.29d)\\
& \alpha^T \gamma \leq 1, && && (5.29e)
\end{aligned}$$

and the MCP is given by

$$\begin{aligned}
\min_{y,z,\eta} \quad & \eta && && (5.30a)\\
s.t. \quad & \sum_h y_h = \bar{x} + \eta\gamma, && && (5.30b)\\
& \sum_h z_h = 1, && && (5.30c)\\
& A y_h = z_h b, && \forall h, && (5.30d)\\
& D_h y_h \succeq_{\mathcal{Q}_h} z_h d_h, && \forall h, && (5.30e)\\
& (y_h, z_h) \in \mathcal{K} \times \mathbb{R}_+, && \forall h, && (5.30f)\\
& \eta \geq 0. && && (5.30g)
\end{aligned}$$

It follows from Theorem 5 that, if there exists $\eta \geq 0$ such that $(\bar{x} + \eta\gamma) \in \operatorname{conv} \mathcal{D}$, then the MCP (5.30) is feasible. This is always the case if $\gamma = x^* - \bar{x}$, for some $x^* \in \operatorname{conv} \mathcal{D}$. If, in addition, each individual disjunction is strongly feasible and x^* is obtained as a convex combination of strongly feasible points, then the MCP (5.30) is strongly feasible.

Split cuts obtained for Example 4 with the polar normalization are illustrated in Figure 5.3, and the corresponding CGCP statistics are reported in Table 5.2. In each case, we set $\gamma = x^* - \bar{x}$, where $x^* = (R, 0, 0)$.

In all three cases, the obtained cut is identical to the one obtained with the α normalization, although this is not the case in general. Furthermore, as reported in Table 5.2, numerical issues are also encountered for Example 4(b), for the same reasons as for the α normalization: the CGCP is not solvable, and Mosek terminates with slow progress while the magnitude of u_2, λ_2, v_2 diverges.

Similar to the α-normalization, if all disjunctive terms are non-empty, then cuts obtained with the polar normalization are also supporting hyperplanes of the disjunctive hull, as expressed

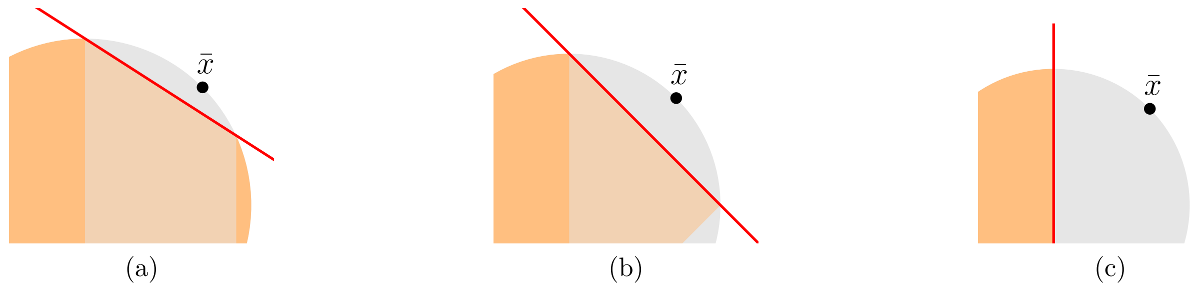


Figure 5.3 Split cuts (in red) obtained with the polar normalization.

Table 5.2 CGCP statistics for Example 4 and polar normalization

	Iter	$\|\alpha\|$	$\|u_1\|$	$\|u_2\|$	$\|\lambda_1\|$	$\|\lambda_2\|$	$\|v_1\|$	$\|v_2\|$
(a)	8	1.2	0.0	1.1	1.1	2.7	0.5	1.2
(b)	66*	1.2	0.0	9912.6	1.0	14019.5	0.7	9912.6
(c)	8	1.6	0.0	20.5	0.0	28.9	1.6	18.8

*: slow progress

by Theorem 8.

Theorem 8. *Assume that the CGCP* (5.29) *and the MCP* (5.30) *are solvable, and let* $(\alpha, \beta, u, \lambda, v)$ *and* (η, y, z) *be corresponding optimal solutions. Then,*

$$\alpha^T(\bar{x} + \eta\gamma) = \beta.$$

Proof. By conic strong duality, we have $\alpha^T\bar{x} - \beta = -\eta$. If the optimal value of the CGCP is 0, then the result is trivial. Similarly, if $\alpha^T\gamma \leq 0$, then the optimal value of the CGCP must be 0 and the result is trivial.

We now assume that $\alpha^T\gamma > 0$ and opt$(CGCP) < 0$. Since $(\alpha, \beta, u, \lambda, v)$ is optimal for the CGCP, we must have $\alpha^T\gamma = 1$. Then,

$$\begin{aligned} \alpha^T\bar{x} - \beta &= -\eta \\ &= -(\alpha^T\gamma)\eta, \end{aligned}$$

i.e., $\alpha^T(\bar{x} + \eta\gamma) = \beta$. □

5.4.3 Standard normalization

One of the most common normalizations in the MILP setting is the so-called *standard normalization* [132, 136, 141]. Here, we introduce its conic generalization

$$\sum_h |\lambda_h|_\rho + |v_h|_{\sigma_h} \leq 1, \tag{5.31}$$

where $\rho \in \operatorname{int} \mathcal{K}$ and $\sigma_h \in \operatorname{int} \mathcal{Q}_h$. When all cones $\mathcal{K}, \mathcal{Q}_1, ..., \mathcal{Q}_H$ are non-negative orthants, (5.31) indeed generalizes both the standard normalization for MILP and its *Euclidean normalization* variant [136], by setting ρ and σ appropriately.

The separation problem then writes

$$\begin{aligned}
\min_{\alpha,\beta,u,v,\lambda} \quad & \alpha^T \bar{x} - \beta && && \text{(5.32a)}\\
s.t. \quad & \alpha = A^T u_h + \lambda_h + D_h^T v_h, && \forall h, && \text{(5.32b)}\\
& \beta \leq b^T u_h + d_h^T v_h, && \forall h, && \text{(5.32c)}\\
& (u_h, \lambda_h, v_h) \in \mathbb{R}^m \times \mathcal{K}^* \times \mathcal{Q}_h^*, && \forall h, && \text{(5.32d)}\\
& \sum_h |\lambda_h|_\rho + |v_h|_{\sigma_h} \leq 1. && && \text{(5.32e)}
\end{aligned}$$

Note that (5.32e) consists of a single linear inequality in the λ, v space, and that it explicitly bounds their magnitude.

Up to a change of sign in the objective value, the MCP is

$$\begin{aligned}
\min_{y,z,\eta} \quad & \eta && && \text{(5.33a)}\\
s.t. \quad & \sum_h y_h = \bar{x}, && && \text{(5.33b)}\\
& \sum_h z_h = 1, && && \text{(5.33c)}\\
& A y_h = z_h b, && \forall h, && \text{(5.33d)}\\
& D_h y_h + \eta \sigma_h \succeq_{\mathcal{Q}_h} z_h d_h, && \forall h, && \text{(5.33e)}\\
& (y_h + \eta \rho, z_h) \in \mathcal{K} \times \mathbb{R}_+, && \forall h, && \text{(5.33f)}\\
& \eta \geq 0. && && \text{(5.33g)}
\end{aligned}$$

The dual counterpart of the standard normalization corresponds to penalizing the violation of the original conic constraints (5.22f) and (5.22e), through the artificial slack variable η in (5.33f) and (5.33e). As shown in Theorem 9, this ensures that the MCP is strongly feasible.

Theorem 9. *The CGCP* (5.32) *and the MCP* (5.33) *are strongly feasible.*

Proof. It was shown in Section 5.3.1 that the un-normalized CGCP (5.19) admits a strongly feasible solution. Scaling this point appropriately yields a strongly feasible solution for the CGCP (5.32).

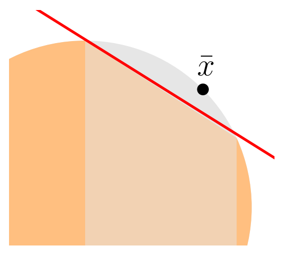
We now show that the MCP (5.36d) is strongly feasible. Let $y_h = \frac{1}{H}\bar{x}$ and $z_h = \frac{1}{H}, \forall h$. Then, since $\mathcal{K}, \mathcal{Q}_1, ..., \mathcal{Q}_H$ are proper cones, there exists $\eta_h \geq 0$ and $\tau_h \geq 0$ such that

$$\begin{aligned} &\forall h,\ y_h + \eta_h \rho \succ_{\mathcal{K}} 0, \\ &\forall h,\ D_h y_h + \tau_h \sigma_h \succ_{\mathcal{Q}_h} d_h. \end{aligned}$$

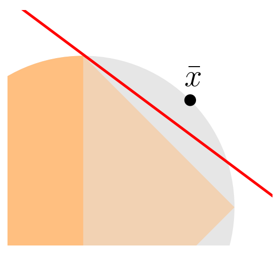
Letting $\eta = \max(\eta_1, ..., \eta_H, \tau_1, ..., \tau_H)$ then yields a strongly feasible point (y, z, η), which concludes the proof. $\square$

A direct consequence of Theorem 9 is that both the CGCP (5.32) and the MCP (5.33) are solvable, and that conic strong duality holds. Importantly, aside from the assumption $A\bar{x} = b$, this result does not depend on $\bar{x}$, nor on the well-posedness of individual disjunctions.

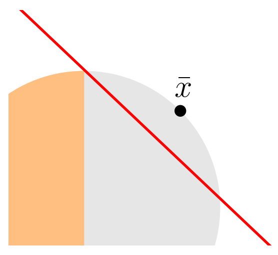
Split cuts obtained with the standard normalization are illustrated in Figure 5.4, and the corresponding CGCP statistics are reported in Table 5.3. On the one hand, Table 5.3 illustrate the good numerical behavior of the CGCP (5.32), a direct consequence of having enforced strong feasibility of the MCP. On the other hand, the obtained cuts are not as strong as the ones obtained with the α or polar normalizations. Indeed, in general, the cut obtained with the standard normalization may not be a supported hyperplane of the disjunctive hull.



(a)



(b)



(c)

Figure 5.4 Split cuts (in red) obtained with the standard normalization.

Table 5.3 CGCP statistics for Example 4 and standard normalization

	Iter	$\|\alpha\|$	$\|u_1\|$	$\|u_2\|$	$\|\lambda_1\|$	$\|\lambda_2\|$	$\|v_1\|$	$\|v_2\|$
(a)	8	0.4	0.0	0.2	0.3	0.6	0.1	0.2
(b)	8	0.3	0.0	0.2	0.3	0.6	0.1	0.2
(c)	7	0.3	0.0	0.3	0.3	0.6	0.1	0.2

5.4.4 Trivial normalization

The trivial normalization is obtained by setting $\rho = 0$ in the standard normalization. The separation problem then writes

$$\begin{aligned}
\min_{\alpha,\beta,u,v,\lambda} \quad & \alpha^T \bar{x} - \beta && && \text{(5.34a)}\\
s.t. \quad & \alpha = A^T u_h + \lambda_h + D_h^T v_h, && \forall h, && \text{(5.34b)}\\
& \beta \leq b^T u_h + d_h^T v_h, && \forall h, && \text{(5.34c)}\\
& (u_h, \lambda_h, v_h) \in \mathbb{R}^m \times \mathcal{K}^* \times \mathcal{Q}_h^*, && \forall h, && \text{(5.34d)}\\
& \sum_h |v_h|_{\sigma_h} \leq 1. && && \text{(5.34e)}
\end{aligned}$$

Note that, for a split disjunction $(-\pi^T x \geq -\pi_0) \vee (\pi^T x \geq \pi_0 + 1)$ and $\sigma_h = 1$, (5.34e) reduces to

$$v_1 + v_2 \leq 1, \tag{5.35}$$

which is the so-called trivial normalization [136] in the MILP setting. In particular, Gomory Mixed-Integer cuts correspond to optimal solutions of the CGLP with trivial normalization.

Up to a change of sign in the objective value, the MCP writes

$$\begin{aligned}
\min_{y,z,\eta} \quad & \eta && && \text{(5.36a)}\\
s.t. \quad & \sum_h y_h = \bar{x}, && && \text{(5.36b)}\\
& \sum_h z_h = 1, && && \text{(5.36c)}\\
& Ay_h = z_h b, && \forall h, && \text{(5.36d)}\\
& (y_h, z_h) \in \mathcal{K} \times \mathbb{R}_+, && \forall h, && \text{(5.36e)}\\
& D_h y_h + \eta \sigma_h \succeq_{\mathcal{Q}_h} z_h d_h, && \forall h, && \text{(5.36f)}\\
& \eta \geq 0. && && \text{(5.36g)}
\end{aligned}$$

Theorem 10. *The MCP* (5.36) *is*

1. *strongly infeasible if and only if $\bar{x}$ is infeasible for (CP);*

2. *strongly feasible if and only if $\bar{x}$ is strongly feasible for (CP);*

3. *weakly feasible if and only if $\bar{x}$ is weakly feasible for (CP).*

Proof. Since $\bar{x}$ is either infeasible, strongly feasible or weakly feasible for (CP), and that these are mutually exclusive alternatives, it suffices to prove that the above conditions are sufficient; necessity follows immediately by contraposition. For simplicity, we assume that $\mathcal{K}$ is an irreducible, non-polyhedral cone. The general case is treated similarly.

1. Assume $\bar{x} \notin \mathcal{C}$, i.e., $\bar{x} \notin \mathcal{K}$ since we assumed that $A\bar{x} = b$. Consequently, there exists $s \in \mathcal{K}^*$ such that $s^T \bar{x} < 0$ and, since $\mathcal{K}$ is a proper cone, we can assume without loss of generality that $s \in \operatorname{int} \mathcal{K}^*$. Thus, setting

$$\begin{aligned}
(\alpha, \beta) &= (s, 0) \\
(u_h, \lambda_h, v_h) &= (0, s, 0), && \forall h,
\end{aligned}$$

yields a strongly feasible solution of the CGCP (5.34), which is also an unbounded ray. Therefore, the MCP is strongly infeasible.

2, 3. We first show that, if $\bar{x}$ is feasible for (CP), then the MCP is feasible. Let

$$\begin{aligned} \tilde{y}_h &= H^{-1}\bar{x}, && \forall h,\\ \tilde{z}_h &= H^{-1}, && \forall h. \end{aligned}$$

It is then immediate that $A\tilde{y}_h = \tilde{z}_h b$, $\tilde{y}_h \in \mathcal{K}$, and $\sum_h \tilde{z}_h = 1$. Then, since $\sigma_h \in \operatorname{int} \mathcal{Q}_h$, there exists $\eta_h \geq 0$ such that

$$D_h y_h + \eta_h \sigma_h \succ_{\mathcal{Q}_h} z_h d_h.$$

Letting $\tilde{\eta} = \max_h\{\eta_h\}$, it follows that $(\tilde{y}, \tilde{z}, \tilde{\eta})$ is feasible for the MCP.

If $\bar{x} \in \operatorname{int} \mathcal{K}$, then we also have $\tilde{y}_h \in \operatorname{int} \mathcal{K}$, and thus $(\tilde{y}, \tilde{z}, \tilde{\eta})$ is strongly feasible for the MCP, which proves **2.** Reciprocally, let (y, z, η) be a strongly feasible solution of MCP. In particular, we have $y_h \in \operatorname{int} \mathcal{K}$. Then,

$$\bar{x} = \sum_h y_h \in \operatorname{int} \mathcal{K},$$

i.e., $\bar{x}$ is strongly feasible for (CP), thereby proving **3.** by contraposition. □

Theorem 10 motivates the following remarks. First, case **1.** typically arises in the context of outer-approximation algorithms, wherein fractional points generally violate nonlinear conic constraints. Then, the CGCP (5.34) is unbounded, and the normalization (5.34e) imposes $|v_h|_{\sigma_h} = 0, \forall h$ in any unbounded ray, i.e., $v_h = 0$ and the obtained cut is always a trivial inequality. Thus, the trivial normalization is not suited for use within outer approximation-based algorithms.

Second, conic-infeasible points are not encountered in nonlinear branch-and-bound algorithms. However, solving (CP) yields a fractional point $\bar{x} \in \partial \mathcal{C}$ that is weakly feasible, unless all nonlinear conic constraints are inactive at the optimum. Although the current fractional point $\bar{x}$ may become strongly feasible after several rounds of cuts, or deeper in the branch-and-bound tree, our experience is that case **2.** rarely occurs in practice.

Third, case **3.** corresponds to the setting of Example 4. Split cuts obtained with the trivial normalization are displayed in Figure 5.5, and the corresponding CGCP statistics are reported in Table 5.4. Remarkably, all three cuts appear to be $\mathcal{K}^*$ cuts, while numerical issues, namely, slow convergence, are systematically encountered. Here, the MCP (5.36) is weakly feasible and the CGCP (5.34) is bounded but not solvable. Thus, there exists a diverging sequence of close-to-optimal solution, thereby causing slow convergence. In addition, since (5.34e) bounds the value of v_1, v_2, the iterates become equivalent to a $\mathcal{K}^*$ cut as the magnitude of λ increases, which explains the cuts obtained in Figure 5.5.

Note that the CGCP (5.34) may be solvable even though $\bar{x}$ is weakly feasible. However, our experience suggests that this rarely happens, and that most cases are similar to Example 4, leading to numerical issues and weak cuts.

5.4.5 Uniform normalization

The uniform normalization is obtained by setting $\sigma_h = 0$ in the standard normalization.

The CGCP then writes

$$\min_{\alpha,\beta,u,v,\lambda} \quad \alpha^T\bar{x} - \beta \tag{5.37a}$$

$$s.t. \quad \alpha = A^T u_h + \lambda_h + D_h^T v_h, \qquad \forall h, \tag{5.37b}$$

$$\beta \leq b^T u_h + d_h^T v_h, \qquad \forall h, \tag{5.37c}$$

$$(u_h, \lambda_h, v_h) \in \mathbb{R}^m \times \mathcal{K}^* \times \mathcal{Q}_h^*, \qquad \forall h, \tag{5.37d}$$

$$\sum_h |\lambda_h|_\rho \leq 1, \tag{5.37e}$$

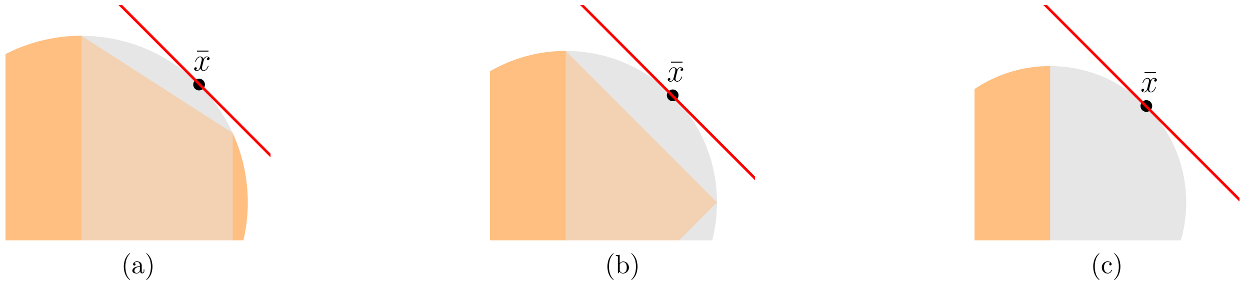


Figure 5.5 Split cuts (in red) obtained with the trivial normalization.

Table 5.4 CGCP statistics for Example 4 and trivial normalization

	Iter	$\|\alpha\|$	$\|u_1\|$	$\|u_2\|$	$\|\lambda_1\|$	$\|\lambda_2\|$	$\|v_1\|$	$\|v_2\|$
(a)	37*	6189.7	0.0	0.7	6189.6	6190.6	0.2	0.8
(b)	49*	6814.2	0.0	0.7	6814.0	6815.0	0.3	0.7
(c)	72*	6364.4	0.0	0.7	6364.2	6365.2	0.4	0.6

*: slow progress

and, up to a change of sign in the objective, the MCP is

$$\min_{y,z,\eta} \quad \eta \tag{5.38a}$$

$$s.t. \quad \sum_h y_h = \bar{x}, \tag{5.38b}$$

$$\sum_h z_h = 1, \tag{5.38c}$$

$$A y_h = z_h b, \quad \forall h, \tag{5.38d}$$

$$(y_h + \eta\rho, z_h) \in \mathcal{K} \times \mathbb{R}_+, \quad \forall h, \tag{5.38e}$$

$$D_h y_h \succeq_{\mathcal{Q}_h} z_h d_h, \quad \forall h, \tag{5.38f}$$

$$\eta \geq 0. \tag{5.38g}$$

As illustrated by Example 5, in general, the MCP (5.38) may not be feasible.

Example 5. *Let*

$$\mathcal{C} = \left\{ (x_1, x_2) \in \mathbb{R}^2_+ \mid x_1 + x_2 = 1 \right\},$$

and consider the disjunction

$$\{x_1 \geq 0, -x_1 \geq 0\} \vee \{x_1 \geq 1, -x_1 \geq -1\}.$$

Constraints (5.38d) *and* (5.38f) *first yield*

$$y_1 = \begin{pmatrix} 0 \\ z_1 \end{pmatrix}, \quad y_2 = \begin{pmatrix} z_2 \\ 1 - z_2 \end{pmatrix},$$

which, combined with (5.38c) *and* (5.38b), *yields* $\bar{x} = (1 - z_1, z_1)$ *for* $0 \leq z_1 \leq 1$. *Therefore, if* $\bar{x} = (-1, 2)$, *then the MCP* (5.38) *is infeasible.*

Nevertheless, the following results demonstrate that, for certain classes of disjunctions, namely split disjunctions, strong feasibility in the MCP is guaranteed.

Lemma 1. *If*

$$\bar{x} \in \operatorname{conv}\left(\bigcup_h \{x \mid Ax = b, D_h x \succeq_{\mathcal{Q}_h} d_h\}\right),$$

then the MCP (5.38) *is feasible.*

Proof. Immediate from Theorem 5. □

Lemma 2. *Assume that* $\forall h$, $\mathcal{Q}_h$ *is polyhedral. Then, the MCP* (5.38) *is strongly feasible if and only if it is feasible.*

Proof. Strong feasibility implies feasibility. Reciprocally, if (x, y, z, η) is feasible for the MCP (5.38), then $(x, y, z, \eta + \epsilon)$ is strongly feasible for any $\epsilon > 0$. □

Theorem 11. *If* $\mathcal{X} \neq \emptyset$ *and* $\mathcal{D}$ *is a split disjunction, i.e.,*

$$\mathcal{D} = \left\{ x \;\middle|\; \begin{array}{l} Ax = b, x \in \mathcal{K} \\ \pi^T x \leq \pi_0 \end{array} \right\} \cup \left\{ x \;\middle|\; \begin{array}{l} Ax = b, x \in \mathcal{K} \\ \pi^T x \geq \pi_0 + 1 \end{array} \right\},$$

then the MCP (5.38) *is strongly feasible.*

Proof. Let

$$\mathcal{S} = \operatorname{conv}\left(\left\{x \;\middle|\; Ax = b, \pi^T x \leq \pi_0\right\} \cup \left\{x \;\middle|\; Ax = b, \pi^T x \geq \pi_0 + 1\right\}\right).$$

First, assume there exists $\xi \in \mathbb{R}^n$ such that $A\xi = 0$ and $\pi^T \xi \neq 0$; without loss of generality, we can assume that $\pi^T \xi = 1$. Then, let $t \geq 0$ such that

$$\begin{aligned} \pi^T(\bar{x} - t\xi) &\leq \pi_0, \\ \pi^T(\bar{x} + t\xi) &\geq \pi_0 + 1. \end{aligned}$$

In particular, $A(\bar{x} \pm t\xi) = b$ and $\bar{x} = \frac{1}{2}(\bar{x} + t\xi) + \frac{1}{2}(\bar{x} - t\xi)$. Thus, $\bar{x} \in \mathcal{S}$, and it follows from Lemma 1 that the MCP (5.38) is feasible.

Now assume that $\forall \xi \in \ker(A), \pi^T \xi = 0$. On the one hand, if $\pi_0 < \pi^T \bar{x} < \pi_0 + 1$, then $\mathcal{S} = \emptyset$ and thus $\mathcal{X}$ is empty, which would contradict the $\mathcal{X} \neq \emptyset$ assumption. Thus, either $\pi^T \bar{x} \leq \pi_0$ or $\pi^T \bar{x} \geq \pi_0 + 1$, i.e., $\bar{x} \in \mathcal{S}$ and the MCP is feasible by Lemma 1. Note that this latter case would never occur in practice, since one would always consider a split such that $\pi_0 < \pi^T\bar{x} < \pi_0 + 1$.

The result then follows from Lemma 2. $\square$

Similar to the standard normalization, a consequence of Theorem 11 is that, when considering split disjunctions, the CGCP (5.37) and the MCP (5.38) are solvable with identical objective value. This is confirmed by the results of Table 5.2, which reports statistics for the CGCP in Example 4: no numerical issue is encountered. The corresponding split cuts are displayed in Figure 5.6, and are similar to the cuts obtained with the standard normalization. Likewise, cuts obtained with the uniform normalization are not, in general, supporting hyperplanes of the disjunctive hull.

5.5 Separating conic-infeasible points

When $(MICP)$ (5.17) is solved by outer-approximation, the fractional point $\bar{x}$ may not satisfy all conic constraints. In preliminary experiments, wherein lift-and-project cuts were separated by rounds in a callback, a large proportion –often higher than 90%– of the cuts yielded by the CGCP turned out to be $\mathcal{K}^*$ cuts, which is obviously detrimental to performance. To the best of our knowledge, despite the popularity and performance of outer-approximation algorithms, this behavior has not been studied in the literature. Therefore, in this section, we will assume that $A\bar{x} = b$, but $\bar{x} \notin \mathcal{K}$.

Let $(\alpha, \beta, u, \lambda, v)$ be a solution of the CGCP, and assume that $v_h = 0$ for some $h \in \{1, ..., H\}$. Thus, we have

$$\alpha = A^T u_h + \lambda_h, \tag{5.39}$$

$$\beta \leq b^T u_h, \tag{5.40}$$

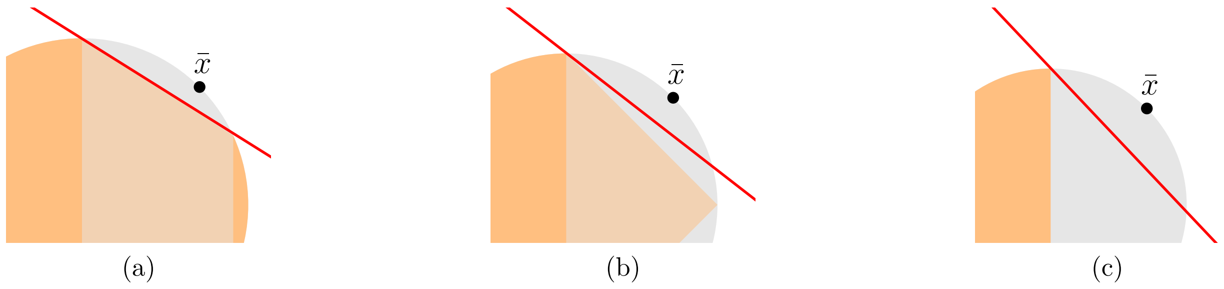


Figure 5.6 Split cuts obtained with the uniform normalization. The continuous relaxation is in gray, the split hull in orange, and the obtained cut is in red.

Table 5.5 CGCP statistics for Example 4 and uniform normalization

	Iter	$\|\alpha\|$	$\|u_1\|$	$\|u_2\|$	$\|\lambda_1\|$	$\|\lambda_2\|$	$\|v_1\|$	$\|v_2\|$
(a)	8	0.5	0.0	0.3	0.5	0.9	0.2	0.3
(b)	8	0.5	0.0	0.4	0.4	1.0	0.2	0.4
(c)	8	0.4	0.0	0.5	0.3	1.1	0.3	0.5

and $\alpha^T x \geq \beta$ is a trivial inequality. In addition, as noted in Section 5.3.1, the inequality $\lambda_h^T x \geq 0$ has the same violation, and cuts off the same portion of the continuous relaxation as $\alpha^T x \geq \beta$. This observation, which does not depend on the normalization condition, allows the *a posteriori* detection of $\mathcal{K}^*$ cuts, by checking the value of the v multipliers.

Once a $\mathcal{K}^*$ cut is identified, it can be disaggregated. Assume that $\mathcal{K} = \mathcal{K}_1 \times ... \times \mathcal{K}_N$; correspondingly, for $\lambda \in \mathcal{K}^*$, we write $\lambda = (\lambda_1, ..., \lambda_N)$ where each $\lambda_i \in \mathcal{K}_i^*$. Then, the $\mathcal{K}^*$ cut $\lambda^T x \geq 0$ is disaggregated as

$$\lambda_i^T x_i \geq 0, \quad i = 1, ..., N. \tag{5.41}$$

This yields more numerous, but sparser, $\mathcal{K}^*$ cuts, and results in tighter polyhedral approximations which, in turn, improves the performance of outer-approximation algorithms [130].

We now derive sufficient conditions that provide an *a priori* indication that a $\mathcal{K}^*$ cut will be generated. Unless stated otherwise, we only consider the CGCP with standard normalization. Define

$$\bar{\eta} = \min_{\eta \geq 0} \left\{ \eta \mid \bar{x} + \eta\rho \in \mathcal{K} \right\}, \tag{5.42}$$

$$\bar{\tau}_h = \min_{\tau \geq 0} \left\{ \tau \mid D_h \bar{x} + \tau\sigma_h \succeq_{\mathcal{Q}_h} d_h \right\}, \qquad \forall h, \tag{5.43}$$

and let $\xi = \bar{x} + \bar{\eta}\rho$.

Lemma 3. *Assume there exists an optimal solution of the CGCP for which $v_h = 0, \forall h$. Then, the optimal value of the CGCP is $-H^{-1}\bar{\eta}$, and there exist a CGCP-optimal solution of the form $(\lambda_0, 0, 0, \lambda_0, 0)$, with $\lambda_0^T \xi = 0$.*

Proof. Let $(\alpha, \beta, u, \lambda, 0)$ be such an optimal solution, and denote by δ its objective value. In particular, we have

$$\begin{aligned} \alpha &= A^T u_h + \lambda_h, && \forall h, \\ \beta &\leq b^T u_h, && \forall h. \end{aligned}$$

Let $\bar{u} = \frac{1}{H}\sum_h u_h$ and $\bar{\lambda} = \frac{1}{H}\sum_h \lambda_h$. It follows that $(\alpha, \beta, \bar{u}, \bar{\lambda}, 0)$ is feasible with objective value δ, i.e., it is an optimal solution of the CGCP.

Next, $(\alpha, \beta, \bar{u}, \bar{\lambda}, 0)$ is also an optimal solution of

$$\begin{aligned}
\min_{u,\lambda} \quad & \alpha^T \bar{x} - \beta \\
s.t. \quad & \alpha = A^T u_h + \lambda_h, && \forall h, \\
& \beta \leq b^T u_h, && \forall h, \\
& \sum_h |\lambda_h|_\rho \leq 1, \\
& \lambda_h \in \mathcal{K}^*.
\end{aligned}$$

Eliminating α, β yields

$$\begin{aligned}
\min_{\lambda} \quad & \lambda^T \bar{x} \\
s.t. \quad & \rho^T \lambda \leq \frac{1}{H}, \\
& \lambda \in \mathcal{K}^*,
\end{aligned}$$

whose dual, up to a change of sign in the objective value, writes

$$\begin{aligned}
\min_{\eta} \quad & \frac{1}{H}\eta \\
s.t. \quad & \eta\rho \preceq_{\mathcal{K}} \bar{x}, \\
& \eta \geq 0,
\end{aligned}$$

and has optimal value $H^{-1}\bar{\eta}$. Thus, $\delta = -H^{-1}\bar{\eta} = \bar{\lambda}^T \bar{x}$, which concludes the proof. □

Theorem 12. *If* $\forall h, \bar{\eta} \geq H^{-1}\bar{\tau}_h$*, then* $(\bar{\lambda}, 0, 0, \bar{\lambda}, 0)$ *is optimal for CGCP.*

Proof. Let $y_h = \frac{1}{H}\bar{x}$, $z_h = \frac{1}{H}$, and $\eta = \frac{\bar{\eta}}{H}$.

We have $Ay_h = z_h b, \forall h$. Then, $y_h + \eta\rho = \frac{1}{H}(\bar{x} + \bar{\eta}\rho) \in \mathcal{K}$. Finally,

$$\begin{aligned}
D_h y_h + \eta\sigma_h &= H^{-1}\left(D_h \bar{x} + \bar{\eta}\sigma_h\right) \\
&\succeq_{\mathcal{Q}_h} H^{-1} d_h \\
&= z_h d_h.
\end{aligned}$$

Thus, (y, z, η) is feasible for the MCP and its objective value is $H^{-1}\bar{\eta}$, which concludes the proof. □

Theorem 12 shows that, if $\bar{x}$ is “sufficiently" conic-infeasible, as measured by the magnitude of $\bar{\eta}$, then there exists a $\mathcal{K}^*$ cut that is an optimal solution of the CGCP. Note that there is no guarantee that this optimal solution is unique –in general, it is not– nor that all CGCP-optimal solutions are $\mathcal{K}^*$ cuts. Nevertheless, we have observed that, whenever the condition of Theorem 12 was met, the obtained solution was indeed a $\mathcal{K}^*$ cut.

This suggests several strategies to avoid generating $\mathcal{K}^*$ cuts when solving the CGCP. First, one can check the value of $\bar{\eta}$ and, if large enough as per Theorem 12, avoid solving the CGCP and add an optimal $\mathcal{K}^*$ cut directly. Nevertheless, our initial experiments suggest that only a small number of cases are captured by Theorem 12. Second, one can increase the magnitude of ρ, thus reducing the value of $\bar{\eta}$, to the point where the assumptions of Theorem 12 no longer hold. Note that, as the magnitude of ρ becomes arbitrarily large, the standard normalization becomes equivalent to the uniform normalization of Section 5.4.5. Third, instead of imposing a normalization condition, one could fix the cut violation to a positive value, e.g., 1, and optimize a different objective; the feasibility of the CGCP is then guaranteed by the fact that $\bar{x}$ is conic-infeasible. This approach directly relates to the *reverse-polar* CGLP introduced in [139]. Finally, one can simply try to avoid conic-infeasible points in the first place, for instance by refining the outer-approximation before cuts are separated.

5.6 Cut lifting and strenghtening

In this section, we present conic extensions of the classical lifting and strengthening procedures in MILP. For simplicity, we will assume that $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2$ where $\mathcal{K}_i \subseteq \mathbb{R}^{n_i}, i = 1, 2$ and, correspondingly, we write

$$A = \begin{bmatrix} A_1 & A_2 \end{bmatrix}, \; D_h = \begin{bmatrix} D_{1,h} & D_{2,h} \end{bmatrix}, \; \lambda_h = \begin{bmatrix} \lambda_{1,h} \\ \lambda_{2,h} \end{bmatrix}, \; \alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}, \; \bar{x} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix}.$$

Finally, we assume that $\bar{x}_2 = 0$.

5.6.1 Cut lifting

In MILP, one can formulate the CGLP in a reduced space, by projecting out the null components of $\bar{x}$, then recover a valid cut in the original space by a lifting procedure. Here, we show that a similar technique can be used in the conic setting.

Recall that $\bar{x}_2 = 0$, and consider the reduced CGCP

$$\begin{aligned}
\min \quad & \alpha_1^T \bar{x}_1 - \beta && && (5.44a)\\
s.t. \quad & \alpha_1 = A_1^T u_h + \lambda_{1,h} + D_{1,h}^T v_h, && \forall h, && (5.44b)\\
& \beta \leq b^T u_h - v_h^T d_h, && \forall h, && (5.44c)\\
& (u_h, \lambda_{1,h}, v_h) \in \mathbb{R}^m \times \mathcal{K}_1^* \times \mathcal{Q}_h^*, && \forall h. && (5.44d)
\end{aligned}$$

All the normalization conditions considered in Section 5.4 can be adapted to the reduced CGCP, namely by normalizing only the α_1, λ_1, v components as appropriate. For instance, the α normalization would write $\|\alpha_1\|_* \leq 1$, and the uniform normalization $\sum_h |\lambda_{1,h}|_{\rho_1} \leq 1$, for $\rho_1 \in \operatorname{int} \mathcal{K}_1$.

Any solution $(\alpha, \beta, u, \lambda, v)$ that is feasible for the CGCP yields a feasible solution $(\alpha_1, \beta, u, \lambda_{1,.}, v)$ for the reduced CGCP. In addition, since $\bar{x}_2 = 0$, the corresponding objective values are the same. Reciprocally, a feasible solution for the CGCP can be obtained from a feasible solution of the reduced CGCP as shown in Lemma 4.

Lemma 4. *Let $(\alpha_1, \beta, u, \lambda_{1,.}, v)$ be feasible for the reduced CGCP* (5.44)*. Let*

$$\alpha_2 \succeq_{\mathcal{K}_2^*} A_2^T u_h + D_{h,2}^T v_h, \qquad \forall h,$$

and, $\forall h$, let $\lambda_{2,h} = \alpha_2 - A_2^T u_h - D_{2,h}^T v_2$. Then, $(\alpha, \beta, u, \lambda, v)$ is feasible for the CGCP (5.19) *and $\alpha^T \bar{x} - \beta = \alpha_1^T \bar{x}_1 - \beta$.*

The proof of Lemma 4 is immediate. Note that the choice of α_2 is not unique, especially if $\mathcal{K}_2^*$ possesses high-dimensional faces. Nevertheless, a reasonable requirement is to impose that α_2 be minimal with respect to $\succeq_{\mathcal{K}^*}$, i.e., that there does not exist a valid $\tilde{\alpha}_2 \preceq_{\mathcal{K}^*} \alpha_2$. Indeed, if α_2 is not $\succeq_{\mathcal{K}^*}$-minimal, then $\alpha^T x \geq \beta$ is trivially dominated by $\tilde{\alpha}^T x \geq \beta$.

A $\succeq_{\mathcal{K}^*}$-minimal α_2 is obtained by solving the lifting conic problem (LCP)

$$\begin{aligned}
(LCP) \quad \min_{\alpha_2} \quad & \rho_2^T \alpha_2 && && (5.45a)\\
s.t. \quad & \alpha_2 \succeq_{\mathcal{K}_2^*} A_2^T u_h + D_{h,2}^T v_h, && \forall h, && (5.45b)
\end{aligned}$$

where $\rho_2 \in \operatorname{int} \mathcal{K}_2$, and denote by $\lambda_{2,h} \in \mathcal{K}_2^*$ the (conic) slack associated to (5.45b). First, since $\mathcal{K}_2$ is a proper cone, the LCP is strongly feasible. Second, we have

$$\rho_2^T \alpha_2 = \rho_2^T \lambda_{2,h} + \rho_2^T \left(A_2^T u_h + D_{h,2}^T v_h \right), \qquad \forall h, \qquad (5.46)$$

thereby showing that the objective value of the LCP is bounded below. In addition, let $(\alpha_2, \lambda_{2,.})$ be a feasible solution for the LCP with objective value Z. It then follows that

$$\forall h, \quad \left|\tilde{\lambda}_{2,h}\right|_{\rho_2} \leq Z - \rho_2^T \left(A_2^T u_h + D_{h,2}^T v_h \right) \tag{5.47}$$

in any feasible solution $(\tilde{\alpha}_2, \tilde{\lambda}_{2,.})$ with objective value $\tilde{Z} \leq Z$. Equation (5.47) implicitly bounds the magnitude of $\lambda_{2,.}$, thereby ensuring that the LCP is solvable. Finally, if α_2 and $\tilde{\alpha}_2$ are feasible and $\tilde{\alpha}_2 \preceq_{\mathcal{K}^*} \alpha_2$, then $\rho_2^T \tilde{\alpha}_2 < \rho_2^T \alpha_2$ and α_2 cannot be an optimal solution. Thus, any optimal solution of the LCP is $\succeq_{\mathcal{K}^*}$-minimal.

Whenever $\mathcal{K}_2$ is not irreducible, the LCP can be decomposed per conic component, yielding smaller problems. In the linear case, taking $\mathcal{K}_2 = \mathbb{R}_+$, the LCP reduces to

$$\alpha_2 = \max_h \left\{ A_2^T u_h + D_{2,h}^T v_h \right\},$$

which is the classical lifting procedure for disjunctive cuts in MILP [136].

Lemma 4 does not account for the normalization constraint in the CGCP. Although the lifting procedure does not modify the objective value, in general, the lifted solution $(\alpha, \beta, u, \lambda, v)$ is not an optimal solution of the normalized original CGCP. Thus, the reduction in the size of the CGCP, and the associated computational gains, come at the expense of potentially weaker cuts.

5.6.2 Cut strengthening

Balas and Jeroslow's original derivation of monoidal strengthening for disjunctive cuts [135] exploited the non-negativity and integrality of *individual* variables to strengthen the corresponding cut coefficients. Since, in general, a conic constraint may involve several variables, it is not clear whether and how one can extend this approach to the conic setting. Thus, we restrict our attention to split cuts, and propose a conic extension of monoidal strengthening that builds on the geometric idea of Wolsey's proof of Theorem 2.2 in [134].

For simplicity, we consider the pure integer case. Given a split disjunction $(\pi^T x \leq \pi_0) \vee$

$(\pi^T x \geq \pi_0 + 1)$, the CGCP writes

$$\begin{aligned} \min \quad & \alpha^T \bar{x} - \beta \\ s.t. \quad & \alpha = A^T u_1 + \lambda_1 - v_1 \pi \\ & \alpha = A^T u_2 + \lambda_2 + v_2 \pi \\ & \beta \leq b^T u_1 - v_1 \pi_0 \\ & \beta \leq b^T u_2 + v_2(\pi_0 + 1) \\ & \lambda_1, \lambda_2 \in \mathcal{K}^*, v_1, v_2 \geq 0. \end{aligned}$$

Lemma 5. *Let $(\alpha, \beta, u, \lambda, v)$ be feasible for the CGCP, and let $\tilde{\alpha}_2$ and $\delta\pi \in \mathbb{Z}^{n_2}$ such that*

$$\begin{aligned} \tilde{\alpha}_2 &\succeq_{\mathcal{K}_2^*} A_2^T u_1 - v_1(\pi_2 + \delta\pi), \\ \tilde{\alpha}_2 &\succeq_{\mathcal{K}_2^*} A_2^T u_2 + v_2(\pi_2 + \delta\pi). \end{aligned}$$

Then, $\tilde{\alpha}^T x \geq \beta$ is a valid inequality for the disjunctive set

$$\tilde{\mathcal{D}} = \left\{ x \,\middle|\, \begin{array}{l} Ax = b, x \in \mathcal{K} \\ \tilde{\pi}^T x \leq \pi_0 \end{array} \right\} \cup \left\{ x \,\middle|\, \begin{array}{l} Ax = b, x \in \mathcal{K} \\ \tilde{\pi}^T x \geq \pi_0 + 1 \end{array} \right\},$$

where $\tilde{\alpha} = (\alpha_1, \tilde{\alpha}_2)$ and $\tilde{\pi} = (\pi_1, \pi_2 + \delta\pi)$.

In the mixed-integer case, one simply needs to set to zero, in Lemma 5, the components of $\delta\pi$ that correspond to continuous variables. Since $\delta\pi \in \mathbb{Z}^{n_2}$, $\tilde{\pi} \in \mathbb{Z}^n$ and $(\tilde{\pi}^T x \leq \pi_0) \vee (\tilde{\pi}^T x \geq \pi_0 + 1)$ is a valid disjunction for $\mathcal{X}$. Thus, the strengthened inequality is valid for $\mathcal{X}$.

Similar to the lifting case, a reasonable requirement is for $\tilde{\alpha}_2$ to be minimal with respect to $\succeq_{\mathcal{K}_2^*}$. Following the same approach as in Section 5.6.1, we obtain the *Cut-Strengthening Problem*

$$\begin{aligned} (CSP) \quad \min_{\tilde{\alpha}_2} \quad & \rho_2^T \tilde{\alpha}_2 && (5.48a) \\ s.t. \quad & \tilde{\alpha}_2 \succeq_{\mathcal{K}_2^*} A_2^T u_1 - v_1(\pi_2 + \delta\pi), && (5.48b) \\ & \tilde{\alpha}_2 \succeq_{\mathcal{K}_2^*} A_2^T u_2 + v_2(\pi_2 + \delta\pi), && (5.48c) \\ & \delta\pi \in \mathbb{Z}^{n_2}. && (5.48d) \end{aligned}$$

Similar to the LCP, the CSP can be decomposed per conic component, so that its resolution remains tractable. Furthermore, in the linear case with $\mathcal{K}_2 = \mathbb{R}_+$, the CSP reduces to

$$\tilde{\alpha}_2 = \max_{\delta \in \mathbb{Z}} \left\{ \min \left(A_2^T u_1 - v_1 \delta, A_2^T u_2 + v_2 \delta \right) \right\},$$

which is the classical monoidal strengthening of Balas and Jeroslow [135].

5.7 Computational results

In this section, we investigate the practical behavior of the normalization conditions of Section 5.4 along two lines: the progression of the gap closed, and the characteristics of the obtained cuts. Indeed, the choice of normalization impacts the numerical stability and computational efficiency of solving the CGCP, thereby affecting the rate at which gap is closed, in terms of both time and number of rounds.

Several solvers, e.g., CPLEX, Gurobi and Mosek, support classes of MI-CONIC problems. Nevertheless, to the best of our knowledge, CPLEX is the only one that exploits nonlinear information when generating lift-and-project cuts. Thus, we use CPLEX as a baseline, and restrict our comparison to MISOCP instances, which is the only class of nonlinear MI-CONIC problems supported by CPLEX.

5.7.1 Instances

We select an initial testset of 114 MISOCP instances from the CBLIB [165] collection. Each instance is first reformulated in the standard form (5.17) and, since the MOI wrapper of CPLEX does not directly support constraints of the form $x \in \mathcal{K}$ where $\mathcal{K}$ is a rotated second-order cone, all such constraints are reformulated as second-order cone constraints.

Each instance is solved using the CPLEX outer-approximation algorithm on a single thread, and all other parameters left to their default value. Instances with a root gap smaller than 1% are removed from the testset, as well as those for which no integer-feasible solution was found by CPLEX after one hour of computing time. This yields a testset of 101 instances, divided into 7 groups. Table 5.6 reports, for each group, the number of instances (#Inst) in that group, and the average number of: variables, integer variables, constraints, non-polyhedral cones, and non-zero coefficients.

Table 5.6 Instance statistics

Group	#Inst	Variables	Integers	Constraints	Cones	Nz coeff.
`clay`	12	829	35	659	80	1711
`flay`	10	535	28	408	4	1121
`fmo`	39	620	48	479	15	1843
`slay`	14	1217	92	936	14	2653
`sssd`	16	694	153	547	18	1414
`tls`	2	295	61	227	10	817
`uflquad`	8	37569	23	30203	3750	71341

5.7.2 Implementation details

Our implementation is coded in Julia 1.4.[2] All solvers, namely, CPLEX 12.10 [17], Gurobi 9.0 [116] and Mosek 9.2 [117], are accessed through the solver-agnostic interface `MathOptInterface` [103]. Experiments are carried out on an Intel Xeon E5-2637@3.50GHz CPU, 128GB RAM machine running Linux.

Baseline We use CPLEX internal lift-and-project cut generation at the root node as a baseline.

For each instance, we run CPLEX outer-approximation algorithm on a single thread, no presolve, no heuristics, and all cuts de-activated to the exception of lift-and-project cuts which are set to the most aggressive setting. The `CutsFactor` parameter is set to 10^{30}, thereby removing any limit on the number of cuts that can be added to the formulation, and the maximum number of cutting plane passes is set to 200. Finally, the node limit is set to zero, i.e., only the root node is explored, and we set a time limit of 1 hour.

Cut separation Cuts are separated by rounds in a callback, and submitted to CPLEX as user cuts. Each round proceeds as follows.

First, $\bar{x}$ is cleaned, i.e, we set $\bar{x}_j = 0$ for all j such that $|\bar{x}_j| \leq 10^{-7}$, and $\bar{\eta}$ is computed as per Equations (5.42). If $\bar{\eta} > \epsilon_{\mathcal{K}}$, the current outer-approximation is refined by adding violated $\mathcal{K}^*$ cuts. This refinement step is cheap, and it is repeated until $\bar{x} \leq \epsilon_{\mathcal{K}}$, up to a maximum of 50 times. Large values for $\epsilon_{\mathcal{K}}$ increase the likelihood of generating $\mathcal{K}^*$ cuts from the CGCP, while small values can lead to an adverse tailing-off effect within the refinement process; following initial tests, we set $\epsilon_{\mathcal{K}} = 0.05$.

Second, for each fractional coordinate of $\bar{x}$, the CGCP is formed and solved to generate a

[2]Code will be publicly released upon publication.

lift-and-project cut. We do not project the CGCP onto the support of $\bar{x}$, i.e., no lifting is performed. For the standard normalization, the sufficient conditions of Theorem 12 are checked first and, if met, $\mathcal{K}^*$ cuts are added and no CGCP is solved. If no CGCP solution is available, or if the objective value of the CGCP is greater than -10^{-4}, no cut is generated and we proceed to the next fractional coordinate.

Third, the values of v_1 and v_2 are checked to identify $\mathcal{K}^*$ cuts, which are disaggregated and added to the formulation. Otherwise, the cut is strengthened following the procedure of Section 5.6.2. For stability, strengthening is performed only if $|v_1| + |v_2| \geq 10^{-4}$. Then, also for numerical stability, we set α_j to zero for every j such that $|\alpha_j| \leq 10^{-7}$. Similarly, if $|\beta| \leq 10^{-8}$, the cut's right-hand side is set to zero. Finally, the cut is passed to the solver.

Compact CGCP Lift-and-project cuts are obtained from elementary split disjunctions, i.e., disjunctions of the form

$$\left(\pi^T x \leq \pi_0\right) \vee \left(\pi^T x \leq \pi_0 + 1\right),$$

where $\pi = e_j$ for some $j \in \{1, ..., p\}$. When formulating the CGCP, we set u_1 to zero, and substitute out α and β. This yields the compact CGCP

$$\begin{aligned}
\min_{u,\lambda,v} \quad & \bar{x}^T \lambda_1 - (\pi^T \bar{x} - \pi_0) v_1 + t_1 \\
s.t. \quad & A^T u_2 + (\lambda_2 - \lambda_1) + (v_1 + v_2)\pi = 0 \\
& b^T u_2 + (t_1 - t_2) + (v_1 + v_2)\pi_0 + v_2 = 0 \\
& \lambda_1, \lambda_2 \in \mathcal{K}^*, \\
& v_1, v_2 \geq 0 \\
& t_1, t_2 \geq 0,
\end{aligned}$$

where t_1, t_2 are non-negative slacks associated to constraints (5.19c). The relation $\alpha = \lambda_1 - v_1 \pi$ allows to formulate the normalization condition in the λ, v space.

In practice, the equality constraints $A\bar{x} = b$ are satisfied only up to numerical tolerances. This can cause the CGCP to be unbounded whenever the u multipliers are not bounded. We have observed that setting u_1 to zero, and writing the objective as in the compact CGCP above, greatly improve the numerical stability of the CGCP, especially when $\bar{x}$ is obtained from an interior-point method.

Other details Cuts that are generated are never added to the CGCP formulation, i.e., we only separate rank-1 lift-and-project cuts. All CGCPs are solved with Gurobi, which we found to be more robust here. We use the barrier algorithm with a single thread, no presolve, and we disable the computation of dual variables by setting the `QCPDual` parameter to 0.

We implement a simple procedure to identify implied-integer variables. If a variable y appears in a constraint of the form $a^T x \pm y = b$, where $b \in \mathbb{Z}$, all coefficients of a are integers, and x is a vector of integer or implied-integer variables, then y is an implied-integer variable. This step is repeated until no additional implied-integer variable is detected. Then, when generating cuts, the coefficients of implied-integer variables can be strengthened using the technique of Section 5.6.2.

Since we use an outer-approximation algorithm, we do not include the trivial normalization in our experiments. The α normalization is formulated using ℓ_2 norm. For the polar normalization, we take $\gamma = x^* - \bar{x}$, where x^* is the analytic center of $\mathcal{C}$, obtained from solving (CP) with zero objective by an interior-point method. Finally, for the standard and uniform normalization, and the computation of $\bar{\eta}$ in Equation (5.42), we set $\rho = (\rho_1, ..., \rho_N)$, where

$$\rho_i = \begin{cases} 1 & \text{if } \mathcal{K}_i = \mathbb{R}_+ \\ -1 & \text{if } \mathcal{K}_i = \mathbb{R}_- \\ (1,0,0) & \text{if } \mathcal{K}_i = \mathcal{L}_3 \end{cases},$$

and, since we only consider split cuts, we take $\sigma_1 = \sigma_2 = 1$.

5.7.3 Gap closed

Tables 5.7 and 5.8 report the performance of each approach after 10 and 200 rounds of cuts, respectively. For each normalization (Alpha, Polar, Standard, Uniform) and the baseline (CPLEX), we report the average percent gap closed (Gap), and the average computing time (CPU), in seconds. The percent gap closed is measured as

$$Gap = 100 \times \frac{Z^* - Z_{CP}}{Z_{MICP} - Z_{CP}},$$

where Z_{CP}, Z_{MICP}, Z^* are the optimal value of the continuous relaxation, the objective value of the best known integer solution, and the current lower bound after adding cuts, respectively. All averages are shifted geometric means with a shift of 1.

Importantly, our definition of a “round" of cuts is likely different from a CPLEX “pass" – which is unknown to us. In addition, total computing times include the time of building the CGCP, which typically represent 30 to 40% of total time, and up to 90% for `uflquad` instances. These large building times are in part due to limitations of the Gurobi Julia wrapper, and could be significantly reduced by using a lower-level interface. Thus, direct comparisons between CGCP-based approaches and the baseline should be cautious.

First, overall, to the exception of the `tls` and `uflquad` instances, the polar normalization performs the worst, with 0.46% and 0.71% gap closed after up to 10 and 200 rounds, respectively. This poor performance is primarily caused by premature termination of the cut generation procedure: in numerous cases, the CGCP is unbounded, no solution is available, and no cut is generated.

Second, the α normalization is the slowest on average, with computing times that are typically 2 to 3 times higher than other approaches. This behavior results from the CGCP (5.27) being more computationally intensive than for alternative normalizations. Indeed, the normalization (5.27e) is nonlinear and, as highlighted in Section 5.4.1, the CGCP (5.27) may not be solvable, then causing slow convergence and more interior-point iterations. The α normalization also performs second-worst with respect to gap closed, with 1.02% and 2.59% gap closed after 10 and 200 rounds, respectively.

Third, the standard and uniform normalizations display similar performance, to the exception of `flay` instances, for which the uniform normalization closes significantly more gap than all other approaches. While both normalizations close similar gaps, namely, 17.97% and 17.27% after 10 rounds, and 32.67% and 39.13% after 200 rounds, the uniform normalization is slightly faster overall, taking an average 65.5s to complete up to 200 rounds, against 85.0s for the standard normalization. Furthermore, no numerical issues were recorded for either normalization, thereby demonstrating the benefits of ensuring strong feasibility of both the CGCP and the MCP.

Fourth and last, the standard and uniform normalizations are competitive with CPLEX in terms of gap closed. Both normalizations close more gap that CPLEX after 200 rounds, mainly on `slay` and `uflquad` instances, as well as `flay` instances for the uniform normalization. In particular, they close over three times more gap than CPLEX in the first 10 rounds. Despite the limited validity of the comparison, this behavior is encouraging, since closing more gap early is a desirable feature in practice.

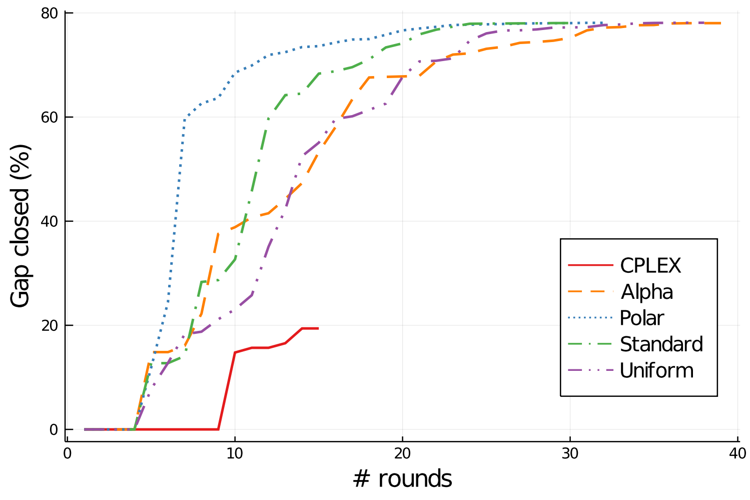
Next, Figures 5.7, 5.8 and 5.9 illustrate the progression of the gap closed for instances `flay02m`, `sssd-weak-15-4` and `tls4`, respectively. Each figure displays, for each normalization and the baseline, the percent gap closed as a function of the number of rounds, and of computing time. Again, recall that what constitutes a “round" differs between our approach and CPLEX, thus, direct comparisons may not be meaningful.

Table 5.7 Gap closed and computing time - 10 rounds

	CPLEX		Alpha		Polar		Standard		Uniform	
Group	Gap	CPU	Gap	CPU	Gap	CPU	Gap	CPU	Gap	CPU
`clay`	2.13	1.0	0.00	12.0	0.00	4.4	5.69	6.0	2.92	5.7
`flay`	0.32	0.3	0.45	3.7	1.50	2.1	0.88	2.1	2.40	1.7
`slay`	3.67	2.4	0.65	23.2	0.15	8.8	15.81	11.2	10.57	8.7
`fmo`	6.42	0.7	0.00	11.7	0.00	5.2	25.31	5.6	25.59	4.6
`sssd`	45.07	0.9	36.49	12.3	0.00	6.6	93.44	5.3	96.47	7.9
`tls`	8.13	0.5	2.56	2.6	20.57	1.9	8.47	1.4	13.36	1.5
`uflquad`	0.46	56.8	0.00	1307.9	12.50	1568.7	18.61	1769.8	18.98	1721.5
All	5.24	1.5	1.02	16.9	0.46	8.5	17.97	9.2	17.27	8.6

Table 5.8 Gap closed and computing time - 200 rounds

	CPLEX		Alpha		Polar		Standard		Uniform	
Group	Gap	CPU	Gap	CPU	Gap	CPU	Gap	CPU	Gap	CPU
`clay`	24.48	32.3	0.00	251.4	0.00	6.2	22.67	184.9	22.72	165.7
`flay`	6.42	1.0	2.62	50.1	8.59	8.6	6.65	23.0	41.25	19.1
`slay`	24.70	9.7	16.75	490.2	0.20	19.6	83.44	196.2	84.85	87.7
`fmo`	25.89	10.5	0.00	243.0	0.00	21.7	26.00	68.1	26.01	46.7
`sssd`	95.88	4.1	95.65	284.4	0.00	8.3	99.43	14.9	99.46	17.8
`tls`	41.47	12.8	3.07	19.9	20.96	7.1	31.90	13.0	34.76	14.5
`uflquad`	5.39	475.0	0.00	1307.9	16.22	3940.6	23.15	4178.0	23.81	4140.0
All	24.79	11.9	2.59	256.9	0.71	21.9	32.67	85.0	39.13	65.5



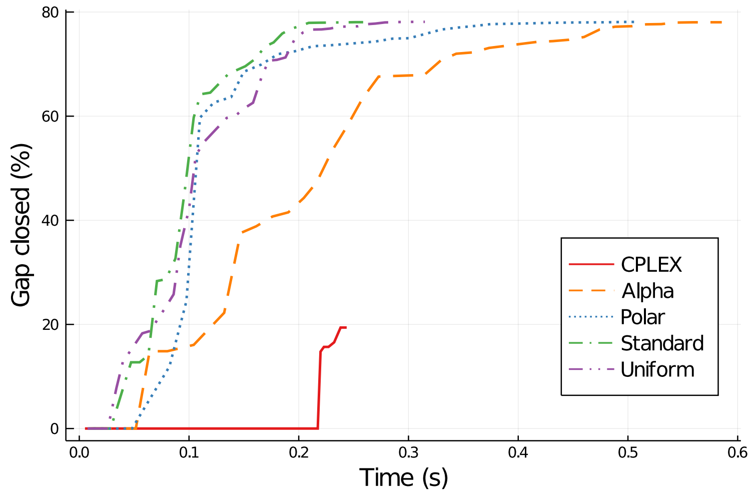
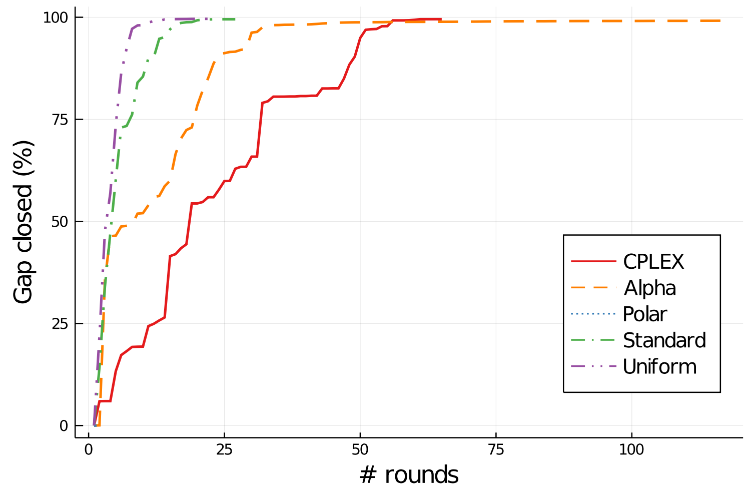


Figure 5.7 Instance `flay02m`: gap closed per round (top) and time (bottom)



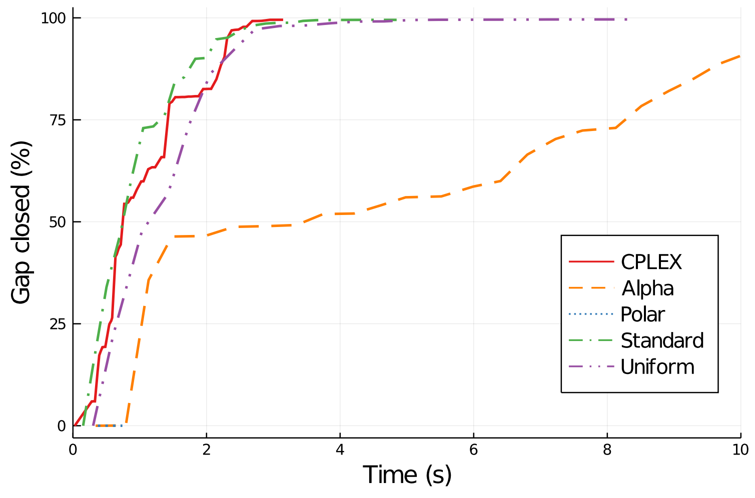
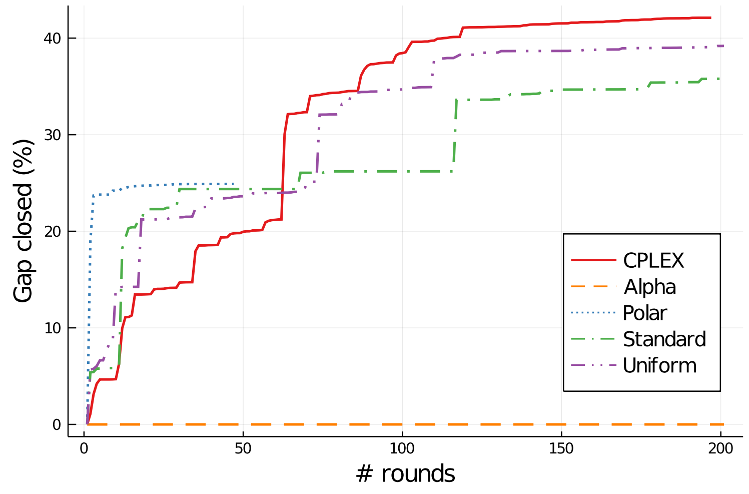


Figure 5.8 Instance `sssd-weak-15-4`: gap closed per round (top) and time (bottom). The polar normalization was terminated after two rounds.



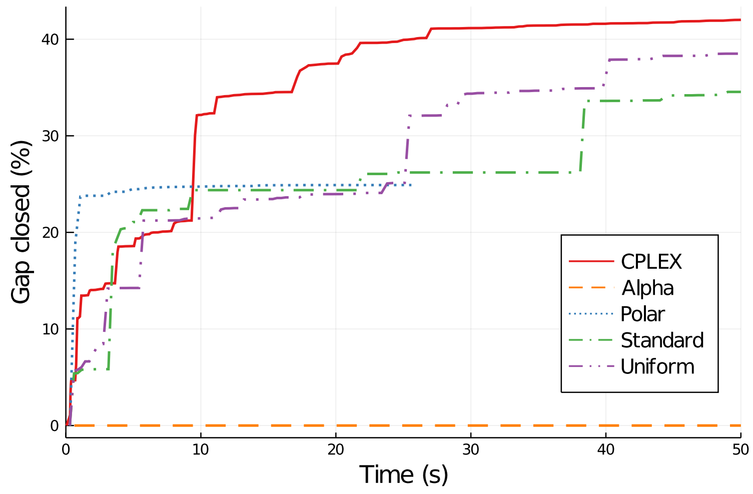


Figure 5.9 Instance `tls4`: gap closed per round (top) and time (bottom)

Overall, Figures 5.7, 5.8 and 5.9 corroborate the previous observations from Tables 5.7 and 5.8: the α normalization is the slowest, while the standard and uniform normalizations display similar progressions and allow to close more gap than CPLEX in the first few rounds. We also observe in Figure 5.7 and Figure 5.9 that, when the polar normalization does not run into numerical issues, it can outperform other approaches for the most gap closed in the first few rounds of cuts.

Furthermore, when comparing the gap closed with respect to time, we observe that the CGCP-based separation procedure is competitive with CPLEX. This is most striking for instance `flay02m`, where all four normalizations outperform CPLEX. For instances `sssd-weak-15-4` and `tls4`, the standard and uniform normalizations are initially on-par with CPLEX, while the α normalization is the slowest.

5.7.4 Cut sparsity

We now study, for each normalization, the characteristics of the cuts obtained from solving the CGCP. Relevant statistics are reported after 10 and 200 rounds, in Table 5.9 and Table 5.10, respectively. For each normalization, we report the total number of disaggregated $\mathcal{K}^*$ cuts ($\mathcal{K}^*$) and of lift-and-project cuts (LaP). We also report the average density of lift-and-project cuts (%nz), measured as the percentage of non-zero coefficients per cut; the density of $\mathcal{K}^*$ cuts is not reported because they are always disaggregated. Here, $\mathcal{K}^*$ cuts refer specifically to cuts obtained from the CGCP that were identified as $\mathcal{K}^*$ cuts, i.e., we do not consider those added in the refinement steps. Thus, the first row of Table 5.9 indicates that, for `clay` instances and after 10 rounds, the CGCP with standard normalization yielded an average of 352.9 disaggregated $\mathcal{K}^*$ cuts, and 70.4 lift-and-project cuts with an average density of 2.5%.

Table 5.9 Cut statistics - 10 rounds

	Alpha			Polar			Standard			Uniform		
Group	$\mathcal{K}^*$	LaP	%nz	$\mathcal{K}^*$	LaP	%nz	$\mathcal{K}^*$	LaP	%nz	$\mathcal{K}^*$	LaP	%nz
clay	0.0	135.6	44.9	0.0	1.4	4.7	352.9	70.4	2.5	59.9	96.2	1.4
flay	0.0	121.2	27.6	0.0	33.1	4.3	17.9	110.1	3.6	11.9	118.3	2.3
slay	0.0	268.9	27.1	0.0	32.6	6.4	48.3	185.4	1.9	15.7	208.1	1.3
fmo	74.0	245.5	35.5	20.7	25.1	8.8	99.8	236.5	1.7	68.5	230.6	1.3
sssd	0.0	282.5	16.5	0.0	51.1	8.0	165.6	128.0	7.4	132.3	147.9	8.0
tls	363.2	123.9	13.3	94.2	113.0	5.8	91.4	84.3	7.2	39.2	104.5	7.0
uflquad	0.0	42.1	34.0	0.0	123.9	1.3	213.3	51.6	0.5	154.8	63.8	0.5
All	14.4	191.4	29.7	6.1	30.1	6.2	105.2	146.1	2.5	57.5	159.9	1.9

Table 5.10 Cut statistics - 200 rounds

	Alpha			Polar			Standard			Uniform		
Group	$\mathcal{K}^*$	LaP	%nz	$\mathcal{K}^*$	LaP	%nz	$\mathcal{K}^*$	LaP	%nz	$\mathcal{K}^*$	LaP	%nz
clay	0.0	2569.7	44.9	0.0	1.7	4.6	6368.9	1290.5	3.8	3005.0	2884.0	2.1
flay	0.0	1961.0	24.4	0.0	188.0	5.0	401.9	1528.9	4.1	210.3	1492.7	3.2
slay	2.2	5491.0	21.6	0.0	71.7	6.5	2941.9	2311.1	5.6	477.0	1766.1	4.6
fmo	1851.8	5060.3	33.9	131.4	133.6	9.1	464.2	2496.6	4.1	311.7	2068.1	3.3
sssd	0.0	5821.6	10.4	0.0	66.1	8.2	344.5	430.3	14.2	224.7	363.4	12.6
tls	2038.8	752.1	13.7	308.2	363.9	6.3	574.1	583.1	8.1	449.7	803.1	8.1
uflquad	0.0	42.1	34.0	0.0	294.2	1.2	412.6	135.8	0.4	346.1	160.5	0.3
All	76.0	2943.1	26.1	19.8	89.6	6.4	769.7	1277.4	4.7	403.5	1246.6	3.7

First, very few cuts are obtained with the polar normalization, to the exception of `tls` and `uflquad` instances. As mentioned earlier, this is the result of premature termination of the cut generation due to numerical issues in the CGCP. Nevertheless, the obtained cuts are relatively sparse, with only 6.2% and 6.4% non-zeros after 10 and 200 rounds, respectively.

Second, to the exception of `sssd` instances, the α normalization always yields denser cuts than other normalizations, with an average of 29.7% and 26.1% non-zeros coefficients after 10 and 200 rounds, respectively. Denser cuts have an adverse effect on performance, as they slow down the resolution of the current linear relaxation, and are more prone to numerical errors. Overall, few $\mathcal{K}^*$ cuts are obtained, except for `fmo` and `tls` instances.

Third, fewer $\mathcal{K}^*$ cuts are obtained with the uniform normalization than with the standard normalization, which is not surprising given the remarks in Section 5.5. Furthermore, cuts obtained with the uniform normalization are slightly sparser, with 1.9% and 3.7% non-zero coefficients after 10 and 200 rounds for the uniform normalization, compared to 2.5% and 4.7% for the standard normalization. Interestingly, the average cut density after 200 rounds is almost twice as large as after 10 rounds, indicating that cuts become denser in the later rounds.

5.8 Conclusion

Motivated by the impact of the disjunctive framework in MILP and the recent success of conic formulations for MI-CONV, we have investigated the computational aspects of disjunctive cuts in MI-CONIC. Building on conic duality, we have extended Balas' cut-generating linear program into a cut-generating conic program, and studied the fundamental role of the normalization condition in its resolution. In doing so, we have answered several relevant questions, especially from the numerical standpoint, left open by previous developments in the area, and have raised new ones.

5.8.1 What we have learned...

From a theoretical standpoint, we have shown that the normalization condition in the CGCP impacts not only the theoretical properties of the obtained cuts, but also whether the CGCP is solvable in the first place. The latter has direct consequences for the numerical robustness of the CGCP's resolution. In particular, we have introduced conic normalizations that guarantee conic strong duality, without any assumptions on the well-posedness of the considered disjunctions. Furthermore, we have identified the risk of generating $\mathcal{K}^*$ cuts when separating conic-infeasible points, and suggested several strategies to alleviate it. Finally, we have proposed extensions of lifting and cut strengthening to the conic setting, with the potential of further computational benefits.

From a computational standpoint, we have investigated the practical behavior of several normalization conditions, on a diverse set of instances. These experiments indicate that our CGCP-based cut separation is competitive with a state-of-the-art MI-CONIC solver, being able to close more gap in the early stages. In particular, the numerical robustness of the standard and uniform normalization translates in faster separation and more gap closed. Finally, carefully managing the outer approximation, so as to avoid large violations of conic constraints, appears critical to the cut generation's performance.

5.8.2 … and what lies ahead

Several theoretical questions are left open. First, while we only consider linear cutting planes, whether conic cuts can be separated efficiently in the general case, and how to best integrate them within existing MI-CONIC algorithms, remains an open question. Second, dominance relations between valid inequalities were introduced in, e.g, [126, 160]. Characterizing optimal solutions of the normalized CGCP in that perspective may offer further insight on the importance of normalization. Third, a unifying framework that generalizes monoidal strengthening to the MI-CONIC setting could further improve the practical effectiveness of disjunctive cuts. Recent developments in duality theory and cut-generating functions for MI-CONIC could offer the tools for doing so.

Computational experience with cuts in MI-CONIC remains scarce, and further research in that direction is needed. Decades of experience in MILP indicate that classes of disjunctive cuts with fast separation rules, e.g., Gmomory Mixed-Integer cuts and Mixed-Integer Rounding cuts, yield the greatest computational benefits. Similar strategies for MI-CONIC would undoubtedly be beneficial. In a similar fashion, Fischetti et al. [136] have shown that scaling impacts the cuts yielded by the CGLP with standard normalization: there is no indication that MI-CONIC should be any different. Finally, the CGCP form allows to experiment with a number of algorithmic techniques for separating disjunctive cuts, whose practical effectiveness will most likely be problem-specific.

CHAPTER 6 GENERAL DISCUSSION

The three works presented in Chapters 3, 4, and 5 explore various ways of exploiting a problem's structure to improve a solver performance. Although each strategy operates at a different level of the optimization pipeline, from mathematical modeling down to linear algebra implementations and cut-separation techniques, they share a number of motivations and insights. This further discussion aims to highlight such connections.

The problems and solution techniques investigated in Chapter 3 lay the groundwork for Chapters 4 and 5. In Section 3.5.3, we observed that, as the number of DERs increases, a growing proportion of time is spent solving the Restricted Master Problem. This motivated us to investigate structured IPMs. Doing so required the development of the relevant linear algebra routines, their integration within an IPM solver and, finally, interfacing said IPM solver with a column-generation algorithm. These requirements justified the development of the modular IPM solver described in Chapter 4, whose performance was demonstrated on generic LP instances, as well as on the same family of instances as those of Chapter 3 –with and without specialized linear algebra.

In Chapter 3, we focused on developing a generic decomposition framework, that can integrate resources of heterogeneous nature. Consequently, the mathematical models presented in Chapter 3 can be enhanced in several ways. For instance, a battery's operation is more accurately described by taking into account the interplay between state-of-charge and charging/discharging rates and efficiencies. Such models have been investigated, for instance, in [166]. Another direction is to explicitly consider the operational constraints of the power grid. It is common practice to consider conic relaxations of such constraints, e.g. SOCP or SDP relaxations. This yields MI-CONIC problems, which are the focus of Chapter 5.

Numerical stability is a core component of Chapters 4 and 5. It relates to the rounding errors that are intrinsic to finite-precision arithmetic, badly-scaled problems, or poor modeling. Early versions of the cut-separation procedure proposed in Chapter 5 displayed erratic numerical behavior, with numerous instances encountering slow progress or numerical failures. The hands-on experience of developing an interior-point solver in Chapter 4 was key to realizing that those numerical failures were not due to rounding errors, but to a lack of constraint qualification (see, e.g., Theorem 10 in Section 5.4.4). Importantly, the latter can be alleviated by a careful choice of the normalization condition, as demonstrated by the computational results of Section 5.7. Similarly, the regularized homogeneous algorithm described in Section 4.3 significantly improved Tulip's numerical robustness.

CHAPTER 7 CONCLUSION AND RECOMMENDATIONS

Throughout this thesis, we have investigated the use of structure-exploiting techniques within generic MIP algorithmic frameworks, and demonstrated the associated computational benefits. In this final chapter, we present a brief summary of our work, then highlight some limitations and identify several promising avenues for future research on the topic.

7.1 Summary of Works

We have shown in Chapter 3 that the MIP paradigm allows to model the operation of numerous and heterogeneous Distributed Energy Resources. Then, informed by a power grid's structure, we have proposed a decomposition scheme and column-generation algorithm to solve the resulting MIP problem, thereby addressing a number of scalability and privacy concerns. Extensive computational experiments confirmed the approach's effectiveness.

This initial work has led us to consider structure-exploiting techniques within the linear algebra of an interior-point algorithm. Thus, in Chapter 4, we introduced the design and implementation of Tulip, a modular open-source interior-point solver for linear optimization. Tulip's codebase leverages several features of the Julia programming language, to allow for an easy customization of data structures and linear algebra implementations, while building on a robust algorithmic framework. We have demonstrated the flexibility and efficiency of the code on both generic and structured LP instances. For the former, we achieve competitive results against open-source interior-point solvers. For the latter, the integration of specialized routines yield a tenfold speedup over generic ones, thereby outperforming state-of-the-art commercial solvers on structured master problems that arise in DW decomposition.

Finally, motivated by the popularity of MI-CONIC formulations in power systems, we have investigated the computational aspects of disjunctive cuts in MI-CONIC. Building on conic duality, we have proposed a numerically robust and computationally efficient framework for separating such cuts, through the resolution of a cut-generating conic program. In particular, we have characterized the theoretical pros and cons of various normalization conditions, investigated the separation of conic-infeasible points in an outer-approximation context, and proposed conic extensions of classical lifting and strengthening procedure. Finally, computational experiments on several classes of instances have demonstrated the effectiveness of the approach, which compares favorably to CPLEX internal lift-and-project cuts, in terms of both gap closed and computing times.

7.2 Limitations and future research

7.2.1 Towards open and modular ecosystems for optimization

The works presented in Chapters 3-5 have focused on the design of structure-exploiting algorithms, and their implementation. However, as mentioned in Chapter 1, the applicability of these techniques also rests on the ability for users to convey structure information. There exists a breadth of practical applications for MIP and, in turn, endless forms of structure-exploiting techniques. Thus, there is little chance that any software managed by a single entity can integrate them all: the corresponding flexibility requirements, software development and additional maintenance would quickly become intractable.

Instead, this need is best addressed by open-source, modular optimization software, from algebraic modeling languages to optimization solvers. This allows users to extend and customize a solver's functionalities, without relying on a centralized development team for support and maintenance, as these services can be provided by the users themselves. A successful example of this paradigm is the academic solver SCIP [20] and corresponding plugins: GCG [21] can be used for DW and Benders decompositions, and SCIP-SDP [167] enables the resolution of Mixed-Integer Semi-Definite Programming problems.

Further software development work is needed, to provide convenient structured modeling tools and interface them with structure-exploiting solvers. Given the relatively young age of open-source modeling languages, compared to decades-old solvers such as CPLEX, it is not entirely clear what a good approach for doing so looks like, and several tools will likely co-exist. In particular, the existing Julia ecosystem for optimization, which includes the algebraic modeling languages JuMP and `Convex.jl`, the solver-agnostic interface `MathOptInterface`, and a number of open-source optimization solvers, represents a promising environment for future developments in this area.

7.2.2 Integrating structure-exploiting techniques and Machine Learning

Recently, a number of authors have investigated the use of Machine Learning techniques within optimization algorithms, see, e.g., [168–171]. Given a distribution of instances, this line of research leverages statistical patterns to better inform the heuristic nature of MIP solvers [7], thereby improving the overall performance of MIP algorithms. This approach, and the one presented in this thesis, complement each other in two ways.

On the one hand, while ML techniques exploit statistical patterns in a given distribution of instances, our approach makes use of mathematically well-defined structure in a principled way. Thus, the additional tools we propose do not require re-training, and can themselves be enhanced with ML-based tools. For instance, in [172], the authors devise a supervised learning approach to help automate the decomposition process.

On the other hand, recent experience suggests that exploiting a problem's structure also improves the efficiency of ML techniques [169, 171, 173]. Typically, this comes in the form of more suitable model priors and meaningful features. For instance, Graph Neural Networks have proved to be an effective model prior for MIP problems [169], while, in [173], the authors achieve greater generalization in learned branching strategies, by explicitly taking into account the structure of a branch-and-bound tree. Reciprocally, ML-based approaches may help identify new problem structures and specialized algorithms.

REFERENCES

[1] R. E. Bixby, “Solving Real-World Linear Programs: A Decade and More of Progress,” *Operations Research*, vol. 50, no. 1, pp. 3–15, Feb. 2002. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/opre.50.1.3.17780

[2] A. Lodi, “Mixed Integer Programming Computation,” M. Jünger *et al.*, Eds. Berlin, Heidelberg: Springer, 2010, pp. 619–645.

[3] R. E. Bixby, “A brief history of linear and mixed-integer programming computation,” *Documenta Mathematica*, pp. 107–121, 2012.

[4] D. Applegate *et al.*, “Concorde tsp solver,” 2006.

[5] D. Juhl *et al.*, “The geosteiner software package for computing steiner trees in the plane: an updated computational study,” *Mathematical Programming Computation*, vol. 10, no. 4, pp. 487–532, 2018.

[6] A. H. Land and A. G. Doig, “An Automatic Method of Solving Discrete Programming Problems,” *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.

[7] A. Lodi, “The Heuristic (Dark) Side of MIP Solvers,” ser. Studies in Computational Intelligence, E.-G. Talbi, Ed. Berlin, Heidelberg: Springer, 2013, pp. 273–284. [Online]. Available: https://doi.org/10.1007/978-3-642-30671-6_10

[8] T. Achterberg *et al.*, “Presolve reductions in mixed integer programming,” *INFORMS Journal on Computing*, vol. 32, no. 2, pp. 473–506, 2020.

[9] M. Fischetti and A. Lodi, “Heuristics in Mixed Integer Programming,” in *Wiley Encyclopedia of Operations Research and Management Science.* Wiley Online Library, 2011.

[10] C. Bingane, M. F. Anjos, and S. Le Digabel, “Tight-and-cheap conic relaxation for the ac optimal power flow problem,” *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 7181–7188, Nov 2018.

[11] P. Bonami, D. Salvagnin, and A. Tramontani, “Implementing Automatic Benders Decomposition in a Modern MIP Solver,” in *Integer Programming and Combinatorial Optimization*, ser. Lecture Notes in Computer Science, D. Bienstock and G. Zambelli, Eds. Cham: Springer International Publishing, 2020, pp. 78–90.

[12] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.

[13] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Operations Research*, vol. 8, no. 1, pp. 101–111, 1960. [Online]. Available: https://doi.org/10.1287/opre.8.1.101

[14] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische Mathematik*, vol. 4, no. 1, pp. 238–252, Dec 1962.

[15] G. Desaulniers, J. Desrosiers, and M. M. Solomon, *Column generation*, 1st ed., ser. GERAD 25th anniversary. Springer Science & Business Media, 2006, vol. 5.

[16] R. Rahmaniani *et al.*, "The benders decomposition algorithm: A literature review," *European Journal of Operational Research*, vol. 259, no. 3, pp. 801 – 817, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221716310244

[17] IBM, "IBM ILOG CPLEX Optimization Studio." [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio

[18] M. Galati, "Decomposition in integer linear programming," Ph.D. dissertation, PhD thesis, Lehigh University, 2009.

[19] T. Ralphs *et al.*, "coin-or/dip: Version 0.92.3," Jan. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.246087

[20] G. Gamrath *et al.*, "The SCIP Optimization Suite 7.0," Optimization Online, Technical Report, March 2020. [Online]. Available: http://www.optimization-online.org/DB_HTML/2020/03/7705.html

[21] G. Gamrath and M. E. Lübbecke, "Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs," in *Experimental Algorithms*, ser. Lecture Notes in Computer Science, P. Festa, Ed. Berlin, Heidelberg: Springer, 2010, pp. 239–252.

[22] F. Vanderbeck, R. Sadykov, and I. Tahiri, "Bapcod–a generic branch-and-price code," 2005. [Online]. Available: https://realopt.bordeaux.inria.fr/?page_id=2

[23] "Coluna.jl." [Online]. Available: https://github.com/atoptima/Coluna.jl

[24] K. Kim and V. M. Zavala, "Algorithmic innovations and software for the dual decomposition method applied to stochastic mixed-integer programs," *Mathematical Programming Computation*, vol. 10, no. 2, pp. 225–266, Jun 2018.

[25] A. Grothey *et al.*, “A Structure Conveying Parallelizable Modeling Language for Mathematical Programming,” ser. Springer Optimization and Its Applications, R. Ciegis *et al.*, Eds. New York, NY: Springer, 2009, pp. 145–156. [Online]. Available: https://doi.org/10.1007/978-0-387-09707-7_13

[26] M. F. Anjos, A. Lodi, and M. Tanneau, “A decentralized framework for the optimal coordination of distributed energy resources,” *IEEE Transactions on Power Systems*, vol. 34, no. 1, pp. 349–359, Jan 2019.

[27] M. H. Albadi and E. F. El-Saadany, “A summary of demand response in electricity markets,” *Electric Power Systems Research*, vol. 78, no. 11, pp. 1989–1996, 2008.

[28] IESO, “Distributed energy resources,” accessed: 03-09-2018. [Online]. Available: http://www.ieso.ca/en/learn/ontario-power-system/a-smarter-grid/distributed-energy-resources

[29] L. Gkatzikis, I. Koutsopoulos, and T. Salonidis, “The role of aggregators in smart grid demand response markets,” *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 7, pp. 1247–1257, 2013.

[30] P. Siano, “Demand response and smart grids - a survey,” *Renewable and Sustainable Energy Reviews*, vol. 30, no. Supplement C, pp. 461 – 478, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032113007211

[31] R. Deng *et al.*, “A survey on demand response in smart grids: Mathematical models and approaches,” *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 570–582, June 2015.

[32] J. S. Vardakas, N. Zorba, and C. V. Verikoukis, “A survey on demand response programs in smart grids: Pricing methods and optimization algorithms,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 152–178, 2015.

[33] M. Zugno *et al.*, “A bilevel model for electricity retailers’ participation in a demand response market environment,” *Energy Economics*, vol. 36, pp. 182 – 197, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140988312003477

[34] P. Yang, G. Tang, and A. Nehorai, “A game-theoretic approach for optimal time-of-use electricity pricing,” *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 884–892, May 2013.

[35] D. T. Nguyen, H. T. Nguyen, and L. B. Le, "Dynamic pricing design for demand response integration in power distribution networks," *IEEE Transactions on Power Systems*, vol. 31, no. 5, pp. 3457–3472, Sept 2016.

[36] E. Alekseeva *et al.*, "Bilevel approach to optimize electricity prices," *Yugoslav Journal of Operations Research*, 2018.

[37] A. H. Mohsenian-Rad *et al.*, "Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid," *IEEE Transactions on Smart Grid*, vol. 1, no. 3, pp. 320–331, Dec 2010.

[38] Z. Baharlouei, H. Narimani, and H. Mohsenian-Rad, "Tackling co-existence and fairness challenges in autonomous demand side management," in *2012 IEEE Global Communications Conference (GLOBECOM)*, Dec 2012, pp. 3159–3164.

[39] L. Chen *et al.*, "Two market models for demand response in power networks," in *2010 First IEEE International Conference on Smart Grid Communications*, Oct 2010, pp. 397–402.

[40] N. Li, L. Chen, and S. H. Low, "Optimal demand response based on utility maximization in power networks," in *2011 IEEE Power and Energy Society General Meeting*, July 2011, pp. 1–8.

[41] H. Zhong, L. Xie, and Q. Xia, "Coupon incentive-based demand response: Theory and case study," *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 1266–1276, May 2013.

[42] J. Li *et al.*, "Aggregator service for pv and battery energy storage systems of residential building," *CSEE Journal of Power and Energy Systems*, vol. 1, no. 4, pp. 3–11, 2015.

[43] M. Parvania, M. Fotuhi-Firuzabad, and M. Shahidehpour, "Optimal demand response aggregation in wholesale electricity markets," *IEEE Transactions on Smart Grid*, vol. 4, no. 4, pp. 1957–1965, 2013.

[44] ——, "Iso's optimal strategies for scheduling the hourly demand response in day-ahead markets," *IEEE Transactions on Power Systems*, vol. 29, no. 6, pp. 2636–2645, 2014.

[45] Z. Zhu *et al.*, "An integer linear programming based optimization for home demand-side management in smart grid," in *2012 IEEE PES Innovative Smart Grid Technologies (ISGT)*, Jan 2012, pp. 1–5.

[46] P. Chavali, P. Yang, and A. Nehorai, “A distributed algorithm of appliance scheduling for home energy management system,” *IEEE Transactions on Smart Grid*, vol. 5, no. 1, pp. 282–290, Jan 2014.

[47] T. Logenthiran, D. Srinivasan, and T. Z. Shun, “Demand side management in smart grid using heuristic optimization,” *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1244–1252, Sept 2012.

[48] K. Paridari *et al.*, “Demand response for aggregated residential consumers with energy storage sharing,” in *2015 54th IEEE Conference on Decision and Control (CDC)*, Dec 2015, pp. 2024–2030.

[49] M. A. A. Pedrasa, T. D. Spooner, and I. F. MacGill, “Coordinated scheduling of residential distributed energy resources to optimize smart home energy services,” *IEEE Transactions on Smart Grid*, vol. 1, no. 2, pp. 134–143, Sept 2010.

[50] M. Kraning *et al.*, “Dynamic network energy management via proximal message passing,” *Foundations and Trends® in Optimization*, vol. 1, no. 2, pp. 73–126, 2014.

[51] J. Rivera *et al.*, “Alternating direction method of multipliers for decentralized electric vehicle charging control,” in *52nd IEEE Conference on Decision and Control*, Dec 2013, pp. 6960–6965.

[52] N. Gatsis and G. B. Giannakis, “Residential demand response with interruptible tasks: Duality and algorithms,” in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, Dec 2011, pp. 1–6.

[53] ——, “Decomposition algorithms for market clearing with large-scale demand response,” *IEEE Transactions on Smart Grid*, vol. 4, no. 4, pp. 1976–1987, Dec 2013.

[54] S. J. Kim and G. B. Giannakis, “Scalable and robust demand response with mixed-integer constraints,” *IEEE Transactions on Smart Grid*, vol. 4, no. 4, pp. 2089–2099, Dec 2013.

[55] S. Mhanna, A. C. Chapman, and G. Verbic, “A fast distributed algorithm for large-scale demand response aggregation,” *IEEE Transactions on Smart Grid*, vol. 7, no. 4, pp. 2094–2107, July 2016.

[56] P. M. Namara and S. McLoone, “Hierarchical demand response using dantzig-wolfe decomposition,” in *IEEE PES ISGT Europe 2013*, Oct 2013, pp. 1–5.

[57] H. A. Toersche *et al.*, “Column generation based planning in smart grids using triana,” in *IEEE PES ISGT Europe 2013*, Oct 2013, pp. 1–5.

[58] M. H. Amini *et al.*, “Hierarchical electric vehicle charging aggregator strategy using dantzig-wolfe decomposition,” *IEEE Design Test*, vol. PP, no. 99, pp. 1–1, 2017.

[59] H. Harb *et al.*, “Decentralized scheduling strategy of heating systems for balancing the residual load,” *Building and Environment*, vol. 86, pp. 132 – 140, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360132314004260

[60] M. Beaudin and H. Zareipour, “Home energy management systems: A review of modelling and complexity,” *Renewable and Sustainable Energy Reviews*, vol. 45, pp. 318 – 335, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032115000568

[61] G. T. Costanzo *et al.*, “A system architecture for autonomous demand side load management in smart buildings,” *IEEE Transactions on Smart Grid*, vol. 3, no. 4, pp. 2157–2165, Dec 2012.

[62] J. P. Holdman, *Heat Transfer*, 10th ed. McGraw-HIll, 1997.

[63] J. A. Gomez and M. F. Anjos, “Collaborative demand-response planner for smart buildings,” GERAD, Tech. Rep. Technical Report G-2017-39, 2017. [Online]. Available: https://www.gerad.ca/en/papers/G-2017-15/view

[64] IESO, “Power data.” [Online]. Available: http://www.ieso.ca/power-data

[65] “Historical data,” toronto City station, ID 6158355. [Online]. Available: http://climate.weather.gc.ca/historical_data/search_historic_data_e.html

[66] Tesla, “Powerwall specifications,” accessed: 08-10-2017. [Online]. Available: https://www.tesla.com/support/powerwall

[67] M. F. Anjos, A. Lodi, and M. Tanneau, “Design and implementation of a modular interior-point solver for linear optimization,” *arXiv:2006.08814 [math]*, Jun. 2020. [Online]. Available: http://arxiv.org/abs/2006.08814

[68] J. Kelley, Jr., “The cutting-plane method for solving convex programs,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, 1960.

[69] T. Westerlund and F. Pettersson, “An extended cutting plane method for solving convex minlp problems,” *Computers & Chemical Engineering*, vol. 19, pp. 131 – 136, 1995.

[70] J. E. Mitchell, "Cutting plane methods and subgradient methods," in *Decision Technologies and Applications*, ch. 2, pp. 34–61.

[71] S. Wright, *Primal-Dual Interior-Point Methods.* Society for Industrial and Applied Mathematics, 1997. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611971453

[72] J. Gondzio, "Interior point methods 25 years later," *European Journal of Operational Research*, vol. 218, no. 3, pp. 587 – 601, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221711008204

[73] R. E. Bixby *et al.*, "Very large-scale linear programming: A case study in combining interior point and simplex methods," *Operations Research*, vol. 40, no. 5, pp. 885–897, 1992.

[74] J. E. Mitchell and B. Borchers, *Solving Linear Ordering Problems with a Combined Interior Point/Simplex Cutting Plane Algorithm.* Boston, MA: Springer US, 2000, pp. 349–366.

[75] S. Elhedhli and J.-L. Goffin, "The integration of an interior-point cutting plane method within a branch-and-price algorithm," *Mathematical programming*, vol. 100, no. 2, pp. 267–294, 2004.

[76] F. Babonneau and J.-P. Vial, "Accpm with a nonlinear constraint and an active set strategy to solve nonlinear multicommodity flow problems," *Mathematical programming*, vol. 120, no. 1, pp. 179–210, 2009.

[77] J. Naoum-Sawaya and S. Elhedhli, "An interior-point benders based branch-and-cut algorithm for mixed integer programs," *Annals of Operations Research*, vol. 210, no. 1, pp. 33–55, 2013.

[78] P. Munari and J. Gondzio, "Using the primal-dual interior point algorithm within the branch-price-and-cut method," *Computers & Operations Research*, vol. 40, no. 8, pp. 2026 – 2036, 2013.

[79] J. Gondzio, P. González-Brevis, and P. Munari, "Large-scale optimization with the primal-dual column generation method," *Mathematical Programming Computation*, vol. 8, no. 1, pp. 47–82, Mar 2016.

[80] J. Gondzio and R. Sarkissian, "Column generation with a primal-dual method," Technical report 96.6, Logilab, Tech. Rep., 1996. [Online]. Available: http://www.maths.ed.ac.uk/~gondzio/reports/pdcgm.pdf

[81] L.-M. Rousseau, M. Gendreau, and D. Feillet, "Interior point stabilization for column generation," *Operations Research Letters*, vol. 35, no. 5, pp. 660 – 668, 2007.

[82] J. Gondzio, P. Gonzalez-Brevis, and P. Munari, "New developments in the primal-dual column generation technique," *European Journal of Operational Research*, vol. 224, no. 1, pp. 41 – 51, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221712005656

[83] J. R. Birge and L. Qi, "Computing block-angular karmarkar projections with applications to stochastic programming," *Management Science*, vol. 34, no. 12, pp. 1472–1479, 1988.

[84] J. K. Hurd and F. H. Murphy, "Exploiting special structure in primal dual interior point methods," *ORSA Journal on Computing*, vol. 4, no. 1, pp. 38–44, 1992.

[85] G. L. Schultz and R. R. Meyer, "An interior point method for block angular optimization," *SIAM Journal on Optimization*, vol. 1, no. 4, pp. 583–602, 1991.

[86] I. C. Choi and D. Goldfarb, "Exploiting special structure in a primal—dual path-following algorithm," *Mathematical Programming*, vol. 58, no. 1, pp. 33–52, Jan 1993.

[87] E. R. Jessup, D. Yang, and S. A. Zenios, "Parallel factorization of structured matrices arising in stochastic programming," *SIAM Journal on Optimization*, vol. 4, no. 4, pp. 833–846, 1994.

[88] J. Gondzio, R. Sarkissian, and J.-P. Vial, "Using an interior point method for the master problem in a decomposition approach," *European Journal of Operational Research*, vol. 101, no. 3, pp. 577 – 587, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221796001828

[89] J. Gondzio and R. Sarkissian, "Parallel interior-point solver for structured linear programs," *Mathematical Programming*, vol. 96, no. 3, pp. 561–584, Jun 2003. [Online]. Available: https://doi.org/10.1007/s10107-003-0379-5

[90] J. Gondzio and A. Grothey, "Parallel interior-point solver for structured quadratic programs: Application to financial planning problems," *Annals of Operations Research*, vol. 152, no. 1, pp. 319–339, Jul 2007. [Online]. Available: https://doi.org/10.1007/s10479-006-0139-z

[91] J. Castro, S. Nasini, and F. Saldanha-da Gama, "A cutting-plane approach for large-scale capacitated multi-period facility location using a specialized interior-point method," *Mathematical Programming*, vol. 163, no. 1, pp. 411–444, May 2017.

[92] E. M. Gertz and S. J. Wright, "Object-oriented software for quadratic programming," *ACM Trans. Math. Softw.*, vol. 29, no. 1, pp. 58–81, Mar. 2003.

[93] M. Lubin *et al.*, "Scalable stochastic optimization of complex energy systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 64:1–64:64.

[94] J. Gondzio and A. Grothey, "Exploiting structure in parallel implementation of interior point methods for optimization," *Computational Management Science*, vol. 6, no. 2, pp. 135–160, May 2009. [Online]. Available: https://doi.org/10.1007/s10287-008-0090-3

[95] J. Castro, "Interior-point solver for convex separable block-angular problems," *Optimization Methods and Software*, vol. 31, no. 1, pp. 88–109, 2016.

[96] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in python," *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011.

[97] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[98] J. Löfberg, "Yalmip : A toolbox for modeling and optimization in matlab," in *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[99] J. Bezanson *et al.*, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[100] T. A. Davis, "SuiteSparse: A suite of sparse matrix software." [Online]. Available: http://faculty.cse.tamu.edu/davis/suitesparse.html

[101] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992.

[102] M. Tanneau, "Tulip.jl." [Online]. Available: https://github.com/ds4dm/Tulip.jl

[103] B. Legat *et al.*, "Mathoptinterface: a data structure for mathematical optimization problems," 2020.

[104] X. Xu, P.-F. Hung, and Y. Ye, "A simplified homogeneous and self-dual linear programming algorithm and its implementation," *Annals of Operations Research*, vol. 62, no. 1, pp. 151–171, Dec 1996. [Online]. Available: https://doi.org/10.1007/BF02206815

[105] E. D. Andersen and K. D. Andersen, *The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm.* Boston, MA: Springer US, 2000, pp. 197–232.

[106] M. P. Friedlander and D. Orban, “A primal–dual regularized interior-point method for convex quadratic programs,” *Mathematical Programming Computation*, vol. 4, no. 1, pp. 71–107, Mar. 2012.

[107] J. Gondzio, “Multiple centrality corrections in a primal-dual method for linear programming,” *Computational Optimization and Applications*, vol. 6, no. 2, pp. 137–156, Sep 1996. [Online]. Available: https://doi.org/10.1007/BF00249643

[108] E. D. Andersen and K. D. Andersen, “Presolving in linear programming,” *Mathematical Programming*, vol. 71, no. 2, pp. 221–245, Dec 1995.

[109] J. Gondzio, “Presolve analysis of linear programs prior to applying an interior point method,” *INFORMS Journal on Computing*, vol. 9, no. 1, pp. 73–91, 1997.

[110] M. Udell *et al.*, “Convex optimization in Julia,” in *Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages.* IEEE Press, 2014, pp. 18–28.

[111] M. Anjos and S. Burer, “On handling free variables in interior-point methods for conic linear optimization,” *SIAM Journal on Optimization*, vol. 18, no. 4, pp. 1310–1325, 2008.

[112] D. Orban, “LDLFactorizations.jl,” Feb. 2019. [Online]. Available: https://github.com/JuliaSmoothOptimizers/LDLFactorizations.jl

[113] “CLP.” [Online]. Available: https://projects.coin-or.org/Clp

[114] “GNU Linear Programming Kit.” [Online]. Available: https://www.gnu.org/software/glpk/

[115] A. Domahidi, E. Chu, and S. Boyd, “ECOS: An SOCP solver for embedded systems,” in *European Control Conference (ECC)*, 2013, pp. 3071–3076.

[116] G. O. LLC, “Gurobi optimizer reference manual,” 2018. [Online]. Available: https://www.gurobi.com

[117] MOSEK ApS, “The MOSEK Optimization Suite.” [Online]. Available: https://www.mosek.com/

[118] D. Ma and M. A. Saunders, “Solving multiscale linear programs using the simplex method in quadruple precision,” in *Numerical Analysis and Optimization*, M. Al-Baali, L. Grandinetti, and A. Purnama, Eds. Cham: Springer International Publishing, 2015, pp. 223–235.

[119] A. M. Gleixner, D. E. Steffy, and K. Wolter, “Iterative refinement for linear programming,” *INFORMS Journal on Computing*, vol. 28, no. 3, pp. 449–464, 2016.

[120] A. Lodi, M. Tanneau, and J. P. Vielma, “Disjunctive cuts in Mixed-Integer Conic Optimization,” *arXiv:1912.03166 [math]*, Sep. 2020. [Online]. Available: http://arxiv.org/abs/1912.03166

[121] P. Bonami *et al.*, “An algorithmic framework for convex mixed integer nonlinear programs,” *Discrete Optimization*, vol. 5, no. 2, pp. 186–204, 2008.

[122] A. M. Geoffrion, “Elements of large-scale mathematical programming,” *Management Science*, vol. 16, no. 11, pp. 676–691, 1970.

[123] M. A. Duran and I. E. Grossmann, “An outer-approximation algorithm for a class of mixed-integer nonlinear programs,” *Mathematical Programming*, vol. 36, no. 3, pp. 307–339, Oct 1986.

[124] J. Kronqvist *et al.*, “A review and comparison of solvers for convex minlp,” *Optimization and Engineering*, vol. 20, no. 2, pp. 397–455, Jun 2019.

[125] A. Ben-Tal and A. Nemirovski, *Lectures on Modern Convex Optimization.* Society for Industrial and Applied Mathematics, 2001.

[126] F. Kilinç-Karzan, “On Minimal Valid Inequalities for Mixed Integer Conic Programs,” *Mathematics of Operations Research*, vol. 41, no. 2, pp. 477–510, Sep. 2015.

[127] M. Grant, S. Boyd, and Y. Ye, “Disciplined convex programming,” in *Global Optimization*, ser. Nonconvex Optimization and Its Applications, L. Liberti and N. Maculan, Eds. Springer US, 2006, vol. 84, pp. 155–210.

[128] MOSEK ApS, “Mosek modeling cookbook,” 2020. [Online]. Available: https://docs.mosek.com/modeling-cookbook/index.html

[129] M. Lubin *et al.*, “Polyhedral approximation in mixed-integer convex optimization,” *Mathematical Programming*, vol. 172, no. 1, pp. 139–168, Nov 2018.

[130] C. Coey, M. Lubin, and J. P. Vielma, "Outer approximation with conic certificates for mixed-integer convex problems," *arXiv preprint arXiv:1808.05290*, 2018.

[131] M. Lubin *et al.*, "Extended Formulations in Mixed-Integer Convex Programming," in *Proceedings of the 18th Conference on Integer Programming and Combinatorial Optimization (IPCO 2016)*, ser. Lecture Notes in Computer Science, Q. Louveaux and M. Skutella, Eds., vol. 9682, 2016, pp. 102–113.

[132] E. Balas, "Disjunctive programming," in *Discrete Optimization II*, ser. Annals of Discrete Mathematics, P. Hammer, E. Johnson, and B. Korte, Eds. Elsevier, 1979, vol. 5, pp. 3 – 51.

[133] E. Balas, S. Ceria, and G. Cornuéjols, "A lift-and-project cutting plane algorithm for mixed 0–1 programs," *Mathematical Programming*, vol. 58, no. 1, pp. 295–324, Jan 1993.

[134] E. Balas, S. Ceria, and G. Cornuéjols, "Mixed 0-1 Programming by Lift-and-Project in a Branch-and-Cut Framework," *Management Science*, Sep. 1996.

[135] E. Balas and R. G. Jeroslow, "Strengthening cuts for mixed integer programs," *European Journal of Operational Research*, vol. 4, no. 4, pp. 224–234, Apr. 1980.

[136] M. Fischetti, A. Lodi, and A. Tramontani, "On the separation of disjunctive cuts," *Mathematical Programming*, vol. 128, no. 1, pp. 205–230, Jun 2011.

[137] P. Bonami, "On optimizing over lift-and-project closures," *Mathematical Programming Computation*, vol. 4, no. 2, pp. 151–179, Jun 2012.

[138] F. Cadoux and C. Lemaréchal, "Reflections on generating (disjunctive) cuts," *EURO Journal on Computational Optimization*, vol. 1, no. 1, pp. 51–69, May 2013.

[139] T. Serra, "Reformulating the disjunctive cut generating linear program," *Annals of Operations Research*, Jul. 2020.

[140] M. Conforti and L. A. Wolsey, ""Facet" separation with one linear program," *Mathematical Programming*, vol. 178, no. 1, pp. 361–380, Nov. 2019.

[141] E. Balas and M. Perregaard, "A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer gomory cuts for 0-1 programming," *Mathematical Programming*, vol. 94, no. 2, pp. 221–245, Jan 2003.

[142] ——, "Lift-and-project for mixed 0–1 programming: recent progress," *Discrete Applied Mathematics*, vol. 123, no. 1, pp. 129 – 154, 2002.

[143] A. Kazachkov, "Non-Recursive Cut Generation," Ph.D. dissertation, Carnegie Mellon University, Apr. 2018.

[144] S. Ceria and J. Soares, "Convex programming for disjunctive convex optimization," *Mathematical Programming*, vol. 86, no. 3, pp. 595–614, Dec 1999.

[145] R. A. Stubbs and S. Mehrotra, "A branch-and-cut method for 0-1 mixed convex programming," *Mathematical Programming*, vol. 86, no. 3, pp. 515–532, Dec 1999.

[146] Y. Zhu and T. Kuno, "A disjunctive cutting-plane-based branch-and-cut algorithm for 0- 1 mixed-integer convex nonlinear programs," *Industrial & engineering chemistry research*, vol. 45, no. 1, pp. 187–196, 2006.

[147] P. Bonami, "Lift-and-project cuts for mixed integer convex programs," in *Integer Programming and Combinatoral Optimization*, O. Günlük and G. J. Woeginger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 52–64. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-20807-2_5

[148] M. R. Kılınç, J. Linderoth, and J. Luedtke, "Lift-and-project cuts for convex mixed integer nonlinear programs," *Mathematical Programming Computation*, vol. 9, no. 4, pp. 499–526, Dec 2017.

[149] A. Atamtürk and V. Narayanan, "Conic mixed-integer rounding cuts," *Mathematical Programming*, vol. 122, no. 1, pp. 1–20, Mar 2010.

[150] S. Modaresi, M. R. Kılınç, and J. P. Vielma, "Split cuts and extended formulations for mixed integer conic quadratic programming," *Operations Research Letters*, vol. 43, no. 1, pp. 10 – 15, 2015.

[151] S. Modaresi, "Valid inequalities and reformulation techniques for mixed integer nonlinear programming," Ph.D. dissertation, University of Pittsburgh, 2016.

[152] K. Andersen and A. N. Jensen, "Intersection cuts for mixed integer conic quadratic sets," in *Integer Programming and Combinatorial Optimization*, M. Goemans and J. Correa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 37–48.

[153] P. Belotti *et al.*, "A conic representation of the convex hull of disjunctive sets and conic cuts for integer second order cone optimization," in *Numerical Analysis and Optimization.* Springer, 2015, pp. 1–35.

[154] F. Kılınç-Karzan and S. Yıldız, “Two-term disjunctions on the second-order cone,” *Mathematical Programming*, vol. 154, no. 1-2, pp. 463–491, 2015.

[155] S. Yildiz and G. Cornuéjols, “Disjunctive cuts for cross-sections of the second-order cone,” *Operations Research Letters*, vol. 43, no. 4, pp. 432–437, Jul. 2015.

[156] M. Çezik and G. Iyengar, “Cuts for mixed 0-1 conic programming,” *Mathematical Programming*, vol. 104, no. 1, pp. 179–202, Sep 2005.

[157] A. Atamtürk and V. Narayanan, “Lifting for conic mixed-integer programming,” *Mathematical Programming*, vol. 126, no. 2, pp. 351–363, Feb 2011.

[158] D. Dadush, S. Dey, and J. Vielma, “The split closure of a strictly convex body,” *Operations Research Letters*, vol. 39, no. 2, pp. 121 – 126, 2011.

[159] S. Modaresi, M. R. Kılınç, and J. P. Vielma, “Intersection cuts for nonlinear integer programming: convexification techniques for structured sets,” *Mathematical Programming*, vol. 155, no. 1, pp. 575–611, Jan 2016.

[160] F. Kılınç-Karzan and D. E. Steffy, “On sublinear inequalities for mixed integer conic programs,” *Mathematical Programming*, vol. 159, no. 1, pp. 585–605, Sep 2016.

[161] R. T. Rockafellar, *Convex analysis.* Princeton university press, 1970, vol. 28.

[162] H. A. Friberg, “Facial reduction heuristics and the motivational example of mixed-integer conic optimization,” Tech. Rep., 2016. [Online]. Available: http://www.optimization-online.org/DB_FILE/2016/02/5324.pdf

[163] P. Bonami, J. Linderoth, and A. Lodi, “Disjunctive cuts for mixed integer nonlinear programming problems,” *Prog Combin Optim*, vol. 18, pp. 521–541, 2011.

[164] M. Perregaard and E. Balas, “Generating cuts from multiple-term disjunctions,” in *Integer Programming and Combinatorial Optimization*, K. Aardal and B. Gerards, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 348–360.

[165] H. A. Friberg, “Cblib 2014: a benchmark library for conic mixed-integer and continuous optimization,” *Mathematical Programming Computation*, vol. 8, no. 2, pp. 191–214, Jun 2016.

[166] D. Bienstock *et al.*, “Robust linear control of storage in transmission systems, and extensions to robust network control problems,” in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec. 2017, pp. 799–806.

[167] T. Gally, M. E. Pfetsch, and S. Ulbrich, "A framework for solving mixed-integer semidefinite programs," *Optimization Methods and Software*, vol. 33, no. 3, pp. 594–632, 2018.

[168] A. Lodi and G. Zarpellon, "On learning and branching: a survey," *TOP*, vol. 25, no. 2, pp. 207–236, Jul. 2017.

[169] Y. Bengio, A. Lodi, and A. Prouvost, "Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon," *European Journal of Operational Research*, Aug. 2020.

[170] G. Zarpellon, "Machine learning algorithms in mixed-integer programming," Ph.D. dissertation, Polytechnique Montréal, 2020.

[171] C. D. Hubbs *et al.*, "OR-Gym: A Reinforcement Learning Library for Operations Research Problem," *arXiv:2008.06319 [cs]*, Aug. 2020, arXiv: 2008.06319. [Online]. Available: http://arxiv.org/abs/2008.06319

[172] M. Kruber, M. E. Lübbecke, and A. Parmentier, "Learning when to use a decomposition," in *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems.* Springer, 2017, pp. 202–210.

[173] G. Zarpellon *et al.*, "Parameterizing branch-and-bound search trees to learn branching policies," *arXiv preprint arXiv:2002.05120*, 2020.

APPENDIX A DANTZIG-WOLFE DECOMPOSITION AND COLUMN GENERATION

In this section, we present the Dantzig-Wolfe decomposition principle [13] and the basic column-generation framework. We refer to [15] for a thorough overview of column generation, and the relation between Dantzig-Wolfe decomposition and Lagrangian decomposition.

A.1 Dantzig-Wolfe decomposition

Consider the problem

$$
\begin{aligned}
(P) \quad \min_{x} \quad & \sum_{r=0}^{R} c_r^T x_r \\
s.t. \quad & \sum_{r=0}^{R} A_r x_r = b_0, \\
& x_0 \geq 0, \\
& x_r \in \mathcal{X}_r, \quad r = 1, ..., R,
\end{aligned}
$$

where, for each $r = 1, ..., R$, $\mathcal{X}_r$ is defined by a finite number of linear inequalities, plus integrality restrictions on some of the coordinates of x_r. Therefore, the convex hull of $\mathcal{X}_r$, denoted by $conv(\mathcal{X}_r)$, is a polyhedron whose set of extreme points (resp. extreme rays) is denoted by Ω_r (resp. Γ_r). Any element of $conv(\mathcal{X}_r)$ can thus be written as a convex combination of extreme points $\{\omega\}_{\omega \in \Omega_r}$, plus a non-negative combination of extreme rays $\{\rho\}_{\rho \in \Gamma_r}$ i.e.,

$$
conv(\mathcal{X}_r) = \left\{ \sum_{\omega \in \Omega_r} \lambda_\omega \omega + \sum_{\rho \in \Gamma_r} \lambda_\rho \rho \;\middle|\; \lambda \geq 0, \sum_{\omega} \lambda_\omega = 1 \right\}. \tag{A.1}
$$

The Dantzig-Wolfe decomposition principle [13] then consists in substituting x_r with such a combination of extreme points and extreme rays. This change of variable yields the so-called

Master Problem

$$(MP) \quad \min_{x,\lambda} \quad c_0^T x_0 + \sum_{r=1}^{R} \sum_{\omega \in \Omega_r} c_{r,\omega} \lambda_{r,\omega} + \sum_{r=1}^{R} \sum_{\rho \in \Gamma_r} c_{r,\rho} \lambda_{r,\rho} \tag{A.2}$$

$$s.t. \quad \sum_{\omega \in \Omega_r} \lambda_{r,\omega} = 1, \quad r = 1, ..., R \tag{A.3}$$

$$A_0 x_0 + \sum_{r=1}^{R} \sum_{\omega \in \Omega_r} a_{r,\omega} \lambda_{r,\omega} + \sum_{r=1}^{R} \sum_{\rho \in \Gamma_r} a_{r,\rho} \lambda_{r,\rho} = b_0, \tag{A.4}$$

$$x_0, \lambda \geq 0, \tag{A.5}$$

$$\sum_{\omega \in \Omega_r} \lambda_{r,\omega} \omega + \sum_{\rho \in \Gamma_r} \lambda_{r,\rho} \rho \in \mathcal{X}_r, \quad r = 1, ..., R \tag{A.6}$$

where $c_{r,\omega} = c_r^T \omega$, $c_{r,\rho} = c_r^T \rho$, and $a_{r,\omega} = A_r \omega$, $a_{r,\rho} = A_r \rho$. Constraints (A.3) and (A.4) are referred to as *convexity* and *linking* constraints, respectively.

The linear relaxation of (MP) is given by (A.2)-(A.5); its objective value is greater or equal to that of the linear relaxation of (P) [15]. Note that if (P) in a linear program, i.e., all variables are continuous, then constraints (A.6) are redundant, and (A.2)-(A.5) is equivalent to (P). In the mixed-integer case, problem (A.2)-(A.5) is the root node in a branch-and-price tree. In this work, we focus on solving this linear relaxation. Thus, in what follows, we make a slight abuse of notation and use the term “Master Problem" to refer to (A.2)-(A.5) instead.

A.2 Column generation

The Master Problem has exponentially many variables. Therefore, it is typically solved by column generation, wherein only a small subset of the variables are considered. Additional variables are generated iteratively by solving an auxiliary sub-problem.

Let $\bar{\Omega}_r$ (resp. $\bar{\Gamma}_r$) be a small subset of Ω_r (resp. of Γ_r), and define the *Restricted Master Problem* (RMP)

$$(RMP) \quad \min_{\lambda} \quad c_0^T x_0 + \sum_{r=1}^{R} \sum_{\omega \in \bar{\Omega}_r} c_{r,\omega} \lambda_{r,\omega} + \sum_{r=1}^{R} \sum_{\rho \in \bar{\Gamma}_r} c_{r,\rho} \lambda_{r,\rho} \tag{A.7}$$

$$s.t. \quad \sum_{\omega \in \bar{\Omega}_r} \lambda_{r,\omega} = 1, \quad r = 1, ..., R \tag{A.8}$$

$$\sum_{r=1}^{R} \sum_{\omega \in \bar{\Omega}_r} a_{r,\omega} \lambda_{r,\omega} + \sum_{r=1}^{R} \sum_{\rho \in \bar{\Gamma}_r} a_{r,\rho} \lambda_{r,\rho} = b_0, \tag{A.9}$$

$$x_0, \lambda \geq 0. \tag{A.10}$$

In all that follows, we assume that (RMP) is feasible and bounded. Note that feasibility can be obtained by adding artificial slacks and surplus variables with sufficiently large cost, effectively implementing an l_1 penalty. If the RMP is unbounded, then so is the MP.

Let $\sigma \in \mathbb{R}^R$ and $\pi \in \mathbb{R}^{m_0}$ denote the vector of dual variables associated to convexity constraints (A.8) and linking constraints constraints (A.9), respectively. Here, we assume that (σ, π) is dual-optimal for (RMP); the use of interior, sub-optimal dual solutions is explored in [80]. Then, for given r, $\omega \in \Omega_r$ and $\rho \in \Gamma_r$, the reduced cost of variable $\lambda_{r,\omega}$ is

$$\bar{c}_{r,\omega} = c_{r,\omega} - \pi^T a_{r,\omega} - \sigma_r = (c_r^T - \pi^T A_r)\omega - \sigma_r,$$

while the reduced cost of variable $\lambda_{r,\rho}$ is

$$\bar{c}_{r,\rho} = c_{r,\rho} - \pi^T a_{r,\rho} = (c_r^T - \pi^T A_r)\rho.$$

If $\bar{c}_{r,\omega} \geq 0$ for all r, $\omega \in \Omega_r$ and $\bar{c}_{r,\rho} \geq 0$ for all r, $\rho \in \Gamma_r$, then the current solution is optimal for the MP. Otherwise, a variable with negative reduced cost is added to the RMP. Finding such a variable, or proving that none exists, is called the *pricing step*.

Explicitly iterating through the exponentially large sets Ω_r and Γ_r is prohibitively expensive. Nevertheless, the pricing step can be written as the following MILP:

$$(SP_r) \quad \min_{x_r} \quad (c_r^T - \pi^T A)x_r - \sigma_r \tag{A.11}$$

$$s.t. \quad x_r \in \mathcal{X}_r, \tag{A.12}$$

which we refer to as the r^{th} *sub-problem*. If SP_r is infeasible, then $\mathcal{X}_r$ is empty, and the original problem P is infeasible. This case is ruled out in all that follows. Then, since the objective of SP_r is linear, any optimal solution is either an extreme point $\omega \in \Omega_r$ (bounded case), or an extreme ray $\rho \in \Gamma_r$ (unbounded case). The corresponding variable $\lambda_{r,\omega}$ or $\lambda_{r,\rho}$ is identified by retrieving an optimal point or unbounded ray. Finally, note that all R sub-problems $SP_1, \dots, SP_R$ can be solved independently from one another. Optimality in the Master Problem is attained when no variable with negative reduced cost can be identified from all R sub-problems.

We now describe a basic column-generation procedure, which is formally stated in Algorithm 4. The algorithm starts with an initial RMP that contains a small subset of columns, some of which may be artificial to ensure feasibility. At the beginning of each iteration, the RMP is solved to optimality, and a dual solution (π, σ) is obtained which is used to perform the pricing step. Each sub-problem is solved to identify a variable with most negative reduced cost. If a variable with negative reduced cost is found, it is added to the RMP; if not, the column-generation procedure stops.

```
Input: Initial RMP
 1: while stopping criterion not met do
 2:     Solve RMP and obtain optimal dual variables $(\pi, \sigma)$
 3:     // Pricing step
 4:     for all $r \in \mathcal{R}$ do
 5:         Solve $SP_r$ with the query point $(\pi, \sigma_r)$; obtain $\omega^*$ or $\rho^*$
 6:         if $\bar{c}_{r,\omega^*} < 0$ or $\bar{c}_{r,\rho^*} < 0$ then
 7:             Add corresponding column to the RMP
 8:         end if
 9:     end for
10:     // Stopping criterion
11:     if no column added to RMP then
12:         STOP
13:     end if
14: end while
```

Algorithm 4 Column-generation procedure

For large instances with numerous subproblems, full pricing, wherein all subproblems are solved at each iteration, is often not the most efficient approach. Therefore, we implemented a partial pricing strategy, in which subproblems are solved in a random order until either all subproblems have been solved, or a user-specified number of columns with negative reduced cost have been generated.

APPENDIX B DETAILED RESULTS ON STRUCTURED LP INSTANCES

Table B.1 Structured instances: performance comparison of IPM solvers

			CPLEX		Gurobi		Mosek		Tulip		Tulip*	
Instance	R	CG	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter
DER-24	1024	10	**0.0**	19	0.0	15	0.1	19	0.4	19	0.3	19
DER-24	1024	20	0.1	27	**0.1**	23	0.1	21	0.5	23	0.3	23
DER-24	1024	30	0.1	28	**0.1**	22	0.1	24	0.8	26	0.3	26
DER-24	1024	40	0.2	44	**0.2**	27	0.2	28	1.2	37	0.4	39
DER-24	1024	43	0.2	33	0.2	27	**0.2**	21	1.1	33	0.5	33
DER-24	2048	10	0.2	31	**0.1**	21	0.1	20	0.8	23	0.3	23
DER-24	2048	20	0.3	30	**0.1**	19	0.2	18	1.0	22	0.3	22
DER-24	2048	30	**0.3**	29	0.3	28	0.3	20	1.4	30	0.5	30
DER-24	2048	40	0.4	48	**0.4**	36	0.4	27	2.5	47	0.6	47
DER-24	4096	10	0.4	35	**0.2**	20	0.3	19	1.3	28	0.5	28
DER-24	4096	20	0.8	39	0.4	23	**0.4**	22	2.1	29	0.5	29
DER-24	4096	30	1.3	65	1.0	42	**0.7**	29	5.4	58	0.9	56
DER-24	4096	40	1.1	38	1.1	32	**0.9**	26	5.0	38	0.9	38
DER-24	4096	41	1.0	40	1.0	32	**0.8**	26	4.7	38	1.0	38
DER-24	8192	10	0.9	32	**0.5**	18	0.6	21	2.6	25	0.7	25
DER-24	8192	20	2.0	39	1.1	26	**1.1**	21	5.9	34	1.1	34
DER-24	8192	30	2.9	62	**1.8**	36	2.1	40	12.5	55	1.9	55
DER-24	8192	40	4.3	79	2.7	46	**2.4**	38	19.0	67	2.6	68
DER-24	16384	10	2.5	47	1.3	26	1.7	26	9.0	39	**1.2**	36
DER-24	16384	20	4.1	42	2.1	29	2.5	22	13.8	37	**2.0**	37
DER-24	16384	30	5.4	55	3.4	36	3.6	26	23.1	48	**3.0**	48
DER-24	16384	40	12.4	110	10.2	88	**6.0**	51	57.6	100	6.7	100
DER-24	16384	42	10.8	93	5.3	48	**5.3**	42	49.8	86	5.3	83
DER-24	32768	10	4.6	39	3.3	34	3.5	23	17.9	36	**2.2**	34
DER-24	32768	20	11.0	53	8.8	52	8.0	39	47.4	66	**5.5**	65
DER-24	32768	30	14.5	68	12.3	56	**8.2**	31	96.1	100	11.2	100
DER-24	32768	40	33.9	148	19.4	85	12.3	43	103.6	91	**11.4**	86
DER-48	1024	10	0.1	24	**0.1**	13	0.1	21	0.8	24	0.3	24
DER-48	1024	20	0.2	26	**0.2**	20	0.2	22	1.1	26	0.3	26

(continued)

			CPLEX		Gurobi		Mosek		Tulip		Tulip*	
Instance	R	CG	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter
DER-48	1024	30	0.3	31	**0.2**	16	0.2	22	1.5	32	0.5	32
DER-48	1024	40	0.4	32	0.4	21	**0.3**	22	1.6	30	0.4	30
DER-48	1024	49	0.5	37	0.4	19	**0.3**	21	1.8	28	0.4	28
DER-48	2048	10	0.3	26	0.3	19	**0.3**	20	1.3	24	0.4	25
DER-48	2048	20	0.7	37	0.5	21	0.5	22	2.2	31	**0.4**	31
DER-48	2048	30	0.8	37	0.6	19	0.5	21	2.6	27	**0.5**	27
DER-48	2048	40	1.2	38	0.9	24	0.7	21	4.1	33	**0.7**	33
DER-48	2048	49	1.6	40	1.0	21	**0.8**	25	5.7	37	0.9	37
DER-48	4096	10	0.8	34	0.6	19	0.6	21	3.2	28	**0.4**	28
DER-48	4096	20	1.5	41	1.1	24	1.0	24	5.8	34	**0.8**	34
DER-48	4096	30	1.7	38	1.4	23	1.4	23	8.2	35	**1.0**	35
DER-48	4096	40	3.0	40	2.2	30	1.9	25	9.9	33	**1.4**	33
DER-48	4096	49	4.1	44	2.0	25	2.0	27	14.1	39	**1.7**	39
DER-48	8192	10	2.1	39	1.5	26	1.5	25	8.1	32	**1.1**	32
DER-48	8192	20	3.9	44	1.9	18	2.0	23	12.7	31	**1.5**	31
DER-48	8192	30	7.2	55	2.9	26	2.9	26	20.4	39	**2.2**	39
DER-48	8192	40	7.3	45	4.0	24	3.8	22	25.4	38	**2.4**	38
DER-48	8192	50	9.7	51	4.2	20	4.5	24	37.0	46	**3.4**	47
DER-48	16384	10	5.0	49	2.9	25	3.8	35	22.4	41	**2.3**	41
DER-48	16384	20	7.8	45	5.5	28	5.2	26	31.5	37	**2.8**	37
DER-48	16384	30	14.6	59	7.3	29	6.3	25	53.6	50	**5.0**	48
DER-48	16384	40	16.3	53	9.3	27	8.5	27	64.5	50	**5.2**	45
DER-48	16384	48	22.3	64	9.9	29	9.3	28	89.7	60	**7.4**	57
DER-48	32768	10	10.8	49	7.4	27	8.1	29	46.9	41	**4.1**	41
DER-48	32768	20	16.8	47	8.6	24	11.2	32	69.5	42	**5.7**	41
DER-48	32768	30	30.5	61	14.9	26	13.4	26	107.5	51	**10.0**	51
DER-48	32768	40	36.2	57	21.1	31	16.9	28	133.8	51	**10.4**	46
DER-48	32768	47	57.1	85	21.6	32	21.1	33	178.8	59	**14.2**	54
DER-96	1024	10	0.5	27	0.3	18	**0.3**	20	1.4	23	0.5	23
DER-96	1024	20	0.8	29	0.5	18	0.5	25	2.4	27	**0.4**	27
DER-96	1024	30	1.2	32	0.6	17	0.6	23	3.5	30	**0.5**	30
DER-96	1024	40	1.6	34	1.0	19	0.7	22	4.1	31	**0.6**	32
DER-96	1024	50	2.2	34	1.2	19	0.8	22	5.8	30	**0.7**	30

(continued)

			CPLEX		Gurobi		Mosek		Tulip		Tulip*	
Instance	R	CG	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter
DER-96	1024	60	2.6	34	1.4	19	1.0	23	6.9	31	**0.8**	31
DER-96	1024	64	3.3	38	1.2	19	0.9	22	6.6	31	**0.9**	31
DER-96	2048	10	1.2	37	0.8	21	0.7	29	4.0	29	**0.5**	29
DER-96	2048	20	2.2	33	1.1	19	0.9	23	5.9	26	**0.8**	26
DER-96	2048	30	2.5	44	1.6	22	1.3	25	9.9	35	**1.1**	35
DER-96	2048	40	4.8	38	2.3	26	1.5	23	12.4	33	**1.2**	33
DER-96	2048	50	6.7	41	2.4	23	1.8	25	15.2	36	**1.6**	36
DER-96	2048	56	7.9	45	2.4	20	1.7	21	18.2	38	**1.7**	37
DER-96	4096	10	3.0	41	1.7	24	1.5	28	9.5	32	**1.0**	32
DER-96	4096	20	4.4	51	2.9	27	1.9	26	18.2	39	**1.6**	40
DER-96	4096	30	6.0	53	3.6	24	2.7	28	21.1	35	**2.0**	36
DER-96	4096	40	13.6	53	4.4	24	3.3	27	31.2	39	**2.4**	39
DER-96	4096	50	14.2	45	5.7	25	4.2	25	36.1	37	**2.7**	37
DER-96	4096	53	16.3	51	5.5	24	5.2	28	42.6	40	**3.2**	40
DER-96	8192	10	5.6	53	4.3	27	3.0	28	23.0	35	**2.6**	35
DER-96	8192	20	11.1	62	7.2	33	4.9	32	43.4	40	**3.4**	40
DER-96	8192	30	13.5	59	8.8	31	5.9	26	54.4	40	**3.8**	40
DER-96	8192	40	32.7	63	12.6	35	7.4	25	77.6	45	**5.0**	45
DER-96	8192	50	39.3	65	11.5	25	10.1	33	89.1	44	**6.6**	45
DER-96	8192	60	51.7	75	15.5	29	11.1	31	137.6	60	**8.8**	57
DER-96	16384	10	12.6	61	11.0	37	6.9	34	55.0	41	**4.4**	41
DER-96	16384	20	21.2	62	14.5	31	10.9	31	92.3	42	**6.1**	42
DER-96	16384	30	30.1	68	18.5	32	14.2	34	147.9	50	**10.1**	52
DER-96	16384	40	70.0	69	21.8	28	16.1	30	196.5	54	**11.5**	52
DER-96	16384	50	85.5	73	27.7	29	18.6	32	231.8	57	**14.5**	54
DER-96	16384	57	107.8	86	31.9	31	24.4	39	260.0	55	**17.3**	59
DER-96	32768	10	28.1	70	25.4	45	18.0	39	152.5	52	**11.8**	49
DER-96	32768	20	39.9	57	33.8	36	18.9	28	180.4	37	**10.7**	39
DER-96	32768	30	61.9	72	46.6	34	27.9	31	337.6	58	**21.3**	58
DER-96	32768	40	174.6	88	70.8	42	40.4	39	483.2	69	**30.0**	66
DER-96	32768	50	233.6	102	58.8	32	46.8	36	609.0	74	**43.1**	72
DER-96	32768	54	291.9	119	102.9	54	**55.5**	47	753.7	89	65.4	86
4node	1024	10	0.1	28	0.3	53	**0.1**	28	0.9	31	0.5	30

(continued)

			CPLEX		Gurobi		Mosek		Tulip		Tulip*	
Instance	R	CG	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter
4node	1024	20	0.2	27	0.2	22	**0.2**	26	1.0	27	0.4	27
4node	1024	24	15.7	21	0.4	43	**0.2**	25	1.3	30	0.5	32
4node	2048	10	19.6	24	0.7	51	**0.4**	32	1.9	44	0.9	37
4node	2048	20	0.7	38	0.8	42	**0.5**	37	1.9	33	0.9	32
4node	2048	24	0.7	38	0.6	27	**0.6**	25	2.1	36	0.9	36
4node	4096	10	0.9	36	2.1	63	**0.7**	28	3.6	40	1.3	40
4node	4096	20	0.9	23	1.0	27	**0.6**	19	3.7	26	1.3	26
4node	4096	22	1.1	27	1.7	37	**0.7**	17	4.5	28	1.2	28
4node	8192	10	**1.8**	33	3.4	62	1.8	33	10.2	44	2.1	43
4node	8192	20	3.2	42	4.3	51	**2.3**	36	14.2	43	3.2	44
4node	8192	23	2.7	30	2.3	29	**1.8**	24	12.3	35	2.7	33
4node	16384	10	6.8	61	11.4	85	**5.7**	53	34.4	62	5.7	67
4node	16384	20	7.0	42	20.8	108	6.4	44	31.6	45	**5.5**	44
4node	16384	23	5.8	29	10.7	53	**4.0**	22	26.4	33	4.6	33
4node	32768	10	9.8	42	11.8	42	9.1	40	56.4	52	**8.3**	53
4node	32768	20	17.0	58	35.0	95	**13.5**	45	81.9	60	15.7	65
4node	32768	21	17.0	57	18.7	55	14.6	41	74.8	56	**14.2**	59
4node-base	1024	10	**0.1**	24	0.2	18	0.3	21	0.8	24	0.3	24
4node-base	1024	20	0.3	25	0.3	23	**0.2**	28	1.3	28	0.5	28
4node-base	1024	26	17.0	17	1.0	60	**0.3**	27	1.4	28	0.6	27
4node-base	2048	10	15.0	15	0.8	36	**0.3**	23	1.4	29	0.5	30
4node-base	2048	20	0.8	37	1.0	31	**0.7**	35	3.1	36	0.8	36
4node-base	2048	27	1.0	35	2.6	72	**0.8**	33	3.7	32	0.9	33
4node-base	4096	10	0.9	35	3.3	92	**0.7**	24	4.4	38	0.9	42
4node-base	4096	20	1.8	38	4.4	76	**1.1**	30	8.8	42	1.6	42
4node-base	4096	25	2.3	38	5.3	72	**1.5**	34	9.1	34	1.8	34
4node-base	8192	10	1.8	33	2.0	21	1.5	26	7.6	29	**1.5**	29
4node-base	8192	20	4.4	39	16.3	133	3.9	40	18.1	39	**2.7**	39
4node-base	8192	22	3.8	29	3.7	27	**2.6**	25	19.7	36	2.8	36
4node-base	16384	10	4.4	38	10.1	57	3.6	30	19.3	39	**3.4**	39
4node-base	16384	20	10.6	49	17.1	51	6.9	36	46.2	46	**5.6**	45
4node-base	16384	25	13.5	53	26.2	74	8.0	37	63.4	53	**7.0**	47
4node-base	32768	10	10.9	45	76.1	214	10.3	40	44.3	36	**6.0**	36

(continued)

			CPLEX		Gurobi		Mosek		Tulip		Tulip*	
Instance	R	CG	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter	T(s)	Iter
4node-base	32768	20	27.8	68	80.1	125	25.9	72	119.3	59	**15.6**	63
4node-base	32768	23	20.3	37	29.0	43	14.9	30	107.7	48	**12.9**	50
assets	37500	6	1.6	21	**0.6**	12	1.1	20	2.0	13	1.0	13
env	1200	6	0.0	21	**0.0**	12	0.1	16	0.3	16	0.3	16
env	1875	6	0.1	22	**0.0**	12	0.1	13	0.4	16	0.4	16
env	3780	6	0.1	25	**0.1**	12	0.1	14	0.7	17	0.5	17
env	5292	6	0.2	27	**0.1**	13	0.1	13	0.7	17	0.7	17
env	8232	6	0.3	26	**0.2**	13	0.3	14	1.1	18	1.2	18
env	32928	6	1.7	26	**0.9**	13	1.3	17	5.1	21	4.4	21
env-diss	1200	10	0.0	17	**0.0**	19	0.0	16	0.4	22	0.4	22
env-diss	1200	13	0.1	15	**0.0**	15	0.1	17	0.4	23	0.4	23
env-diss	1875	10	0.1	27	**0.1**	17	0.1	20	0.6	22	0.5	22
env-diss	1875	15	0.1	17	**0.1**	18	0.1	18	0.6	22	0.5	22
env-diss	3780	10	0.2	23	**0.1**	16	0.1	17	0.8	21	0.7	21
env-diss	3780	15	0.2	20	**0.1**	18	0.2	18	1.0	22	1.0	22
env-diss	5292	10	0.4	31	**0.2**	25	0.2	21	1.0	25	1.3	26
env-diss	5292	15	0.3	22	**0.3**	23	0.3	22	1.3	25	1.5	25
env-diss	8232	10	0.6	26	0.4	22	**0.4**	18	1.7	22	1.8	22
env-diss	8232	15	1.0	31	0.6	29	**0.5**	23	3.2	35	2.6	35
env-diss	32928	10	4.6	37	2.8	36	**1.9**	17	8.0	27	7.2	27
env-diss	32928	14	4.8	28	**2.1**	22	2.5	19	10.0	27	7.7	27
phone	32768	5	0.5	15	**0.4**	8	0.6	8	1.9	10	0.7	10
stormG2	1000	10	0.7	35	0.5	21	**0.3**	21	2.0	32	1.5	31
stormG2	1000	20	1.4	33	0.8	18	**0.5**	19	4.5	29	1.7	29
stormG2	1000	21	1.6	37	0.8	18	**0.5**	22	4.0	29	1.7	28