Alma Mater Studiorum – Università di Bologna

Dottorato di Ricerca in Automatica e Ricerca Operativa
Ciclo XVIII

Settore Concorsuale di afferenza: 01/A6 - RICERCA OPERATIVA
Settore Scientifico disciplinare: MAT/09 - RICERCA OPERATIVA

On the interplay of Mixed Integer Linear, Mixed Integer Nonlinear and Constraint Programming

Sven Wiese

Coordinatore Dottorato:
Chiar.mo Prof. Daniele Vigo

Relatore:
Chiar.mo Prof. Andrea Lodi

Esame finale anno 2016

Abstract

In this thesis we study selected topics in the field of Mixed Integer Programming (MIP), in particular Mixed Integer Linear and Nonlinear Programming (MI(N)LP). We set a focus on the influences of Constraint Programming (CP).

First, we analyze Mathematical Programming approaches to water network optimization, a set of challenging optimization problems frequently modeled as non-convex MINLPs. We give detailed descriptions of many variants and survey solution approaches from the literature. We are particularly interested in MILP approximations and present a respective computational study for water network design problems. We analyze this approach by algorithmic considerations and highlight the importance of certain convex substructures in these non-convex MINLPs. We further derive valid inequalities for water network design problems exploiting these substructures.

Then, we treat Mathematical Programming problems with indicator constraints, recalling their most popular reformulation techniques in MIP, leading to either big-M constraints or disjunctive programming techniques. The latter give rise to reformulations in higher-dimensional spaces, and we review special cases from the literature that allow to describe the projection on the original space of variables explicitly. We theoretically extend the respective results in two directions and conduct computational experiments. We then present an algorithm for MILPs with indicator constraints that incorporates elements of CP into MIP techniques, including computational results for the JobShopScheduling problem.

Finally, we introduce an extension of the class of MILPs so that linear expressions are allowed to have non-contiguous domains. Inspired by CP, this permits to model holes in the domains of variables as a special case. For such problems, we extend the theory of split cuts and show two ways of separating them, namely as intersection and lift-and-project cuts, and present computational results. We further experiment with an exact algorithm for such problems, applied to the Traveling Salesman Problem with multiple time windows.

Keywords

· Mixed Integer Linear Programming

· Mixed Integer Nonlinear Programming

· Constraint Programming

· Water network optimization

· Nonlinear network flows

· Piecewise linear approximations

· Indicator constraints

· big-M constraints

· Disjunctive Programming

· Job Shop Scheduling

· Cutting planes

· Split cuts

· Intersection cuts

· Lift-and-project cuts

· Traveling Salesman Problem with multiple time windows

Acknowledgements

It was in late 2014 that I stumbled upon a video on YouTube, showing Steve Jobs, the former CEO of Apple Inc., giving a speech at the 114th Stanford Commencement ceremony [STF]. Part of that speech was a story entitled *connecting the dots*. It tells how Jobs dropped out of college shortly after enrolling, but then stayed as a drop-in and randomly attended classes that arose his curiosity. Calligraphy, for example, was one of them. Only years after, he realized that his dropping out, that seemed an unfortunate and desperate circumstance at the time, led to the fact that today's Apple's personal computers (and possibly others) support beautiful typography in the form of "multiple typefaces and proportionally spaced fonts". It was an inspiring speech, and for some reason, it was also a moment that made me inevitably think of my time as one of Andrea Lodi's PhD students at the University of Bologna. I somehow remembered the feeling I had after so many research meetings during which at some point, talking about one particular aspect of our work, we wondered if and how there was an intimate connection to a supposedly different one that we talked about the other day. And sometimes we managed to connect the dots. Not everything is connected all the time, but I learned that it can be beneficial to at least wonder whether it might be. In some sense I feel that I learned an important life lesson here.

Andrea, if I were to put all my gratitude into a single sentence, I would probably say: thank you for never being superficial, but trying to always get the full picture, and for teaching me the value in that. Your persistent ambition to understand Mathematical Programming more and more, and to open it to other areas of the mathematical sciences constantly inspired me to write this thesis. I am sure that sooner or later, I will be able to connect the dots and fully understand what writing this thesis has done for me and means to me. Oh, and thank you for infecting me with that affinity for electronic devices with an apple on them.

I would have never written this thesis if Prof. Aristide Mingozzi had not introduced me to the beautiful world of Optimization and Operations Research, and if Roberto Roberti and Enrico Bartolini had not taught me proper programming during my stay in Cesena. I am very grateful for that. I thank the research group in Bologna for having accepted me so warmly. Thanks to former and current fellow PhD students Eduardo Àlvarez-Miranda, Tiziano Parriani, Paolo Tubertini, Claudio Gambella, Dimitri Thomopulos, Alberto Santini and Maxence Delorme for discussions, lunches, aperitivi and dinners, and occasional help with bureaucracy in Italian. Thank you Jonas Schweiger, your presence here has been an enrichment, in many ways.

I feel indebited to Pierre Bonami and Andrea Tramontani for giving me the possibility to work with them, for encouraging and advising me. Without them, I woudn't be able to implement Mathematical Programming algorithms into a computer. I also thank Claudia D'Ambrosio and Cristiana Bragalli for all the time they spent on discussions with me.

Thanks to the research group of Prof. Michael Jünger in Cologne for making my stay there so pleasant and memorable. The same goes for Dennis Michaels and my visit in Dortmund. Thanks to Jesco Humpola, Ambros Gleixner, Ralf Lenz and Robert Schwarz for inviting me to and taking care of me at the Zuse-Intitut in Berlin. Thanks to Felipe Serrano for being so curious when it comes to mathematics. Passing time at ZIB has always been very inspiring to me.

I have to thank Stefan Vigerske and again Ambros Gleixner for giving technical advice regarding SCIP, Wei Huang, Lars Schewe and Antonio Morsi for their helpful comments in the context of water network operation, Slim Belhaiza and Marco Lübbecke for sharing VRPMTW and RCPSP instances, Fabio Furini for pointing my attention to the Lazy Bureaucrat Problem, and Marc Pfetsch for his eminent interest in variables with non-contiguous domains and for sharing an extensively long list of ideas in that regard.

Bologna, March 11, 2016

To Giulia and Fefé.

"God made the integers, all else is the work of man."

Leopold Kronecker

Contents

Introduction

"You have different ideas of what a MIP is", concluded the professor, as two of his academic children did not reach a common denominator in a discussion during the workshop[1]. MIP - Mixed Integer Programming. Should be pretty clear, right? Well, it is not.

The acronym MIP and its usage currently undergo a certain change. In the Mixed Integer Programming community, things like "MIP is not MIP anymore"[2] can be heard here and there, and underline this development. Some time ago, saying "MIP" actually meant, without much doubt, Mixed Integer Linear Programming (MILP). Nowadays, notions like Mixed Integer Conic, Semidefinite or Convex Programming are receiving more and more attention, thus making just "MIP" somewhat ambiguous. Another, very popular extension has become Mixed Integer Nonlinear Programming (MINLP). Some of these programming paradigms are extensions of others, in the sense that the problem classes to which they can be applied exhibit inclusion relations among each other. For example, MINLP is an extension of MILP. Nevertheless, because a programming paradigm is characterized not only by the modeling tools it allows for, but also by the algorithmic frameworks it applies in practice, it does make sense to keep a strict distinction. MILP and especially algorithms for solving MILPs have been studied and developed over decades, while MINLP can be seen as a more recent phenomenon. Commercial MILP codes have been around for years, but reliable MINLP codes are relatively fresh. The latter being an extension of the former, it is natural to attempt to extend also the algorithms used to tackle MILPs. Although, or maybe just because this works only up to a certain point, the two paradigms are intimately connected and influence each other, in both directions, and this interplay is one of the subjects of this thesis.

Another paradigm in mathematical optimization is Constraint Programming (CP), that in some sense has become a competitor of MILP, especially for solving combinatorial optimization problems. Also for CP, commercial codes are nowadays available. There have been efforts to explore the algorithmic synergies between CP

[1] Meant is the 20th Combinatorial Optimization Workshop, held in Aussois, France in January 2016.

[2] Jeffrey T. Linderoth at the 2015 Mixed Integer Programming workshop, held at the University of Chicago

and MIP, and this is another subject of our work.

We study roughly three topics that live inside the world of MIP, divided into three chapters. The aforementioned interactions of MILP, MINLP and CP are the oxygen that nurtures our analysis in many cases, or at least its initial motivation. In the first chapter we introduce some basics of MILP, MINLP and CP that we believe to be useful for the exposition of the main Chapters 2 - 4. Figure 1.0 schematically represents which interactions are intended to be treated, to what degree and in which chapters and sections.

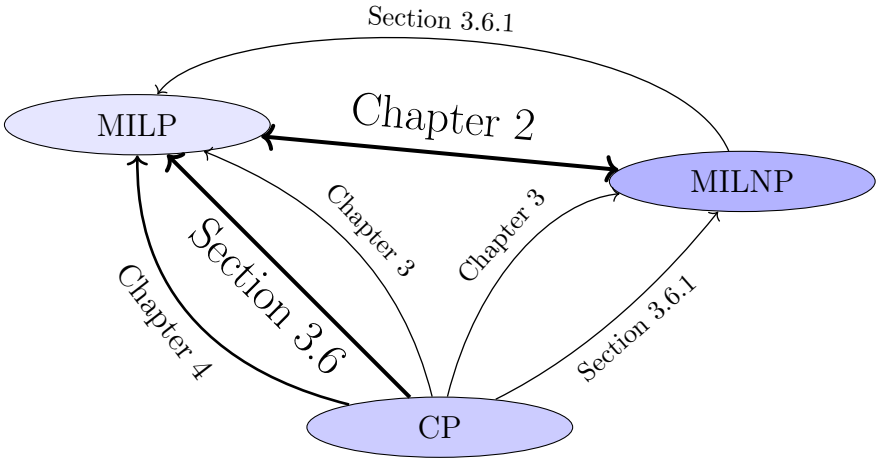


Figure 1.0: Interactions of MILP, MINLP and CP

Parts of this thesis have been published as articles in scientific journals, in particular the main parts of

- Chapter 2 in 'Claudia D'Ambrosio, Andrea Lodi, Sven Wiese, and Cristiana Bragalli. Mathematical programming techniques in water network optimization. *European Journal of Operational Research*, 243(3):774–788, 2015',
- and of Chapter 3 in 'Pierre Bonami, Andrea Lodi, Andrea Tramontani, and Sven Wiese. On mathematical programming with indicator constraints. *Mathematical Programming*, 115(1):191–223, 2015'.

We hope that our findings will be considered an original contribution to the understanding of Mixed Integer Programming.

1 Concepts

In this first chapter we try to complete the difficult, if not impossible task of providing an overview of the main concepts, that we build our work on, in a concise manner. We keep a generally low level of detail, but will of course be more specific in selective parts that serve the ease of exposition in successive chapters. We will point the reader's attention to more advanced material wherever possible, and stress that none of the results and definitions given in this chapter are original. If their origin is not indicated, they can be found in standard text books.

We consider optimization problems $(c, \mathcal{F})$, given by a feasible region $\mathcal{F}$ and a (for our purposes real-valued) objective function c defined on $\mathcal{F}$. If we deal with a minimization problem, we seek an $x \in \mathcal{F}$ such that

$$c(x) \leq c(y) \ \forall \ y \in \mathcal{F}.$$

Maximization problems are treated in an analogue fashion. For any $\tilde{\mathcal{F}} \subseteq \mathcal{F}$, we call $(c, \tilde{\mathcal{F}})$ a subproblem of $(c, \mathcal{F})$. A special role often play convex optimization problems, given by the minimization of convex functions over convex feasible regions. Optimization problems that cannot be cast by this description are thus non-convex. We will encounter various problems of both types in this text.

We mostly use standard notation that is self-explanatory, or otherwise introduce more involved concepts whenever they are needed. The only recurring notation that we use throughout the whole text is $[n] := \{1, \ldots, n\}$ for any positive integer n. Also, in Chapter 3, bold letters like $\mathbf{u}$ will stand for constant vectors in $\mathbb{R}^n$, and $(\mathbf{u})_i$ for their i-th component.

1.1 Mixed Integer Linear Programming

MILP is the name for the mathematical problem of minimizing a linear function over a subset of $\mathbb{R}^n$, that can be described by using only linear (in-)equalities and requiring a subset of the variables to take only integer values. MILP evolved from Linear Programming (LP), cf. Section 1.1.1, in the mid-sixties [Bix12, Coo12]. Over the years, MILP has become a technology and reliable and robust commercial codes for solving MILPs are available. The most famous ones are probably the MILP versions

of the optimization codes IBM ILOG CPLEX [CPX], Gurobi Optimizer [GUR] and FICO Xpress [XPR]. Also non-commercial codes like Cbc from COIN-OR [COIN] are available. It is safe to say that there is competition going on in the industry of MILP solvers, and their performance is steadily tested on the electronically available MIPLIB, a library of test instances, updated several times since 1992. The current version is MIPLIB 2010 [MLib, KAA+11]. This collection tries to cover the variety of applications that can be modeled by MILP, that is fairly vast [Dan60]. In this text, we consider an MILP of the general form

$$\min \quad c^T x \tag{1.1}$$

$$\text{s.t.} \quad Ax = b \tag{1.2}$$

$$x \in \mathbb{R}^{n-p}_+ \times \mathbb{Z}^p_+, \tag{1.3}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. In this form, we thus have n variables, corresponding to the columns of the matrix A, and m equality constraints, $a^i x = b^i$, corresponding to its row vectors a^i. The feasible region is completed by the fact that the variables have to be non-negative, and that the last p of them have to take integral values. A MILP is thus a non-convex optimization problem. In the spacial case of a purely Integer Linear Program (ILP), where $p = n$, any feasible solution is thus a completely integral vector. However, also when $p < n$, we will call a solution to (1.1) - (1.3), i.e., a vector whose last p components only are integer values, an integral solution.

There is a lot of flexibility in obtaining an MILP formulation for a specific problem. That is, there are degrees of freedom on the modeling side, and different formulations can lead to more, or to less success when attempting to apply an MILP solver [Vie15b]. This phenomenon, generalized to convex MINLPs, cf. Section 1.2, will be the initial theme of Chapter 3 and is, on a somewhat large scale, due to the general way in which the current generation of MILP solvers proceed in solving such a problem. At the same time, also on this computational side there is, on a smaller scale, much flexibility in using the arsenal of techniques that can be included as ingredients into a solver. Many different components like presolving techniques, branching schemes, cutting planes or primal heuristics interact, see, e,g , [LL11b] and references therein. We will present the ones with most importance to the topics of the later chapters in the following sections. A very complete handbook on the interior of an MILP solver can be seen in [Ach09].

1.1.1 Linear Programming

A key to understanding the general way in which an MILP solver tackles a problem like (1.1) - (1.3) is the study of a problem class that is contained in MILP itself,

namely, LP. One general form of an LP can be introduced by considering again (1.1) - (1.3), but dropping the requirement that some variables have to be integral. This is precisely how to obtain an LP in the so-called standard form,

$$\begin{aligned} \min \quad & c^T x && (1.4)\\ \text{s.t.} \quad & Ax = b && (1.5)\\ & x \in \mathbb{R}^n_+. && (1.6) \end{aligned}$$

There are other general forms of LPs, for example the canonical one, including inequality constraints instead of equalities, $a^i x \lesseqgtr b^i$. It is also possible to have so-called free variables, that are unconstrained in sign, in an LP. However, different forms can be transformed into each other by the use of slack variables and the (dis-)aggregation of variables. We will analyze an LP of such a different form further down. In the same way, also an MILP can be introduced in slightly different, but equivalent general forms. Keeping that in mind, we can assume that upper bounds on variables, if present, are contained in the matrix A in (1.1) - (1.3) or (1.4) - (1.6). In this way, the notation (1.1) - (1.3) is flexible enough to include binary variables. The point of view that allows inequality constraints in the feasible region of (1.1) - (1.3) will become important in Section 1.1.3.

The hour of birth of LP as a programming paradigm is often claimed to be the year 1947, when George B. Dantzig introduced the simplex algorithm [Bix12]. This algorithm is based on the fact that the feasible region of (1.4) - (1.6) is a polyhedral subset of a finite-dimensional space. Hence, differently from an MILP, an LP is a convex optimization problem. Furthermore, this polyhedral set can be characterized by a finite number of vertices and extreme rays, and the optimal objective value - assuming that it is bounded - can be shown to be attained in at least one of these vertices. It is therefore sufficient to concentrate on this finite set of points in order to find an optimal solution to (1.4) - (1.6). In standard textbooks, this geometric point of view usually goes hand in hand with algebraic approaches to the simplex method. Therefore - assuming that A has full row rank - one chooses m linearly independent columns of A, whose indices are collected in the set $B \subseteq [n]$, thus defining one of finitely many so-called bases. After a possible reordering of the columns, and setting $N = [n] \setminus B$, one can rewrite (1.5) as $A_B x_B + A_N x_N = b$, and the multiplication with A_B^{-1} leads to the so-called simplex tableau, a reformulation of (1.5) - (1.6),

$$\begin{aligned} x_i &= \hat{x}_i \ + \sum_{j \in N} r^i_j x_j && i \in B && (1.7)\\ x_i &\geq 0 && i \in [n]. && (1.8) \end{aligned}$$

Formulas for $\hat{x}_i$ and r_j^i can be recovered by linear algebraic transformations. The point $\hat{x}$, augmented by $|N|$ zeros, is called the basic solution corresponding to the basis B, in which the variables x_i are partitioned into basic, $i \in B$, and non-basic ones, $i \in N$. If $\hat{x} \geq 0$, it is a basic feasible solution, and one can show that such solutions exhibit a one-to-one correspondence to the vertices of the feasible region of (1.4) - (1.6). The simplex tableau is fundamental for deciding whether a given basis leads to an optimal solution, and if not, how to construct a new basic feasible solution with strictly lower objective value. Most LP codes (that are building blocks of MILP codes, as we will see in the next section,) give the possibility of recovering the simplex tableau whenever an LP has been solved by the simplex method, and this will play an important role in Section 4.2.1. For profound textbooks of the theories of Linear Programming and polyhedral sets, we refer the reader to [LY84, DT06a, DT06b, Sch98]. We point out that although the simplex algorithm is complexity-wise not a polynomial algorithm, it is highly efficient in practice, and this fact has vastly contributed to the advances made in the design of algorithms for MILP.

We now spend some time on introducing an important theoretical concept in the context of LP, that has, in turn, also significant impact on the practical implementation of the simplex algorithm. Namely, we will study the concept of duality. This notion led, among other things, to the introduction of the dual simplex algorithm, that is outside the scope of this text though. Therefore, consider an LP in the form

$$\begin{aligned} \max \quad & c^T x && (1.9)\\ \text{s.t.} \quad & A_1 x = b_1 && (1.10)\\ & A_2 x \leq b_2 && (1.11)\\ & x \in \mathbb{R}^n_+, && (1.12) \end{aligned}$$

with $A_1 \in \mathbb{R}^{m_1 \times n}$ and $A_2 \in \mathbb{R}^{m_2 \times n}$. (1.9) - (1.12) could, as sketched before, also be transformed into standard form. To every LP, in the context of duality called the primal problem, is associated its so-called dual LP, that can be constructed following concise rules. Roughly speaking, to every constraint in the primal LP corresponds a variable in the dual, and whether such a variable is non-negative or free depends on whether the constraint is an inequality or equality constraint[1]. Following these rules, the dual of (1.9) - (1.12) can be shown to be

$$\min \quad b_1^T \lambda + b_2^T \mu \qquad (1.13)$$

[1]This sentence can be repeated after changing the roles of primal and dual.

$$\text{s.t.} \quad A_1^T \lambda + A_2^T \mu \geq c \tag{1.14}$$

$$\lambda \in \mathbb{R}^{m_1} \tag{1.15}$$

$$\mu \in \mathbb{R}_+^{m_2}. \tag{1.16}$$

The notion of duality also gives rise to explicit conditions for a feasible solution of the primal LP, or equivalently, for a pair of primal and dual solutions, to be optimal. These conditions are often referred to as complementary slackness, and for the above pair of primal, (1.9) - (1.12), and dual LP, (1.13) - (1.16), they can be written as

$$(a_2^i x - b_2^i) \cdot \mu_i = 0 \quad \forall\, i = 1, \ldots, m_2. \tag{1.17}$$

We will see that these optimality conditions in LP can be extended to Nonlinear Programming in Section 1.2.1. Furthermore, the so-called strong duality theorem of LP states that the optimal objective values of a primal-dual pair like (1.9) - (1.12) and (1.13) - (1.16) coincide. We will need the concept of duality in the context of certain LPs in Section 4.2.2.

After having sketched how an LP can be solved algorithmically and having discussed the notion of duality, we turn to the relation between an MILP as in (1.1) - (1.3) and the LP (1.4) - (1.6), obtained by dropping the partial integrality requirement on the variables. This relation can be seen in the light of a relaxation, that we now define formally as is, e.g., [Geo71].

Definition 1.1. *An optimization problem $(\hat{c}, \hat{\mathcal{F}})$ is a relaxation of $(c, \mathcal{F})$ if and only if $\mathcal{F} \subseteq \hat{\mathcal{F}}$ and $\hat{c}(y) \leq c(y)\ \forall\ y \in \mathcal{F}$.*

In fact, the LP (1.4) - (1.6) is a relaxation of the MILP (1.1) - (1.3), and it is commonly and inevitably called its LP-relaxation, sometimes also its continuous relaxation. Every solution to the LP-relaxation of an MILP is also called - in contrast to integral solutions - a fractional solution. The notion of a relaxation will be very useful in the context of branch & bound, treated in the next section. If we denote the optimal objective values of $(c, \mathcal{F})$ (the MILP) and $(\hat{c}, \hat{\mathcal{F}})$ (the LP) by z and $\hat{z}$, respectively, we have that $\hat{z} \leq z$. That is, $\hat{z}$ is a lower bound on z, also called a dual bound[2]. We will see that a driving theme in Mixed Integer Programming is the search for dual bounds that are close to z. These are often called tight, or simply good dual bounds. The following definition provides a way to measure how close a problem is to one of its relaxations.

[2]The name dual bound has its origin, as can be guessed, in LP duality; one can show that the dual objective value of any dual feasible solution is a lower bound of the objective value of any primal feasible solution.

Definition 1.2. *Let $(\hat{c}, \hat{\mathcal{F}})$ be a relaxation of $(c, \mathcal{F})$, and denote their optimal objective values by $\hat{z}$ and z, respectively. The dual gap between $(\hat{c}, \hat{\mathcal{F}})$ and $(c, \mathcal{F})$ is defined as $\delta := (z - \hat{z})/z$.*

The dual gap is only one of many possible measures of the aforementioned distance. It is, however, a very meaningful one with regard to the way in which MILPs are usually solved, as will become clear in the next section.

1.1.2 Branch & bound

Branch & bound is a generic framework for solving optimization problems $(c, \mathcal{F})$ in which every subset of $\mathcal{F}$ can be further partitioned into mutually exclusive subsets - the branching part in *branch & bound* - in such a way that the arising subproblems allow for relaxations with easily computable dual bounds. Discrete optimization problems almost naturally satisfy the first requirement. For an MILP, such partitions are obtained by implicitly decomposing the feasible region into all possible assignments of values to the integer constrained variables. *Enumeration* is often used to describe this process. Every subproblem is again an MILP, and as such has an LP-relaxation, that is (in practice and in comparison to the initial MILP) easily solvable. In many cases, including that of MILPs, the way of decomposing an initial problem into subproblems can be implemented in a way that gives rise to a data structure that is a tree: every newly created subproblem can be seen as a child node of the node whose feasible region has just been partitioned. The process of finding an optimal solution in this way is therefore also called a tree-search[3]. Of fundamental importance during the enumeration process are dual bounds of the subproblems and an upper bound UB on the optimal objective value of the original problem, also called a primal bound: whenever the dual bound of a subproblem exceeds UB, the optimal solution cannot be contained in the subtree originating at this node, and the latter can be discarded. This is the bounding part in *branch & bound*. It highlights the importance of good dual and primal bounds and is the main contributor to the success of branch & bound, because it makes the enumeration in some sense intelligent.

We give the outline of a branch-and-bound procedure for an MILP like (1.1) - (1.3) in Algorithm 1.1. As sketched before, subproblems correspond to nodes in the tree, and these are explored one after another during the search process. If a node has not been explored yet, it is also called open, and we denote the set of open nodes by $\mathcal{T}$. Because we always solve the LP-relaxation of a subproblem, the nodes are conventionally already memorized as just LPs. The branching step in Lines 10

[3] In virtually all MILP codes, the feasible region of every subproblem is partitioned into at most two subsets, and the tree is actually a heap.

Algorithm 1.1: MILP branch & bound

```
1  let the root node ρ be the LP-relaxation of (1.1) - (1.3);
2  set T = {ρ} and UB = ∞;
3  while T ≠ ∅ do
4  |  choose η ∈ T, set T = T \ {η};
5  |  solve η and denote by x̂ and ẑ its optimal solution and objective value;
6  |  if ẑ < UB then
7  |  |  if x̂ integral then
8  |  |  |  update UB = ẑ;
9  |  |  else
10 |  |  |  choose an integer-constrained x_i with fractional value x̂_i;
11 |  |  |  create nodes η⁻ and η⁺ by adding the constraints x_i ≤ ⌊x̂_i⌋ and
   |  |  |  x_i ≥ ⌈x̂_i⌉, respectively, to η;
12 |  |  |  set T = T ∪ {η⁻, η⁺};
13 |  |  end
14 |  end
15 end
```

and 11 partitions the feasible region of a node by simply dividing it into those x that satisfy $x_i \leq \lfloor \hat{x}_i \rfloor$, and those that satisfy $x_i \geq \lceil \hat{x}_i \rceil$. The primal bound UB is the objective value of the best, and to (1.1) - (1.3) feasible, solution encountered so far. Convergence of Algorithm 1.1 can be assured if, for example, the feasible region of an MILP is a compact set. Figure 1.1 schematically represents the enumeration of an MILP in a search tree. Open nodes are depicted in blue, and nodes in which a feasible solution has been found are marked with an asterisk.

There are significant degrees of freedom in several steps of Algorithm 1.1. For example, with respect to which criteria do we choose a node η in Line 4? And how do we choose a fractional variable in the branching step? Answers to these highly non-trivial questions lead to some of the aforementioned ingredients included in an MILP solver and can significantly influence its performance. The bounding step happens precisely in Line 6. Only if the optimal LP objective value $\hat{z}$ is lower than the current primal bound UB (and, of course, if the LP is feasible), the remaining steps are executed. New nodes, that enlarge the list of LP problems to be solved and thus represent computational workload, are only potentially created if the condition in Line 6 is true. It is therefore desirable to be false as often as possible. Then, η will not give rise to the creation of child nodes, and is said to be pruned or fathomed. The diamond nodes in Figure 1.1 symbolize nodes that have been pruned. Apart from just establishing a complete enumeration of the feasible solutions, the branching step in Line 11 also serves this purpose: by adding one of the constraints $x_i \leq \lfloor \hat{x}_i \rfloor$ OR $x_i \geq \lceil \hat{x}_i \rceil$ when creating a new child node, its feasible region is reduced with

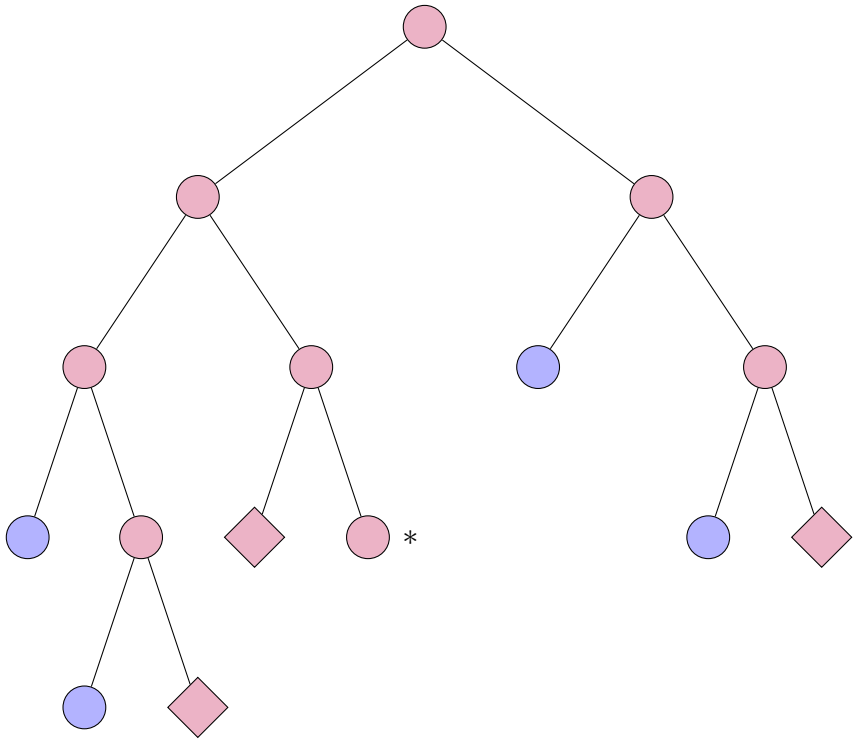


Figure 1.1: Schematic representation of a search tree in branch & bound

respect to the one of the parent node. Hence, by going downwards in the search tree, the values of $\hat{z}$ in Line 6 can only increase and therefore have higher probabilities of leading to a pruning.

At any point during the tree-search, a lower bound on the optimal objective value of an open node is given by the optimal objective value of its already solved parent node. The current dual bound on the optimal objective value of the MILP is then defined as the minimum of these lower bounds of all open nodes in the tree. All in all, the importance of good, that is, possibly tight dual bounds on the optimal objective value of the MILP is highlighted. If a tree-search is initiated with an already good dual bound, it takes less depth in the tree to be able to perform some pruning. For the same reason, an important component of an MILP solver are primal heuristics, meaning any attempt to decrease the primal bound UB during the search process, possibly already in Line 2 of Algorithm 1.1. Another component that is again motivated by the need for good dual bounds is given by cutting planes, that we will treat in the next section.

If in Line 7, we end up with an LP solution that is feasible to the underlying MILP, we can update UB. Also this type of nodes does not lead to the creation of new nodes, cf. Figure 1.1, because the solution of their relaxation has automatically solved the actual associated subproblem. A special case of a node η for which the condition tested in Line 7 will always be true, is the one in which all integer variables

Algorithm 1.2: MILP cutting plane

1 **do**
2 | solve the LP-relaxation (1.1) - (1.3) to the point $\hat{x}$;
3 | identify one or more cutting planes in $\mathcal{L}$ that are violated by $\hat{x}$;
4 | if some were found, add them to the LP-relaxation;
5 **while** *cutting planes have been added*;

have been completely fixed to integer values by previous branching decisions. Binary variables for example become fixed immediately after they have been branched on once. In any case, the subproblems associated to such nodes η deserve a special name.

Definition 1.3. *The LP that is obtained from* (1.1) - (1.3) *by additionally imposing the constraint that the last p components of x be equal to some* $\tilde{x}_{(p)} \in \mathbb{Z}^p_+$ *is called a leaf problem of* (1.1) - (1.3).

Leaf problems can also be defined for MINLPs (cf. Section 1.2) and in that context will become important in Chapter 2.

1.1.3 Cutting planes

Historically, the notion of *cutting planes*, or just *cuts*, arose from an approach of solving (mostly pure) integer programs that is different from the concept of branch & bound. A cutting plane or valid linear inequality of an MILP, $\alpha^T x \geq \beta$, is a linear inequality that is satisfied by all feasible solutions. If we assume that a family of such valid linear inequalities $\mathcal{L}$ is given, a generic cutting-plane algorithm for an MILP like (1.1) - (1.3) is outlined in Algorithm 5. The name cutting plane becomes clear from therein: if a valid inequality, that is violated by the current fractional solution, is added to the LP-relaxation, then this fractional solution is separated, or *cut off* from its feasible region. The identification of violated cutting planes in Line 3 is also called the separation problem. Algorithm 5 can be seen to terminate in a finite number of steps if, for example, $\mathcal{L}$ is finite. However, whether it terminates with a feasible or even optimal solution to (1.1) - (1.3) or not depends on the given family $\mathcal{L}$. A sufficient condition to guarantee the former is given when $\mathcal{L}$ contains all facet-defining inequalities of the polyhedral set that defines the convex hull of the feasible region of (1.1) - (1.3). To find such a family $\mathcal{L}$ is usually pursued in the discipline of polyhedral combinatorics. It is a strategy that has mostly been applied in situations where problem-specific structure of the matrix A and the vector b are given. The motivation behind this approach is that, ultimately, an MILP could be solved to optimality by solving just an LP.

However, this holy grale of integer programming, i.e., finding such an $\mathcal{L}$, has so far not turned out practical for general MILPs. In contrast to facet-defining inequalities that, by definition, are strong but might be hard to identify, general-purpose MILP cutting planes have therefore been studied extensively, see, e.g., [Cor08] for a nice tutorial. Such cutting planes do not rely on specific structure of the matrix A, but in some sense only on the integrality requirements of the integer-constrained variables. They are usually weaker than facet-defining ones, but are in turn easy to generate. As an example, we introduce a special version of Gomory's Mixed Integer (GMI) cuts [Gom63], that relies on the optimal simplex tableau of the LP-relaxation of an MILP.

Definition 1.4. *Let the optimal simplex tableau of the LP-relaxation of* (1.1) - (1.3) *be given as in* (1.7) - (1.8)*, and let* $J := N \cap \{n-p+1, \dots, n\}$ *denote the integer-constrained non-basic variables. If the* i*-th basic variable is integer-constrained and* $\hat{x}_i$ *is fractional, then the corresponding GMI cut is given by the linear inequality*

$$\sum_{j \in J} \min \left\{ \frac{\lceil r_j^i \rceil - r_j^i}{\hat{x}_i - \lfloor \hat{x}_i \rfloor}, \frac{r_j^i - \lfloor r_j^i \rfloor}{\lceil \hat{x}_i \rceil - x_i} \right\} x_j + \sum_{j \in N \setminus J} \max \left\{ \frac{-r_j^i}{x_i - \lfloor \hat{x}_i \rfloor}, \frac{r_j^i}{\lceil \hat{x}_i \rceil - x_i} \right\} x_j \geq 1.$$

Even in cases where Algorithm 5 does not terminate with an optimal solution to (1.1) - (1.3), it does have a very important effect, the same that any addition of valid inequalities should have. By adding cutting planes to the LP-relaxation of an MILP we reduce its feasible region and thus increase the dual bound. Together with the importance of good dual bounds inside branch & bound, this has led to the fact that, at some point, the concepts of branch & bound and cutting planes have been integrated into frameworks that are commonly called branch-and-cut algorithms. The hour of birth of branch & cut is sometimes seen in the seminal work of Padberg and Rinaldi on the Traveling Salesman Problem [PR91]. The basic idea of a branch & cut algorithm for a general MILP is, after all that has been said up to now, easy to understand. In line 5 of Algorithm 1.1, if after the LP-relaxation of the node η has been solved, some violated cutting planes are separated, then the LP can be resolved in order to increase the value $\hat{z}$. The separation of cutting planes at a node can as well be iterated and several rounds of separation and LP resolves can be performed. Cutting planes are nowadays an important ingredient of any MILP code, and different strategies regarding the separation of different families of general-purpose MILP cutting planes influence their performance. A special role in this regard often plays the root node in Algorithm 1.1. Usually, more effort on the separation of cutting planes is spent at this designated node. Nevertheless, also

at the remaining nodes cutting planes can be separated. In this case we further encounter two conceptually different types of cutting planes, namely globally valid and locally valid ones. A cutting plane is globally valid if it is guaranteed to be a valid linear inequality for the original MILP, and thus for the subproblems corresponding to all nodes in the tree. Instead, a locally valid cutting plane may be valid for the nodes in the subtree originating at the node η at which it is generated only. From a practical point of view, the discrepancy between global and local cuts is non-negligible, and we will briefly come back to this issue in Section 3.6.1.

In any case, the most common empirical quality measure of cutting planes is the amount by which they reduce the dual gap after they are added to the LP-relaxation of an MILP. We will use this measure in order to analyze the performance of an extension of a family of general-purpose MILP cutting planes in Section 4.2.4. In Section 4.2.1 we compare two cutting planes theoretically, and therefore the definition of dominance will be useful.

Definition 1.5. *In the setting of* (1.1) - (1.3)*, we say that the valid inequality* $\alpha^T x \geq \beta$ *dominates the valid inequality* $\tilde{\alpha}^T x \geq \tilde{\beta}$*, if*

$$\{x \in P \mid \alpha^T x \geq \beta\} \subseteq \{x \in P \mid \tilde{\alpha}^T x \geq \tilde{\beta}\},$$

where $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ *denotes the feasible region of the LP-relaxation of* (1.1) - (1.3)*.*

1.2 Mixed Integer Nonlinear Programming

Life is not always a straight line, and neither are the problems we encounter in real-world applications always describable by linear functions. The modeling of real-world phenomena by nonlinear optimization models has experienced growing popularity over the last decades. As in the linear case, a collection of test instances has been established by now in the MINLPLib [MNLib]. The straightforward extension of an MILP to an MINLP is achieved by allowing that the functions involved in the constraints $a^i x \lesseqgtr bi$ (or the objective function) are nonlinear in the variables x. The general form of an MINLP that we consider in this text is

$$\begin{aligned}
\min \quad & f(x) && && (1.18)\\
\text{s.t.} \quad & g_i(x) \leq 0 && \forall\ i = 1, \ldots, m_1 && (1.19)\\
& h_j(x) = 0 && \forall\ j = 1, \ldots, m_2 && (1.20)\\
& x \in \mathbb{R}^{n-p} \times \mathbb{Z}^p. && && (1.21)
\end{aligned}$$

In the context of MINLP, the transformation from inequality constraints to equality constraints cannot be undertaken without any care as in the case of MILP and LP. While the continuous relaxation of an MILP, its LP-relaxation, is always a convex optimization problem, this is not true for the continuous relaxation of (1.18) - (1.21), obtained again by dropping the partial integrality requirements. In fact, as soon as one function $h_j(\cdot)$ is non-linear, the set of points x that satisfy $h_j(x) = 0$ is non-convex. Also, if a function $g_i(\cdot)$ is a non-convex function, the constraint $g_i(x) \leq 0$ describes a not necessarily convex set. Thus, (1.18) - (1.21) can be seen as the general form of a non-convex MINLP. A convex MINLP instead is usually written as just (1.18), (1.19) and (1.21), together with the specification that the functions $f(\cdot)$ and $g_i(\cdot)$ are convex. The name convex MINLP is of course not hundred percent precise, since the integrality requirements make an MINLP always a non-convex optimization problem. Instead. it refers to the fact that the continuous relaxation of a convex MINLP is convex. Several subclasses of MINLP problems have their own acronym. For example, Mixed Integer Quadratic Programming (MIQP) is identical to MILP, except that it allows for a quadratic objective function. Allowing the constraints to be quadratic as well leads to Mixed Integer Quadratically Constrained Quadratic Programming (MIQCQP), and further extensions are possible.

MINLP as a programming paradigm has not been studied for as long as MILP and respective codes are less robust. Yet, a variety of them, commercial and open-source ones, are nowadays available. They are generally divided into solvers for convex MINLPs on the one hand, and non-convex ones on the other. In the first category fall, for example, BONMIN [BBC+08] or FilMINT [ALL10]. In the second category we have BARON [Sah14, TS05], COUENNE [BLL+09], SCIP [BHV12] or ANTIGONE [MF14]. In the special case of the involved non-linear functions being (convex-)quadratic, also the software packages mostly famous for their MILP codes, CPLEX, Gurobi and Xpress, now include MINLP codes.

In the following sections, after having introduced Nonlinear Programming, we will sketch some of the algorithms implemented in the above MINLP codes. We will see their application to a class of practical problems in Section 2.3. MILP often plays a role as a building block. Also MINLP algorithms are driven by the need for good dual bounds, the general theme of Mixed Integer Programming. The reader is referred to [BKL+13] and [LL11a] for a comprehensive survey and further collections on MINLP.

1.2.1 Nonlinear Programming

Just like the continuous relaxation of an MILP gives rise to a well-studied problem class, namely LP, so does the one of an MINLP. Dropping the integrality require-

ments in (1.18) - (1.21) leads to the general Nonlinear Programming (NLP) problem we consider here,

$$\begin{aligned} \min \quad & f(x) && && (1.22)\\ \text{s.t.} \quad & g_i(x) \leq 0 && \forall\ i = 1, \ldots, m_1 && (1.23)\\ & h_j(x) = 0 && \forall\ j = 1, \ldots, m_2 && (1.24)\\ & x \in \mathbb{R}^n. && && (1.25) \end{aligned}$$

NLP has been studied basically for as long as LP, and identical as in the case of an MINLP, the difference between convex and non-convex NLPs can be made. Convex NLPs are easier to handle because roughly speaking, any local minimum of a convex function is automatically a global one. This is the reason why often in the context of (MI)NLPs, the distinction between the attributes *locally optimal* and *globally optimal* is made when talking about solutions. General-purpose algorithms that have been proposed for NLPs can again be divided into two categories. There are algorithms that converge to local minima, for example the very popular class of interior-point algorithms. It should be clear from what was said before that such an algorithm automatically solves a convex NLP to global optimality. Also, such algorithms can be implemented efficiently in practice, and respective codes are often called local NLP solvers. For non-convex NLPs instead, spatial branching is usually required in order to solve a problem, and we will illustrate this concept in Section 1.2.4. Such approaches are exploited in the second category of algorithms for NLP, that attempt to always converge to a global optimal solution, regardless of whether the NLP at hand is convex or non-convex.

A powerful theoretical tool, that is general enough for both the convex and non-convex case of NLPs, are the so-called Karush-Kuhn-Tucker (KKT) conditions. We state them in the following result that can be found, e.g., in [BV04].

Theorem 1.6. *Assume that x^* is a local minimum of* (1.22) - (1.25) *that satisfies a constraint qualification*[4]*, and that the involved functions are continuously differentiable. Then, there exist constants μ_i, $i = 1, \ldots, m_1$, and λ_j, $j = 1, \ldots, m_2$, such that*

$$\begin{aligned} & \nabla f(x^*) + \sum_{i=1}^{m_1} \mu_i \nabla g_i(x^*) + \sum_{j=1}^{m_2} \lambda_j \nabla h_j(x^*) = 0 && && (1.26)\\ & g_i(x^*) \leq 0 && \forall\ i = 1, \ldots, m_1 && (1.27)\\ & h_j(x^*) = 0 && \forall\ j = 1, \ldots, m_2 && (1.28)\\ & \mu_i \geq 0 && \forall\ i = 1, \ldots, m_1 && (1.29) \end{aligned}$$

[4]This is a technical condition of which several versions exist [BV04].

$$\mu_i \cdot g_i(x^*) = 0 \qquad \forall\ i = 1, \ldots, m_1. \tag{1.30}$$

The constants μ_i and λ_j are called KKT multipliers. The KKT conditions are necessary conditions for a point x^* to be a local minimum, and any point satisfying them is called a KKT point. Since the conditions are, however, not sufficient in general, not any KKT point is a local minimum. There is a set of sufficient conditions, also called second-order conditions, because they involve second derivatives of the functions. It turns out that these conditions are automatically satisfied in the case of a convex NLP[5], and thus, in a convex NLP, a KKT point is automatically a local and hence global minimum. In the special case of an LP, the KKT conditions reduce to the complementary slackness conditions (1.17), and the KKT multipliers correspond to the dual variables. Interior-point algorithms, like the one implemented in IPOPT [WB06], solve an NLP to a KKT point, and are also able to return the KKT multipliers. We will use the KKT conditions in Section 2.6 when deriving valid nonlinear inequalities for certain MINLPs.

1.2.2 Nonlinear branch & bound

We now discuss a first algorithm for an MINLP, that we assume to be convex for the moment. It can be seen as the most straightforward extension of the branch-and-bound concept, that has been proven to be very successful in the context of MILP, to MINLP. Basically, a nonlinear branch-and-bound algorithm works like Algorithm 1.1. The only conceptual difference is that instead of being LP-relaxations of the subproblems of the underlying MILP, the nodes correspond to NLP-relaxations of the subproblems of the MINLP. These can be solved to global optimality efficiently, since they are convex NLPs.

Remark 1.7. *Most notions introduced in the context of branch & bound for MILP can be naturally extended to nonlinear branch & bound, like dual bounds and gaps for example. In particular, we can readily extend Definition 1.3, noting that a leaf problem of an MINLP is an NLP.*

Nonlinear branch & bound is implemented, for example, in BONMIN[6]. It should be clear by now that any such implementation would use, as a building block for solving the NLP-relaxations, a local NLP solver. If these NLPs were non-convex, such a solver would return a not necessarily globally optimal solution. Thus, we can explain what happens when a nonlinear branch-and-bound implementation is applied to a non-convex MINLP. Assume that in that case, an NLP-relaxation is

[5] provided the involved functions are twice continuously differentiable

[6] among other algorithms for convex MINLPs

solved to a local minimum with value $\hat{z}$, while its global minimum value, say $\tilde{z}$, is strictly lower. If now $\tilde{z} < UB \leq \hat{z}$, the node will be pruned, even though we have no guarantee that the optimal solution to the entire MINLP is not contained inside the feasible region of the subproblem of this node. If so and if the node is pruned, the optimal solution will never be found during the tree-search. Yet, it is not excluded that other sub-optimal feasible solutions are found, and so nonlinear branch & bound acts as a heuristic algorithm, when applied to a non-convex MINLP. It can find feasible, but not necessarily optimal solutions. We will see an example of a practical application where this use of nonlinear branch & bound is made in Section 2.3.1.

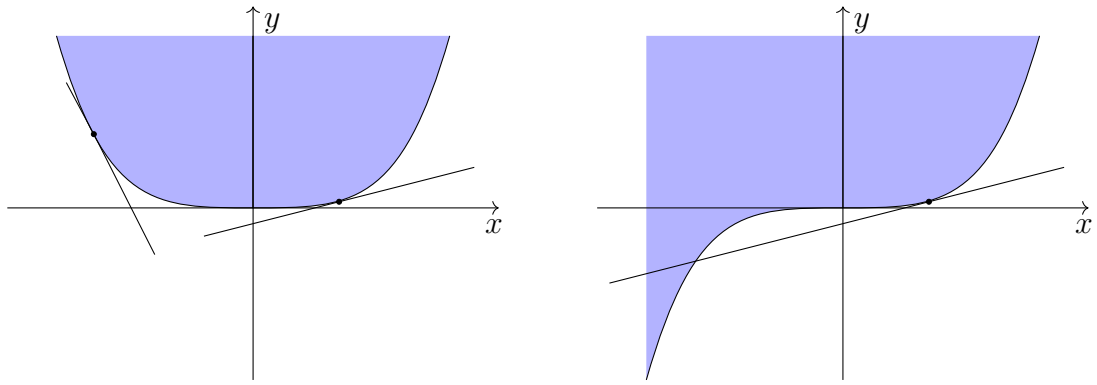
We note that, as in the context of MILP, valid inequalities could be generated after a node's NLP-relaxation has been solved during the tree-search. There is literature on the topic of general-purpose MINLP cutting planes, see, e.g., [Bon11], but the field is much less mature than in MILP. In Section 2.6 we will study valid inequalities for a specific non-convex MINLP that rely, however, very much on the problem structure.

1.2.3 Outer approximation

We now turn to the notion of outer approximations and the algorithms that are based thereon. Outer approximation is, like nonlinear branch & bound, a concept that is originally intended for optimization problems involving convex nonlinear functions. The basic idea of outer approximation can be illustrated geometrically. Take the feasible set that is described by the single constraint $g(x) \leq 0$, and consider the first-order Taylor approximation of the function $g(x)$ at some point $\tilde{x}$,

$$Tg_{\tilde{x}}(x) = g(\tilde{x}) + \nabla g(\tilde{x})^T(x - \tilde{x}).$$

Since $g(\cdot)$ is assumed to be convex, $Tg_{\tilde{x}}(x) \leq g(x)$, and thus $Tg_{\tilde{x}}(x) \leq 0$ describes a relaxation of the initial feasible set. Note that the first-order Taylor approximations are always linear inequalities, and such a relaxation is a polyhedral set. An example involving a bivariate function of the form $g(x, y) = |x|^\kappa - y$, that corresponds to the epigraph of a simple power function, and that is convex if we assume $\kappa \geq 1$, is shown in Figure 1.2 (a). The idea of outer approximation is thus to approximate the feasible region of the continuous relaxation of a convex MINLP by these so-called outer approximation cuts. Together with the integrality requirements on the variables, one obtains an MILP-relaxation of the initial MINLP. A strong theoretical and algorithmically handy result states that if the set of points, at which the outer approximation cuts are generated, is the in some sense correct one, the optimal



(a) The epigraph of a power function (b) The epigraph of a signpower function

Figure 1.2: Outer approximation cuts of non-polyhedral sets

solution of this MILP-relaxation is also optimal to the MINLP [DG86, FL94].

The original version of the outer approximation algorithm for a convex MINLP is sometimes referred to as a multi-tree method, because in each iteration an MILP-relaxation, obtained in the way described above, is solved by exploring an entire search tree. Then, the (same) MILP is slightly modified by adding outer approximation cuts and again solved by exploring another search tree from scratch. The LP/NLP-based branch-and-bound algorithm [QG92], which is implemented for example in FilMINT and BONMIN, aims at reducing this approach into a single search tree. Instead of solving every single MILP-relaxation to optimality and then adding the outer approximation cuts generated at the optimal solution, in every integer feasible node of an initial MILP-relaxation, the corresponding leaf problem of the MINLP is solved to optimality, and the outer approximation cuts generated at that optimal point are added[7]. The LP-relaxation of the current node in the MILP-relaxation with the newly added cuts is then resolved and the tree-search continues.

Again, we spend some time on analyzing what happens when an algorithm, that is based on outer approximation cuts, is applied to a non-convex MINLP. As an example take a slight modification of the previous one, $\tilde{g}(x, y) = \text{sign}(x)|x|^\kappa - y$, corresponding now to the epigraph of a so-called signpower function, that is not convex even if $\kappa > 1$. As can be seen in Figure 1.2 (b), there is no guarantee that an outer approximation cut preserves all feasible solutions. Thus, like nonlinear branch & bound, outer approximation algorithms are heuristic when applied to non-convex MINLPs.

In a mixed integer setting, certain special cases involving the signpower function

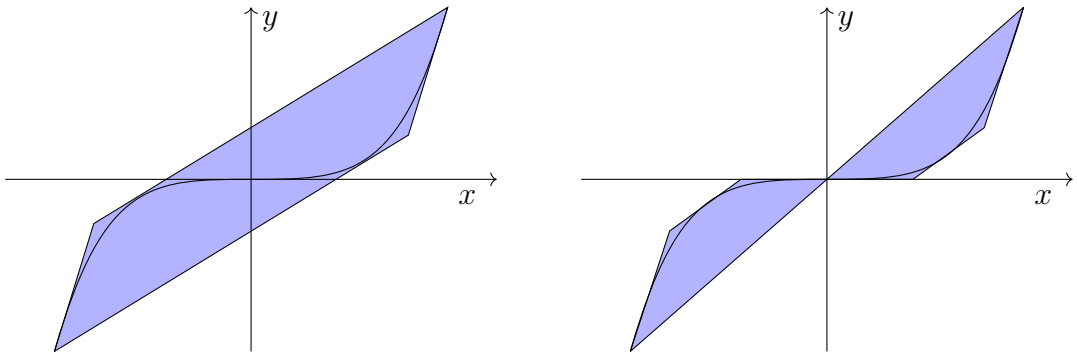
[7] If the leaf problem is infeasible, one can add the outer approximation cuts corresponding to a point that minimizes the infeasibility.

can be cast in the light of convex MINLPs. In fact, this overall non-convex function is actually convex on the positive half-axis of its domain, and concave on the negative half-axis. In Section 2.3.1 we will see a modification of the LP/NLP-based branch and bound that exploits this property and thus remedies a situation similar to the one depicted in Figure 1.2 (b) in the context of a certain MINLP arising from a practical application in the context of water networks. Nevertheless, if the signpower function appears in equality constraints in the underlying MINLP, these become non-convex, as we will see in the next section.

1.2.4 Spatial branch & bound

As mentioned, solving non-convex NLPs to global optimality is a difficult task in general. It is, in fact, NP-hard [SA13, TS02]. One way to do so is to use spatial branching. The basic idea of such an algorithm remains to iteratively divide the problem into subproblems and to solve (usually linear) relaxations of these. The division of subproblems though is done not by branching on integer variables, but by branching on continuous ones. In other words, the continuous domain of some nonlinear function is divided at some breakpoint into two smaller domains, thus creating two new subproblems. Provided that the relaxation of this nonlinear constraint becomes tighter when the domain of the corresponding nonlinear function is reduced, spatial branching gradually refines the relaxations. In the case of a constraint of the form $\text{sign}(x)|x|^\kappa - y = 0$, this is depicted in Figure 1.3. Branching is continued until the relaxations are tight enough to provide solutions that are ϵ-feasible to the original problem for some small ϵ. Integrating spatial branching with mixed integer branching techniques opens the possibility of solving non-convex MINLPs to global optimality. Most non-convex MINLP codes, e.g., would start branching on integer variables and apply spatial branching once all of them have been fixed, i.e., in the leaf problems of the search tree.

One way to obtain relaxations of the subproblems of a non-convex MINLP is to use reformulation techniques aiming for a decomposition of all nonlinear functions into some basic ones. For these basic functions, linear relaxations are then readily available. The tightness of such relaxations and thus the performance of the algorithm are highly dependent on the bounds of the domains of the nonlinear functions in a subproblem, as can be seen directly in Figure 1.3. Therefore, so-called bound-tightening or domain-reduction techniques play an important role in the context of non-convex MINLPs. We will come back to the importance of domain reductions in other contexts in Sections 1.3 and 3.6.



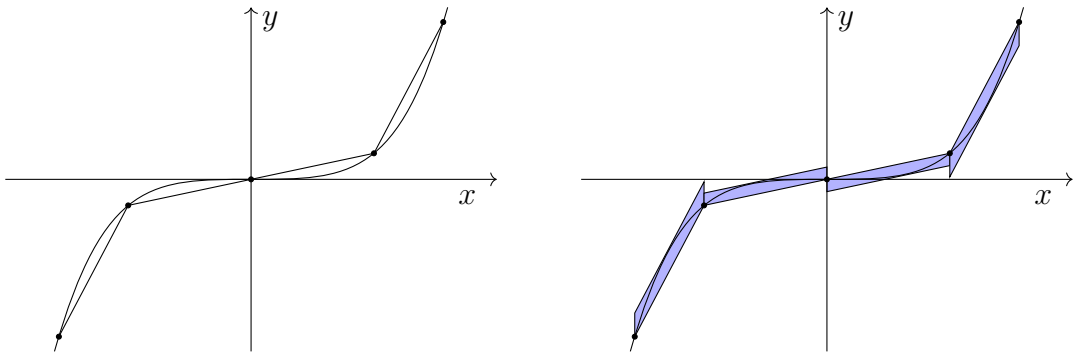
(a) Initial polyhedral relaxation (b) Relaxation after branching on the origin

Figure 1.3: Spatial branching for a signpower function

1.2.5 Piecewise linearizations

Another way to approach MINLPs, that is motivated by the success of MILP algorithms, is to approximate all nonlinearities by piecewise linear functions. The catch is that a piecewise linear function can be modeled by auxiliary binary variables and linear constraints, which opens the possibility of applying MILP solvers as black-boxes to an approximated MINLP. It has to be clear though that this is only an approximation of the original problem. Therefore, when the MINLP at hand is convex, there are more attractive methods, like the ones discussed in Sections 1.2.2 and 1.2.3, toward which one usually orients. However, when dealing with non-convex MINLPs, piecewise linear approximations become an intriguing alternative to expensive and sometimes ineffective MINLP methods. The initial motivation of the analysis that we will present in Chapter 2 arises in this context.

Piecewise linear approximations roughly work as follows. Imagine the domain of a univariate function to be partitioned into several intervals. That function can then be approximated by the line segments connecting the ordinates of the end points of these intervals (the breakpoints), and there is an auxiliary binary variable for each interval, controlling which of the several slopes is in use. Figure 1.4 (a) depicts this situation in the case of a signpower function. This simple point of view already makes clear that the accuracy of such an approximation depends on the number of breakpoints that are used. There are several methods to model univariate piecewise linear functions, such as the so-called incremental method [HMM57] or the convex combination method [Dan60]. For a comparison of different piecewise linear modeling techniques, particularly applied to the underlying applications of Chapter 2, see [GMMS12]. It is also in this context that topics related to special ordered sets of type 2 and therefore developed branching rules came up [BT70]. The piecewise linear modeling techniques have also been extended to multivariate func-



(a) Piecewise linear approximation (b) Piecewise linear relaxation

Figure 1.4: Piecewise linearizations of a signpower function

tions. Instead of intervals, the domain of such a function is usually partitioned into several simplices, and over each simplex the function is approximated by an affine function. Some kind of curse of dimension is encountered here because the number of required simplices, that is directly connected to the number of auxilary binary variables in a resulting MILP formulation, is strongly growing with the dimension of the function's domain. Therefore we often find reformulation ideas that increase the number of variables and constraints, but decrease the dimension of the functions to be approximated. A relatively new approach to piecewise linearly approximate higher-dimensional nonlinear functions is proposed in [RDLM14].

Yet, one has to keep in mind that all the above leads to an approximation of the original problem only. This means that the obtained globally optimal solution of the MILP is not necessarily globally or locally optimal for the MINLP. Even worse, it might not even be feasible to the nonlinear constraints. A way to estimate a priori the approximation error and to refine the latter until it stays inside some desired predefined error bound is proposed in [GMMS12]. The authors also show how to use piecewise linear functions to create relaxations of the MINLP instead of approximations only. Basically, the maximum error of an approximation over each simplex is overestimated as tightly as possible, and a continuous variable, bounded by this maximum error, is added to the ordinate. The resulting relaxation of this procedure is a so-called ε-tube, depicted in Figure 1.4 (b).

There has also been some effort to integrate MILP and NLP concepts in the context of piecewise linearizations. Very intuitively, by fixing the integer variables to the values of a globally optimal MILP solution, one could solve the remaining leaf problem of the MINLP to local optimality. This would guarantee at least the feasibility of the solution, but not necessarily its global optimality. For a more detailed idea of how to integrate MILP and NLP in the context of piecewise linear approximations in the context of water networks, the topic of Chapter 2, see [KL12].

Figures 1.3 and 1.4 show one important difference between spatial branching and piecewise linear relaxations. In the former case, the refinement of the approximating polyhedra is part of the branching, whereas in the latter, the fragmentation of the respective domain is given a priori and is not changed during the optimization process. However, there has recently been some attempt to overcome this difference by extending the concept of piecewise linear relaxations to an algorithm where the approximation is refined dynamically in order to compute a globally optimal solution for the originally non-convex MINLP. This approach has been investigated in the context of gas networks in [GMS13]. Therein is also contained a computational comparison between the proposed approach exploiting piecewise linear approximations and several solvers that use spatial branching techniques.

The initial motivation of a large portion of Chapter 2 was the analysis of piecewise linearizations in the context of water networks, and we will discuss the success of such approaches in that special case in Section 2.4.

1.3 Constraint Programming

Identifying a canonical or standard form of a CP can be considered a less trivial task than it is in the case of an MILP or MINLP. This is somehow due to the modeling flexibility it provides. CP can be described as the process of finding an assignment of values to a set of variables (a solution) that satisfies a set of arbitrary constraints. A variable x is usually assigned a domain $D(x)$, that again is somewhat arbitrary. Different categories of domains can be individualized in CP, for example integer domains or, related to that, general finite domains given by arbitrary finite sets. As well so-called interval variables, whose domains are subsets of the set of all intervals on the real line, have been introduced.

"A constraint can be anything" is indeed a completely informal affirmation, but highlights the modeling flexibility by which CP is characterized. A constraint can be seen as the subset of values that, when assigned to the involved variables, satisfy a certain condition. In this light, instead of giving a general and formal definition, we choose to present an example of a constraint that is quite popular in the CP literature, namely the `alldifferent` constraint [vH01].

Definition 1.8. *Let $x_1, \dots, x_n$ be variables with finite domains $D(x_1), \dots, D(x_n)$. Then,*

$$\texttt{alldifferent}(x_1, \dots, x_n) := \{(v_1, \dots, v_n) \mid v_i \in D(x_i), v_i \neq v_j \ \forall\ i \neq j\}.$$

Thus, `alldifferent` represents a way to describe that a series of variables have to

take pairwise different values. The constraints encountered in CP can be divided into several categories, and CP implementations like IBM ILOG CPO [CPX] provide the user with libraries for each category.

Mixed integer constraints, that are used in MIP, can be seen as one of these constraint categories. One could therefore be tempted to claim that MIP is a subclass of CP. However, no objective function is traditionally present in CP, meaning that often feasibility problems are solved. Even though it is possible to include objective functions, the common point of view is to see MIP and CP as different paradigms. This is mostly due to the algorithmic concepts that are behind the two. Although in both cases branching plays an indispensable role, MIP algorithms, as we have seen, heavily rely on the objective function through the use of dual bounds. In CP instead, the concept of a dual bound does generally not exist. While MIP is driven by the goal of closing the dual gap, CP is driven by the aim of reducing the domains of the variables until they are reduced to single values. This can be achieved by so-called filtering and propagation algorithms, treated further down. Nevertheless, as anticipated in the introduction, there has been considerable effort to integrate MIP and CP techniques [MT04], which is also a subject in this thesis. The modeling flexibility of CP for example will motivate our actions in Chapter 4.

An important part of the machinery used in CP are filtering algorithms. Values, whose assignments to the variables result in a violation of a given constraint, are called inconsistent. A filtering algorithm associated with that constraint is an algorithm that is able to remove inconsistent values from the domains of the involved variables.

Example 1.9. *Consider the variables x_1, x_2, x_3, x_4 with domains*

$$D(x_1) = D(x_2) = \{1, 2\},\ D(x_3) = \{3\} \text{ and } D(x_4) = \{1, 2, 3, 4\}.$$

Further suppose that `alldifferent`(x_1, x_2, x_3, x_4) *is imposed. Any intelligent filtering algorithm will remove the value 3, to which the variable x_3 has been fixed, from the domains of the other variables, if present. That is, 3 will be removed from $D(x_4)$. Based on slightly more complicated reasoning, it should also remove the values 1 and 2 from therein, because these values will certainly be assigned to x_1 and x_2.*

Filtering algorithms give rise to the notion of propagation. Anytime a value has been removed from the domain of a variable x, this might cause values in the domains of other variables, that are linked to x via some constraint, to be inconsistent. Thus, a filtering algorithm might be able to further reduce some domains. Propagation is the process of executing filtering algorithms associated with constraints in which x appears whenever the domain of x changed. In this way, such a domain change is

"propagated" to the variables via the constraints.

As we have sketched in Section 1.2.4, the concept of domain reduction is important in the context of non-convex MINLPs, and respective algorithms make use of the concept of propagation. This is often considered an important intersection point of MIP and CP, and the former can learn from the latter. We will argue in Section 3.6 that propagation might be equally important in the context of certain MILPs and analyze its use inside an MILP-based algorithm for a subclass of MILP problems.

1.4 Disjunctive Programming

A notion that is frequently encountered in the context of MILP or LP, but also MINLP, is Disjunctive Programming. One could say that Disjunctive Programming is a programming paradigm on its own, although there do not seem to be any widely spread codes for it. That is, there is no such thing as a DP solver. Yet, this notion has been highly influential in different areas of Mathematical Programming. It has been originally proposed by Balas [Bal79, Bal98] and in that original form can be described as the minimization of a linear function over the union of polyhedra, i.e., sets that can be described as the intersection of sublevel sets of affine functions. Dropping this restriction and extending the point of view to general convex functions, Disjunctive Programming becomes the minimization of a convex function over the union of convex sets. Balas' results have been extended in this direction in a seminal paper by Ceria and Soares [CS99].

The key ingredient to be able to optimize over the union of sets is the ability of describing the convex hull of this union. If this can be done by the sublevel sets of convex functions, then such a program can be stated as a convex NLP. The main result in [CS99] is precisely a theorem that shows how to build the convex hull of the union of a finite number of sets, each one of which is described by convex functions. Before stating that theorem we need to introduce the so-called perspective function, which plays an important role in it.

Definition 1.10. *For a given closed convex function $g : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, the perspective function $\tilde{g} : \mathbb{R}^{n+1} \to \mathbb{R} \cup \{\infty\}$ is defined as*

$$\tilde{g}(x, \lambda) = \begin{cases} \lambda \cdot g\left(\frac{x}{\lambda}\right), & \lambda > 0 \\ \infty, & \lambda \leq 0 \end{cases}. \tag{1.31}$$

The perspective of a closed convex function is known to be convex, but not necessarily closed. This issue will be discussed further down. For now, we have all the

required elements to state the main result of [CS99].

Theorem 1.11. *Let* $C_j = \{x \in \mathbb{R}^n \mid g_{ji}(x) \leq 0, i = 1, \ldots, \ell_j\} \neq \emptyset$*, for* $j \in \mathcal{J}$*, and assume each* $g_{ji} : \mathbb{R}^n \to \mathbb{R}$ *is a closed convex function. Then, we have that* $\operatorname{conv}(\cup_{j\in\mathcal{J}} C_j) = \operatorname{proj}_x \operatorname{cl}(C)$*, where*

$$C = \left\{ \begin{array}{ll} (x, x^1, \ldots, x^{|\mathcal{J}|}, \lambda_1, \ldots, \lambda_{|\mathcal{J}|}) \in \mathbb{R}^{(|\mathcal{J}|+1)n+|\mathcal{J}|} & \\ x = \sum_{j\in\mathcal{J}} x^j & \\ \tilde{g}_{ji}(x^j, \lambda_j) \leq 0 & \forall\ i \in [\ell_j],\ j \in \mathcal{J} \\ \sum_{j\in\mathcal{J}} \lambda_j = 1 & \\ \lambda_j \geq 0, & \forall\ j \in \mathcal{J} \end{array} \right\}.$$

The construction of Theorem 1.11 begins in a straightforward manner by creating copies x^j of the initial vector x, each one constrained to lie in one of the disjunctive sets C_j. Then, the original x is a convex combination of these copies with weights λ_j. In this way, the resulting bilinear terms $\lambda_j x^j$ lead to a set description of the convex hull that is, however, non-convex. A simple transformation is then used to obtain the convex set C containing the perspective function. A big effort in [CS99] is the insight that the closure of the perspective function also captures the elements in the convex hull that have zero weights λ_j for some $j \in \mathcal{J}$.

Although Theorem 1.11 provides a complete formal description of the convex hull of the union of the convex sets we are concerned with, it bears two main practical difficulties. First, the closure of the perspective function does not have an algebraic representation in general. In other words, the perspective function becomes non-differentiable for $\lambda_j \to 0$. This is a significant difference with respect to the affine case, where differentiability with respect to λ_j can be recovered by trivial algebra.

The second difficulty associated with the practical use of Theorem 1.11 is the fact that the initial set is lifted into a space with a multiple dimension because a copy of the initial space is created for each disjunctive set. Note that the set C in Theorem 1.11 is a subset of $\mathbb{R}^{(|\mathcal{J}|+1)n+|\mathcal{J}|}$, i.e., it is defined in a space whose dimension makes the optimization over it computationally challenging. There are special cases of Theorem 1.11 in which the convex hull can be projected onto the original space of variables explicitly. This will be the main topic of Chapter 3.

In the affine case, the theory of Disjunctive Programming also provided the foundation of lift-and-project cutting planes [BCC93, BCC96, FLT11], where the problem of optimizing a linear function over the union of polyhedra is solved as a separation routine for devising cutting planes for MILPs. The relation between Disjunctive Programming and the separation of cutting planes has its roots in Farkas' Lemma, see, e.g., [Sch98]. Roughly speaking, one can give a polyhedral description

of the set of all valid linear inequalities for the set of feasible points that in addition satisfy a certain disjunction. We formally state these considerations in the following result, that can be found in [Bal98].

Theorem 1.12. *Let $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $W_j \in \mathbb{R}^{m_j\times n}$, $d_j \in \mathbb{R}^{m_j}$ $\forall\ j \in \mathcal{J}$, and denote $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$. The linear inequality $\alpha^T x \geq \beta$ is valid for (the convex hull of) the set $\cup_{j\in\mathcal{J}}\{x \in P \mid W_j x \geq d_j\}$ if and only if there exist $\theta_j \in \mathbb{R}^m$ and $\tau_j \in \mathbb{R}^{m_j}_+$ such that*

$$\alpha \geq A^T\theta_j - {W_j}^T\tau_j \quad \text{and} \quad \beta \leq b^T\theta_j - {d_j}^T\tau_j \quad \forall\ j \in \mathcal{J}.$$

We will use a special version of this Theorem in Section 4.2.2 in order to prove the validity of an extended version of a certain type of lift-and-project cuts for MILP. Lift-and-project cuts are an ingredient of many existing commercial and open-source MILP solvers, like the ones we mentioned in Section 1.1, and they are part of the default setting of CPLEX [Tra13].

In the nonlinear case, the numerical difficulties mentioned above were already noticed by Stubbs and Mehrotra [SM99] in the process of designing a branch-and-cut algorithm for general 0–1 mixed convex programming problems, based on lift-and-project cuts separated through an alternative version of Theorem 1.11. It is worth noting that for more than ten years there was no significant progress in the separation of lift-and-project cuts for general mixed integer convex programming problems. More recently, two important steps toward the effective use of disjunctive cuts for mixed integer convex programming have been made [Bon11, Kıl11, KLL10], which circumvent the non-differentiability issue algorithmically through the solution of some (potentially many) easier optimization problems. The reader is referred to [BLL11b] and [BLL+11a] for recent surveys on disjunctive cuts for MINLPs and applications and extensions of disjunctive inequalities, respectively.

1.5 Examples of Mixed Integer Programs

The purpose of this section is to introduce a few examples of problems that can be modeled by MIP. They are mostly MILPs, and we will come back to most of them in later chapters. In Chapter 2 is discussed a special class of MINLP models and a specific model can be found in Section 2.3.1.

1.5.1 TSP with (multiple) time windows

Many problems in the field of combinatorial optimization have been modeled by the means of MI(L)P. The probably most extensively studied one is the so-called

(asymmetric) Traveling Salesman Problem (TSP). This problem calls for finding a least-cost Hamiltonian tour on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. We give a concise description of the problem variant that will play a role later on. We use the so-called Hamiltonian path version: we assume the existence of two designated nodes p and q and seek for a Hamiltonian path that starts at p and ends at q. We also consider the case in which the arrival time at each node i is constrained to lie in some interval $[l_i, u_i]$, leading to the TSP with time windows (TSPTW). Denoting the travel time from i to j by t_{ij}, the process time at i by p_i and the set of incoming and outgoing arcs of node i by δ_i^- and δ_i^+, respectively, a valid MILP formulation, that has been studied in [DGLT12][8], is given by

$$\min \quad \sum_{\substack{(i,j)\in\mathcal{A} \\ i\neq q, j\neq p}} t_{ij} x_{ij} \tag{1.32}$$

$$\text{s.t.} \quad \sum_{j\in\delta_i^+} x_{ij} = 1 \qquad \forall\, i \in \mathcal{N} \setminus \{q\} \tag{1.33}$$

$$\sum_{i\in\delta_j^-} x_{ij} = 1 \qquad \forall\, j \in \mathcal{N} \setminus \{p\} \tag{1.34}$$

$$a_i + p_i + t_{ij} \leq a_j + M_{ij} \cdot (1 - x_{ij}) \qquad \forall\, (i,j) \in \mathcal{A} : i \neq q \tag{1.35}$$

$$l_i \leq a_i \leq u_i \qquad \forall\, i \in \mathcal{N} \tag{1.36}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in \mathcal{A} : i \neq q, j \neq p. \tag{1.37}$$

The variable a_i models the arrival time at node i[9] and the binary variable x_{ij} indicates whether the path contains the arc (i,j). The constraints (1.35) are so-called big-M constraints and include some "large" constants M_{ij}. They work by making the large constant vanish whenever the involved binary variable x_{ij} is equal to one, thus imposing an actual constraint. Otherwise, when x_{ij} is equal to zero, the magnitude of M_{ij} make the inequality a redundant constraint. In other words, the big-M constraint imposes the logical condition

$$[x_{ij} = 1] \implies [a_i + p_i + t_{ij} \leq a_j]. \tag{1.38}$$

Big-M constraints are widely used to formulate such logical implications in MIP, which is one of the topics of Chapter 3. We will show in more detail in Section 3.1 how big-M constraints work, and how appropriate values for the large constants can be determined in a specific MIP. Some of the following models also contain big-M constraints, and the big-M values therein are just assumed to be reasonably large

[8] Therein, no process times are used.

[9] These are usually fixed at p and q to the start and end date of some time horizon $[0, T]$.

values for now. In Chapter 3, we will also consider the above MILP model for the TSPTW.

One can extend model (1.32) - (1.37) to the TSP with multiple time windows (TSPMTW). The time window associated with a node is now not anymore an interval, but the union of several, non-intersecting ones. That is, the arrival time of node i has to lie in the non-contiguous set $\cup_{d=1}^{L_i}[l_i^d, u_i^d]$. A MILP formulation for the TSPMTW, using another set of big-M constraints for imposing the multiple time windows, was proposed in [BHL14][10]. System (1.32) - (1.37) can be extended in the same way by simply adding the constraints

$$l_i^d \leq a_i + M_{id} \cdot (1 - w_{id}) \qquad \forall\ d \in [L_i],\ i \in \mathcal{N} \setminus \{p, q\} \tag{1.39}$$

$$a_i \leq u_i^d + M'_{id} \cdot (1 - w_{id}) \qquad \forall\ d \in [L_i],\ i \in \mathcal{N} \setminus \{p, q\} \tag{1.40}$$

$$\sum_{d=1}^{L_i} w_{id} = 1 \qquad \forall\ i \in \mathcal{N} \setminus \{p, q\} \tag{1.41}$$

$$w_{id} \in \{0, 1\} \qquad \forall\ d \in [L_i],\ i \in \mathcal{N} \setminus \{p, q\}. \tag{1.42}$$

We will analyze this constraint structure and the TSPMTW when we study variables with non-contiguous domains in Section 4.1.1.

1.5.2 Scheduling problems

Another classical problem that has been modeled by MILP is Job Shop Scheduling (JSS). Not only is it theoretically NP-hard, it has also proven to be one of the computationally most stubborn combinatorial optimization problems. We have jobs $j \in [n]$ that all have to be executed on each of a set of given machines $k \in [m]$. The execution of a certain job on a certain machine is also called an operation. We denote the operation of job i on machine k by (i, k) and the corresponding process time by p_{ik}. Each job is further characterized by its proper routing on the machines, defined by matrix $O \in [m]^{n \times m}$, i.e., O specifies the order in which a single job has to be executed on the different machines. More precisely, O_{ik} is the sequence number of machine k in the order of operations of job i. Further, the machines are disjunctive resources, i.e., they can process only one job at a time, and no preemption is allowed. The objective of JSS is to minimize the overall makespan Ω. A MILP formulation, again including big-M constraints, is given by

$$\min \quad \Omega \tag{1.43}$$

$$\text{s.t.} \quad s_{ik'} \geq s_{ik} + p_{ik} \qquad \forall\ i \in [n], k, k' \in [m] :\ O_{ik} < O_{ik'} \tag{1.44}$$

[10] The MILP formulation therein is actually valid for the vehicle routing version of the problem and is thus more general.

$$s_{ik} + t_i \leq s_{jk} + M_{ij}^k \cdot (1 - x_{ij}^k) \qquad \forall\ i < j \in [n],\ k \in [m] \qquad (1.45)$$

$$s_{jk} + t_j \leq s_{ik} + \tilde{M}_{ij}^k \cdot x_{ij}^k \qquad \forall\ i < j \in [n],\ k \in [m] \qquad (1.46)$$

$$\Omega \geq s_{ik} + p_{ik} \qquad \forall\ i \in [n], k \in [m] \qquad (1.47)$$

$$s_{ik} \geq 0 \qquad \forall\ i \in [n], k \in [m] \qquad (1.48)$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall\ i < j \in [n], k \in [m]. \qquad (1.49)$$

The variable s_{ik} models the start time of operation (i, k), and the binary variables x_{ij}^k indicate whether i is executed on k before j or vice versa. Also this MILP formulation will be of interest in Chapter 3.

JSS is sometimes called the "mother" of all scheduling problems, and we will study another scheduling problem in Section 4.1.1, namely the Multi-mode Resource Leveling problem [CLS10]. Again, we are given a set of non-preemptable jobs $j \in [n]$ with a set of precedence relations $E \subseteq [n] \times [n]$, and a set of renewable resources $\mathcal{R}$. Furthermore, a job i may be run in exactly one mode $\ell \in \mathcal{M}_i$, and in that case requires $p_{i\ell}$ time units and $r_{i\ell k}$ units of resource $k \in \mathcal{R}$. Each resource k is assigned a unit cost c_k. Fixing a time horizon $[0, T]$, the problem calls for determining the resource capacities R_k that have to be provided, so that all jobs can be executed within the time horizon but without exceeding these capacities, minimizing their overall cost. A MILP model, that can be found in [CLS10][11], is given by

$$\min \quad \sum_{k \in \mathcal{R}} c_k R_k \qquad (1.50)$$

$$\text{s.t.} \quad \sum_{t=0}^{T} \sum_{\ell \in \mathcal{M}_i} x_{i\ell t} = 1 \qquad \forall\ i \in [n] \qquad (1.51)$$

$$\sum_{t=0}^{T} \sum_{\ell \in \mathcal{M}_i} t \cdot x_{i\ell t} = s_i \qquad \forall\ i \in [n] \qquad (1.52)$$

$$\sum_{t=0}^{T} \sum_{\ell \in \mathcal{M}_i} (t + p_{i\ell}) \cdot x_{i\ell t} = e_i \qquad \forall\ i \in [n] \qquad (1.53)$$

$$e_i \leq s_j \qquad \forall\ (i, j) \in E \qquad (1.54)$$

$$\sum_{i \in [n]} \sum_{\ell \in \mathcal{M}_i} r_{i\ell k} \sum_{\substack{\tau = t - p_{i\ell} + 1 \\ \tau \geq 0}}^{t} x_{i\ell\tau} \leq R_k \qquad \forall\ k \in \mathcal{R},\ t \in [0, T] \qquad (1.55)$$

$$x_{i\ell t} \in \{0, 1\} \qquad \forall\ t \in [0, T],\ \ell \in \mathcal{M}_i,\ \forall\ i \in [n]. \qquad (1.56)$$

Variables s_i again account for the start times of the jobs, but we also model their completion time e_i. The binary variable $x_{i\ell k}$ in this model indicates whether a job

[11] The model therein does not account for different modes.

i starts at a certain time t and in mode ℓ. One instance of the above MILP model has been included in MIPLIB 2010 (namely, instance 30n20b8), and we will further investigate it in Section 4.1.1.

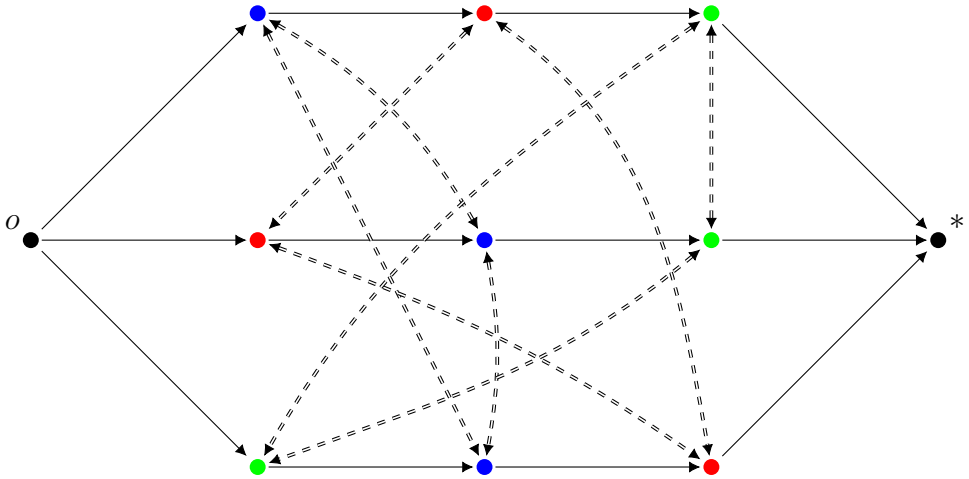
1.5.2.1 Disjunctive graphs in JSS

Of fundamental importance for JSS and for understanding the concepts we will apply in Section 3.6.2 are so-called disjunctive graphs, see, e.g., [BPS00]. In general, a disjunctive graph $\mathcal{G} = (\mathcal{N}, \mathcal{C} \cup \mathcal{D})$ consists of a set of vertices $\mathcal{N}$, a set of conjunctive arcs $\mathcal{C} \subseteq \mathcal{N}^2$ and a set of disjunctive arcs $\mathcal{D} \subseteq \mathcal{N}^2$. Conjunctive arcs are just ordinary arcs. By definition, a disjunctive arc (v, w), instead, has the property that necessarily also $(w, v) \in \mathcal{D}$. In other words, disjunctive arcs can be thought of as pairs of two oppositely directed arcs between two nodes. We require $\mathcal{C} \cap \mathcal{D} = \emptyset$, and the conjunctive graph associated with $\mathcal{G}$ is just $\mathcal{G}_{\mathcal{C}} = (\mathcal{N}, \mathcal{C})$.

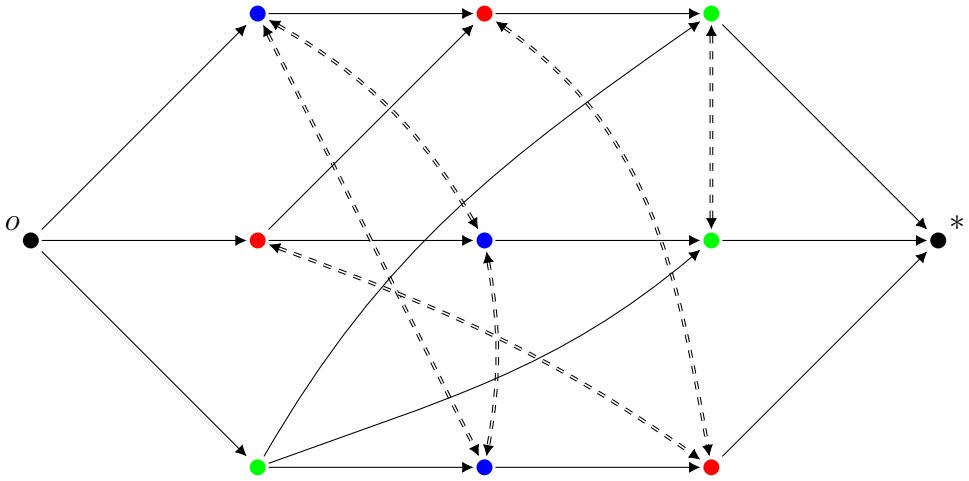
Associated to any of the subproblems of the JSS model (1.43) - (1.49), that can be obtained by fixing a subset of the binary variables x_{ij}^k to zero or one and thus imposing additional precedence relations (including the case in which none of them are fixed), we can build a disjunctive graph in the following way. It contains a vertex for each operation (i, k) plus two artificial vertices, the source o and the sink $*$. For each fixed precedence relation between two operations, either specified in the matrix O, or deriving from a fixed binary variable, there is a conjunctive arc in the graph. For example, if, by abuse of notation, $\nu = (i, k)$ precedes $\varrho = (i, k')$, we have $(\nu, \varrho) \in \mathcal{C}$. The weight of any conjunctive arc (ν, ϱ) is given by p_ν, the process time of the operation corresponding to the tail of (ν, ϱ). By convention, for each job i we have a conjunctive arc from the source to the node $u = (i, k)$ such that $O_{ik} = 1$ (i.e., the first operation of job i), and one conjunctive arc going into the sink from node $w = (i, k')$ such that $O_{ik'} = m$ (i.e., the last operation of job i). All arcs leaving the source node have weight 0.

For each binary variable x_{ij}^k that has not been fixed, there is a pair of disjunctive arcs between the two nodes $u = (i, k)$ and $w = (j, k)$ in $\mathcal{D}$. Disjunctive arcs can also be thought of as edges that still have to be directed, or, in a JSS context, as precedence relations that still have to be established. Note that any two nodes that are connected by a pair of disjunctive arcs correspond to operations that have to be executed on the same machine. Figure 1.5 shows an example of the disjunctive graphs of a JSS instance and one of its subproblems.

A feasible solution to (1.43) - (1.49) or to any of its subproblems can also be described in terms of the disjunctive graph as follows. A pair of disjunctive arcs $(u, w), (w, u)$ is said to be selected if both are removed from $\mathcal{D}$ and either (u, w) or (w, u) is added to $\mathcal{C}$ (meaning that the corresponding binary variable is fixed). A



(a) Original instance



(b) Subproblem with 3 binary fixings

Figure 1.5: Disjunctive graphs with $n = m = 3$

complete selection is a set of selections in a disjunctive graph such that $D = \emptyset$. A feasible solution is given by a complete selection that results in a directed acyclic graph (DAG). The makespan of that solution is the longest path from o to $*$ in the remaining conjunctive graph $\mathcal{G}_C$.

The notion of disjunctive graphs is very useful for calculating lower and upper bounds on the start time of an operation $\nu = (i, k)$ in any subproblem of a JSS instance. These bounds are closely related to the so-called heads and tails in the associated conjunctive graph $\mathcal{G}_C$, denoted by r_ν and q_ν, respectively. Namely, r_ν is defined as the longest path from o to ν in $\mathcal{G}_C$. This quantity represents an amount of time that passes before t can be executed, independently of the binary variables that are yet to be fixed. Thus, r_ν is a valid lower bound on the start time of operation ν

in the subproblem at hand,

$$\underline{s}_{ik} := r_\nu \leq s_{ik}. \tag{1.57}$$

Symmetrically, q_ν is defined as the longest path from ν to the sink $*$ minus p_ν. It represents a lower bound on the time that passes after ν has been executed. Thus, an upper bound on the start time of operation ν in the subproblem can be calculated,

$$\overline{s}_{ik} := UB - q_\nu - p_\nu \geq s_{ik}, \tag{1.58}$$

where UB is an upper bound on the optimal solution value of (1.43) - (1.49). A trivial upper bound is given, e.g., by

$$UB = \sum_{i=1}^{n} \sum_{k=1}^{m} p_{ik}. \tag{1.59}$$

Heads and tails in the context of JSS and the resulting bounds will become important in Section 3.6.2.

1.5.3 Knapsack problems

Another vastly studied and easily explained combinatorial optimization problem is the so-called Knapsack problem [MT90]. We are given a set of items $i \in [n]$, each one equipped with a non-negative profit p_i and a non-negative weight w_i, and a knapsack with capacity C. The aim is to pack a subset of items into the knapsack such that the profit of all packed items is maximized, but their weight does not exceed the capacity. A straightforward MILP formulation, that is actually an ILP, is given by

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq C \\ & x_i \in \{0,1\} \qquad\qquad \forall\, i \in [n]. \end{aligned}$$

Knapsack problems have been studied extensively since the early days of integer programming, and are often found as substructures in MILP. We will study a related ILP in Section 4.1.2.

1.5.4 Supervised classification

At last, we introduce a problem studied in the literature related to machine learning, namely, the so-called supervised classification problem. Consider a set of objects $i \in [m]$, where each object is characterized by the vector $x_i \in \mathbb{R}^d$ and associated with one of two classes, labeled by $y_i \in \{-1, 1\}$. The task is to find a hyperplane $\omega^T x \leq \omega_0$ in $\mathbb{R}^d$ that separates the two classes by maximizing a *confidence margin*. Because it is not always possible to find such a hyperplane, for each object i that is misclassified, one models the violation of the associated constraint $y_i(\omega^T x_i - \omega_0) \leq 0$ with a continuous slack variable ξ_i whose value is penalized in the objective function. Recently, Brooks [Bro11] suggested that the misclassification penalty must be upper bounded, and beyond that bound a fixed penalty must be paid. This again leads to a model with big-M constraints. We give a problem formulation as in [Bro11], namely

$$
\begin{aligned}
\min \quad & \frac{\omega^T \omega}{2} + \frac{C}{m}\left(\sum_{i=1}^{m} \xi_i + 2\sum_{i=1}^{m}(1 - z_i)\right) & \\
\text{s.t.} \quad & y_i(\omega^T x_i + \omega_0) - 1 + \xi_i \geq -M_i \cdot (1 - z_i) & \forall\, i \in [m] \\
& 0 \leq \xi_i \leq 2 & \forall\, i \in [m] \\
& (\omega, \omega_0) \in \mathbb{R}^{d+1} & \\
& z \in \{0, 1\}^m, &
\end{aligned}
$$

where C is some non-negative constant. The quantities x_i and y_i are not variables here, but constants. Instead, ξ_i measures the continuous classification error up to a maximum of 2, at which point the binary variable z_i indicates that i has been misclassified. The continuous variables (ω, ω_0) describe the coefficients and right-hand side of the hyperplane that is to be determined, respectively. The above formulation is a convex MIQP and will play a role in Chapter 3.

2 MILP vs. MINLP insights in water network optimization

Classical network flow problems have been extensively studied as an application of LP [BJS11], basically from its early days on, and later the applications were extended by introducing discrete decisions, thus giving rise to MILPs [MW84, Min89]. The basic common task in almost all variants is to route a flow through a network from a set of sources to a set of sinks, established by imposing linear flow conservation constraints, or balance equations, see (2.1) further down. Such basic models can become coarse when dealing with pressurized water networks, where the fluid is transported in pipes with no air contact and thus possibly varying pressure levels. A first step toward accurately modeling the physical aspects of such networks, is the introduction of pressure variables at nodes in addition to the classical flow variables on arcs. From this perspective, what actually induces a flow between two nodes is explained by a pressure difference. To subsume a broader field of applications, the additional variables in such models are also referred to as node potentials, including as well the electric potential of a point in an electric circuit. Pressure is again an important quantity in gas networks [KHPS15]. The drawback of the resulting accuracy gain is the fact that the relation between flow and potential difference usually leads to nonlinear equations. Consequently, the resulting models are sometimes called nonlinear network flow models [Rag13]. Often, we also have to deal with the presence of discrete decisions that can be made regarding different network elements. In that case, the optimization tasks faced when tackling nonlinear network flow models are put in the context of non-convex MINLP. Several aspects and variants of this modeling approach related to water networks will be discussed in more detail in Section 2.1.

In this chapter we focus on topics related to the optimization of drinking water distribution networks, a field in which the modeling enhancement of nonlinear network flows has experienced eminent interest in order to develop physically sound models for real-world applications. We will drop the attributes *drinking* and *distribution* and for this chapter establish the convention that the term *water networks* subsumes anything that is named by *drinking water distribution networks, water supply systems,* or combinations of the two. Other types of water networks, such

as waste water networks [RH99] or water usage and treatment networks in chemical plants [HCLC99] are not considered here. This is due to the fact that the underlying network flow models, despite containing nonlinearities, differ from the ones that are widely used for pressurized water networks, and that we are mostly interested in.

As mentioned earlier, nonlinear network flow problems belong to the class of non-convex MINLPs and as such, in their general form, they involve two sources of non-convexity. The first arises from the nonlinear equations that model the flow of water into pipes (depending on the pressure difference at the nodes), and the second from discrete choices. Thus, these problems are generally NP-hard. The initial motivation of the work presented in this chapter was the following observation. For certain variants of water network optimization problems, satisfactory results with piecewise linearization approaches as in Section 1.2.5 were repeatedly reported [GMMS12, MGM12, Mor13], while this is not true for certain other variants. Id est, in water network optimization, MILP approximations of the encountered MINLPs sometimes lead to good and practically relevant computational results, but sometimes they do not. This raised a series of questions. Do we maybe just have to find the right tuning of the MILP approach for each variant individually in order to make it work? Or is there something intrinsically nonlinear in some problem variants that make it impossible to achieve satisfactory results with MILP approximations? The question that kept us awake at night was why an algorithmic approach, that is practical for a certain problem class, does not work on an at first sight very related class. The answers to such questions are, of course, never of black-or-white nature, and we will try to give some in Section 2.4.

A partial and not rigorously defined classification of water network optimization problems that separates the formerly mentioned variants (those for which MILP-approximations seem to work), from the latter (those for which it does not) is obtained by considering the somewhat different tasks of *optimal operation of water networks* on the one hand and *optimal design of water networks* on the other. This classification is suggested by surveying the water network optimization literature, and this will become eminent in Section 2.3, where we summarize computational results that were presented by several authors. We point out that besides operation and design there are other topics related to the optimization of water networks, for example the containment detection problem [LBvBW06] or topics related to water quality management [RB96]. Thus, the aforementioned classification is by no means exhaustive, but we choose to focus on these two problem variants. The fact that it turns out to be difficult to obtain satisfactory computational results for water network design problems with MILP approximations was, to the best of our knowledge, first pointed out in [BDL$^+$11]. In Section 2.4.3 we will report on computational experiments that confirm this impression, and we will analyze possible

reasons for this phenomenon by algorithmic considerations. We will conclude that the presence of certain convex substructures, that we treat in detail in Section 2.2, makes some general purpose approaches, including MILP techniques applied to a piecewise linearization of a water network design instance, impractical.

Water network operation and water network design represent two out of the different stages into which water network optimization can be subdivided [dCS12]. On both sides, one assumes to have an underlying network with a fixed topology, i.e., a fixed set of nodes and arcs representing sources, sinks, pipes, pumps, valves, and tanks. The latter of the two optimization tasks, i.e., the design problem, usually disregards pumps, valves and tanks. One then seeks to choose for each pipe in the network a diameter among a discrete set of commercially available diameters in a cost-minimal way, while maintaining the satisfiability of all costumer demands located at sinks. The diameter has an important impact not only on the pipe's capacity, but also on the pressure distribution in the network. The operation problem instead typically assumes fixed pipe diameters but allows for the modeling of pumps, valves, and tanks. The task is then to operate pumps and valves, which again affect flow and pressure distribution, over a certain time horizon in order to satisfy the customer demands, while minimizing the operational costs mainly arising from power consumption of pumps. In its full-scale form, the operation problem hence incorporates the aspect of time into the model and is thus a *dynamic* problem. Conversely, in order to model the design problem, no time parameter is neither needed nor generally used in the literature, which is why we consider it a *static* problem. The fact that active elements (pumps, valves, and tanks) are disregarded in design problems, and that there are only passive elements (pipes), is prevalent in the literature, although this does not necessarily have to be so. However, unifying design and operational phases in a single model bears some difficulties, and we will address this issue shortly in Section 2.5. In spite of the differences between design and operation, there are still some obvious similarities from a mathematical point of view, due to the way water dynamics in a pipe is described. Typically, the majority of the arcs in a network is constituted by pipes, and the equation associated with a pipe will be at the heart of Sections 2.3 and 2.4.

This chapter is mostly based on joint work with Claudia D'Ambrosio, Andrea Lodi and Cristiana Bragalli, and part of it was published as a survey paper of algorithmic nature in [DLWB15]. Thus, a lot of the following can harmlessly be read as a survey that, however, serves the ultimate goal of answering the questions about MILP-approximations raised above. In [DLWB15], we were interested in surveying Mathematical Programming approaches, i.e., methods that explicitly use a Mathematical Programming model. Those methods exploit (different variants of) different algorithmic paradigms to solve MINLPs, including, as mentioned above,

MILP-techniques. We also try to provide an overview of how different techniques succeed in different situations. In the last decade, Mathematical Programming approaches have experienced increasing popularity, whereas before that, optimization problems related to water networks were prevalently attacked by (meta-)heuristic methods, see, e.g., [dCaCR04, dCS12], without explicitly using a Mathematical Programming formulation. In the following, we do not dwell on the variety of those heuristic approaches that exist in the field. For a broader discussion of topics related to water networks that covers also aspects that are not of algorithmic nature, we refer the reader to [CAC14]. Finally, the existence of Partial Differential Equation (PDE) approaches in the field of water network optimization is worth mentioning, see, e.g. [LBvBW07], reflecting the fluid-dynamic nature of the topic. Again, this is outside the scope of the present chapter.

In Section 2.6, we present some results on how to derive nonlinear valid inequalities for one of the water network optimization problems we consider in the preceding sections. That section is inspired by techniques that were published in the context of gas networks and is based on joint work with Jesco Humpola and Andrea Lodi.

2.1 Modeling water networks

To begin with, we present the main modeling aspects found in the literature concerning the design and the operation problems. Regarding the various network elements, different variants at different levels of detail can be found. The most detailed modeling description of the relevant aspects is provided in [BGS08]. A network is naturally represented by a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where nodes stand for sources and sinks and arcs stand for pipes, pumps, and valves. Tanks are usually modeled as nodes, but this is not always true, see, e.g., [MGM12].

2.1.1 Flow & pressure

A main difference between the design and the operation problems is of course the contrast between the static and the dynamic setting. In the latter, in principle all variables and parameters can be made continuous-time dependent on $t \in [0, T]$, where $[0, T] \subset \mathbb{R}$ is the considered time horizon. However, to get a tractable optimization model, time is usually discretized and the quantities depend on the discrete time period $n \in [N]$ of length τ_n. A typical planning horizon is one day, divided in 24 hourly periods. In [BGS08] it is pointed out that the discretization has another practical motivation. Namely, demand forecasts and electricity price tariffs are usually given for discrete and not continuous time. In the following we will highlight the

time dependency of variables or parameters with a superscript n only when different periods are involved in an equation or constraint. Otherwise, the equation usually has to be imposed in every time period. In a static setting there is of course no time dependency to be highlighted. In any case, we introduce a flow variable q_a on each arc $a \in \mathcal{A}$. By allowing the flow to take negative values, a directed graph accounts for its both possible directions: a positive flow q_a on an arc $a = (i, j)$ means that it goes from i to j, while a negative value of q_a stands for a flow of amount $|q_a|$ from j to i. It is as well possible to allow only positive flow values and account for the directions with a binary variable.

From classical network flow problems one inherits the flow conservation constraints. For a node $i \in \mathcal{N}$ with demand d_i, which for the moment is assumed to be a constant, and the set of incoming and outgoing arcs, δ_i^- and δ_i^+, respectively, one has the linear constraint

$$\sum_{a \in \delta_i^-} q_a - \sum_{a \in \delta_i^+} q_a = d_i. \tag{2.1}$$

Sinks have a positive demand, which means that water actually leaves the network at those nodes. At sources, often called reservoirs in the water context, constraint (2.1) is usually not imposed. Otherwise, at such nodes, one can model d_i as a variable that can take only non-positive values, possibly bounded from below. Of course, there can be nodes with zero demands. Sometimes all nodes with positive or zero demand are called junctions. In real-world applications, there is usually uncertainty in the data, for example in the demand d_i. This aspect has been addressed in (meta-)heuristic approaches, see, e.g., [BKSW05]. However, in the Mathematical Programming literature, which is focus of the present chapter, it has been very rarely taken into account. Thus, in the following we consider approaches that assume deterministic data for the demands, which is, in any case, quite reasonable because generally one wants to establish a network design or operation that is feasible also in worst-case scenarios.

Next, one introduces the node potential variables h_i, $i \in \mathcal{N}$, representing the hydraulic head of the water at a node. This variable represents the pressure: In fluid dynamics, total head is the total energy per unit weight of fluid and is the sum of potential energy (elevation), pressure energy due to the pressure exerted on its container (pressure) and kinetic energy (velocity head). Consistent with this definition, total head and pressure are expressed dimensionally as a length. Due to the small value of velocity head in relation to the other two terms of the sum, the velocity head is generally neglected and the total head is assimilated to the hydraulic head, given by the sum of elevation and pressure.

A set of constraints that is regularly found in diverse optimization approaches arises from the fact that the two groups of variables introduced above are typically bounded. The absolute value of the flow is bounded from above due to the capacity of the arcs. For example, taking into account the maximum flow velocity that is allowed in a pipe a, $\overline{v}_a$, the flow bound can be written as

$$-\frac{\pi}{4}\overline{v}_a D_a^2 \leq q_a \leq \frac{\pi}{4}\overline{v}_a D_a^2, \tag{2.2}$$

where D_a is the diameter of pipe a, see [BDL+11]. The node potentials have to stay between certain bounds in order to guarantee minimum and maximum pressure levels at the nodes. Usually, the node potentials are fixed at source nodes, reflecting the fact that at sources water is not pressurized, but it exploits a fixed geographical height. We will see in Section 2.4 that the bounds on the potential values of non-source nodes often constitute the physical bottleneck in a water network optimization task.

2.1.2 Pipes

Typically, the majority of the arcs in a network represents pipes in which water is transported from one node to another, and this transportation is induced by different potentials at the nodes. The fundamental equation for a pipe $a = (i, j)$ is the head-loss equation, also denominated potential-flow coupling constraint in [HF15] (and we will use this term in the following), that is regularly of the form

$$h_i - h_j = \Phi_a(q_a), \tag{2.3}$$

where $\Phi_a(\cdot)$ is a strictly increasing uneven function, concave on the negative half-axis of its domain and convex on the positive half-axis. These are properties that also the signpower function from Sections 1.2.3 - 1.2.5 share. In fact, commonly used potential-flow coupling constraints use different forms of signpower functions, as we will see further down. Figure 2.1 (a) depicts $h_i - h_j$ as a function of q_a with these characteristics. The induced flow is not linear in the potential difference because this kind of function accounts for the modeling of friction in the pipes. A positive potential difference as a function of the flow is strictly increasing but convex: higher flow values mean higher influence of friction. The other way round, for the same reason, a positive flow as a function of the potential difference is strictly increasing and concave. Equation (2.3) is also referred to as the potential-loss equation, because it describes the pressure, or equivalently the energy, loss along a pipe. It is easy to see that once constraints (2.3) are embedded in a Mathematical Programming problem, the latter becomes non-convex, cf. Section 1.2.4. This is the case in most of

the models we consider in this chapter. Explicit forms of the potential-flow coupling equation are the so-called Darcy-Weisbach equation,

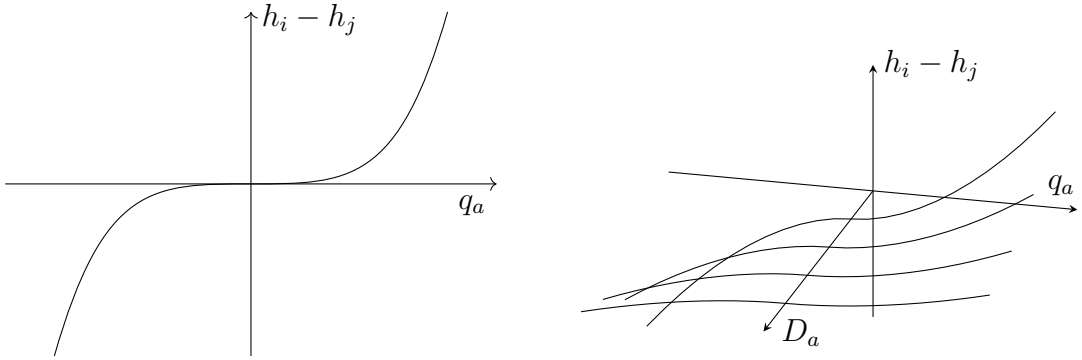
$$h_i - h_j = \frac{\operatorname{sign}(q_a) q_a^2 \cdot 8 \cdot L_a \cdot \lambda_a}{\pi^2 g \cdot D_a^5}, \tag{2.4}$$

or the Hazen-Williams equation,

$$h_i - h_j = \frac{\operatorname{sign}(q_a) |q_a|^{1.852} \cdot 10.7 \cdot L_a}{k_a^{1.852} D_a^{4.87}}, \tag{2.5}$$

both of which include some constants like the gravitational acceleration g, or ones depending on the pipe, such as the length L_a or the roughness coefficient k_a depending on the material of the pipe. In the denominator appears the diameter D_a of a pipe, which in optimal water network design becomes a discrete variable but stays a constant in the operation case. The solutions to the Hazen-Williams equation for some discrete diameters are depicted in Figure 2.1 (b). The friction factor $\lambda_a = \lambda_a(q_a)$ actually depends on the Reynolds number, which in turn depends on the flow, in a nonlinear, but continuous manner. There are several, partly implicit formulas treating the relation between friction factor and Reynolds number, for details see again [BGS08]. The most simplifying approximation neglects the dependency of the friction factor on the flow and thus treats the friction factor as a constant. This is done in most optimization models, see, e.g., [GHHV12, VA13], but sometimes also in simulation models. It can however be kept in the model, see, e.g., [MGM12]. As well in the Hazen-Williams equation, the dependency of the pressure drop on the flow is limited to the term $\operatorname{sign}(q_a)|q_a|^{1.852}$. Another downside of the above head-loss equations can arise when studying their differentiability for $q_a = 0$. The second derivative of the Darcy-Weisbach function is discontinuous at the origin, while the Hazen-Williams equation is not even second-order differentiable there. This can create problems when relying on derivative-based optimization methods as in [BGS08], which is why therein is used some kind of second order polynomial approximation of the Darcy-Weisbach equation with constant friction factor. The Darcy-Weisbach equation is often found with the substitution $\operatorname{sign}(q_a) q_a^2 = q_a|q_a|$. The coefficients of the Hazen-Williams equation are based on empirical data, while the Darcy-Weisbach equation is a theoretical formula. To the best of our knowledge, there is no computational study about the accuracy in the final results of using either formula in the context of the design and operation problems we consider in this text.

The pipe models seen so far assume a constant flow in a pipe, which is natural in a stationary setting. When dealing with a dynamic problem, this can be modeled even more accurately. A pipe model that allows for varying flow inside a pipe is based



(a) Potential-flow coupling equation (b) Discrete but variable diameters

Figure 2.1: Functions describing water dynamics in pipes

on the so-called water hammer equations, see, e.g [GZMA05]. This set of partial differential equations describes the variation of the state vector (q, h) in function of time and space (index a is dropped here),

$$\frac{\partial h}{\partial t} + \frac{c^2}{gA}\frac{\partial q}{\partial x} = 0 \tag{2.6}$$

$$\frac{\partial q}{\partial t} + gA\frac{\partial h}{\partial x} = -\lambda\frac{q|q|}{2DA}, \tag{2.7}$$

again with some coefficients among which are the diameter D and the cross-sectional area A of a pipe, the propagation speed of the pressure wave c and the friction factor λ, as before possibly depending on the Reynolds number. Note that when one assumes constant pressure and flow over time, i.e., turns to the static case, (2.6) implies constant flow in the pipe, and (2.7) becomes the Darcy-Weisbach equation with $\frac{\partial h}{\partial x}$ approximated by $\frac{h_j - h_i}{L_a}$. Here, just like the time dimension, that, as already mentioned, is usually discretized in optimization models, the same can be done in space, for example by implicit box schemes, see, e.g., [KL12]. However, a slight modification of the underlying graph model is necessary, see, e.g., [MGM12]. In fact, the flow in a pipe $a = (i, j)$ is not constant anymore, so a single variable q_a is not appropriate. Choosing the length of a pipe as spatial step size in the discretization scheme results in one flow value at the beginning of a pipe, as well as one at the end. This can be modeled by intermediate nodes. For node $i \in \mathcal{N}$ let δ_i^- denote the set of incoming pipes and δ_i^+ the set of outgoing ones. Then, consider the additional nodes i_a, $a \in (\delta_i^- \cup \delta_i^+)$, which can be imagined to be attached to the graph between the original node and the corresponding incident arc. We then have two flow variables per pipe per time step, $q_{i_a}^n$ and $q_{j_a}^n$. With this, the discretized versions of (2.6) and

(2.7) for a pipe $a = (i, j)$ become

$$\frac{h_i^{n+1} + h_j^{n+1}}{2\tau_n} - \frac{h_i^n + h_j^n}{2\tau_n} + \frac{c^2}{gA_a} \cdot \frac{q_{j_a}^{n+1} - q_{i_a}^{n+1}}{L_a} = 0 \tag{2.8}$$

$$\frac{q_{i_a}^{n+1} + q_{j_a}^{n+1}}{2\tau_n} - \frac{q_{i_a}^n + q_{j_a}^n}{2\tau_n} + gA_a \frac{h_j^{n+1} - h_i^{n+1}}{L_a} = -\frac{1}{2D_aA_a} \left(\frac{\lambda(|q_{i_a}^{n+1}|)q_{i_a}^{n+1}|q_{i_a}^{n+1}|}{2} + \frac{\lambda(|q_{j_a}^{n+1}|)q_{j_a}^{n+1}|q_{j_a}^{n+1}|}{2} \right). \tag{2.9}$$

While equation (2.8) is linear, (2.9) is not, just like its static counterpart (2.3).

2.1.3 Pumps

On the design side, the problem that is usually attacked disregards pumps and thus assumes that the underlying network is gravity-fed. In other words, a source node has a higher elevation than the nodes to which it induces a flow. If a network is fed with groundwater so that gravity does not suffice, or if the flow has to be transported over very long distances and hence loses too much pressure along an arc in order to satisfy the required minimum pressure level at the end node, certain node potentials have to be raised. This can be done by pumps. Moreover, on the operational side pumps can serve to fill tanks for intermediate storage purposes over time (see Section 2.1.5 below).

A pump is usually modeled as an arc $a = (i, j)$, so in particular there is a flow q_a passing through it. In comparison to pipes, pumps have a negligible length and thus the matter of constant or variable flow in a pump is not encountered. The flow through a pump is usually restricted in sign, that is, $q_a \geq 0$, allowing for flows from i to j only. Moreover, the flow through a pump is commonly considered semicontinuous, that is, either zero or in an interval $[\underline{q}_a, \overline{q}_a]$, with $\underline{q}_a > 0$. This already shows the need of a binary variable x_a for a pump, switching it on ($x_a = 1$) or switching it off ($x_a = 0$), i.e., acting like a closed valve ($q_a = 0$, see below).

In the context of the design of gas networks, the authors in [HF15] present an elegant overall MINLP formulation for nonlinear network flow problems exploiting a rather simple static model for potential raising elements, i.e., pumps in the context of water networks. The potential-flow coupling equation in [HF15] has the explicit form $h_i - h_j = \alpha_a q_a |q_a|^{\kappa_a}$, where α_a and κ_a summarize all the physical aspects of flow and pipe, containing as special cases also (2.4) and (2.5). The potential loss on the left-hand side is manipulated into a potential raise by multiplying the whole equation by minus one and adding a variable operating term,

$$h_j - h_i = -\alpha_a q_a |q_a|^{\kappa_a} + \beta_a y_a. \tag{2.10}$$

Here, β_a is a positive scaling factor that subsumes the physical characteristics of the pump and y_a is a non-negative variable. Thus, the product $\beta_a y_a$ represents the increment of the hydraulic head of node i due to pump operation plus the head loss through the pipe. This means that instead of losing some potential along the arc representing a pump, at its head node, the flow can have a potential value strictly higher than the one at its tail node. In other words, although the flow is going from i to j, we can have $h_j > h_i$. The same equation with $\beta_a y_a \leq 0$ can be used to model network elements that are able to reduce the pressure in some way. We will come back to this model in Section 2.4.2.

A different pump model tries to replicate the characteristic curve of a pump. More precisely, for a pump $a = (i, j)$, the potential raise can be written as

$$h_j - h_i = \Theta_a - \vartheta_a^1 q_a^{\vartheta_a^2}, \tag{2.11}$$

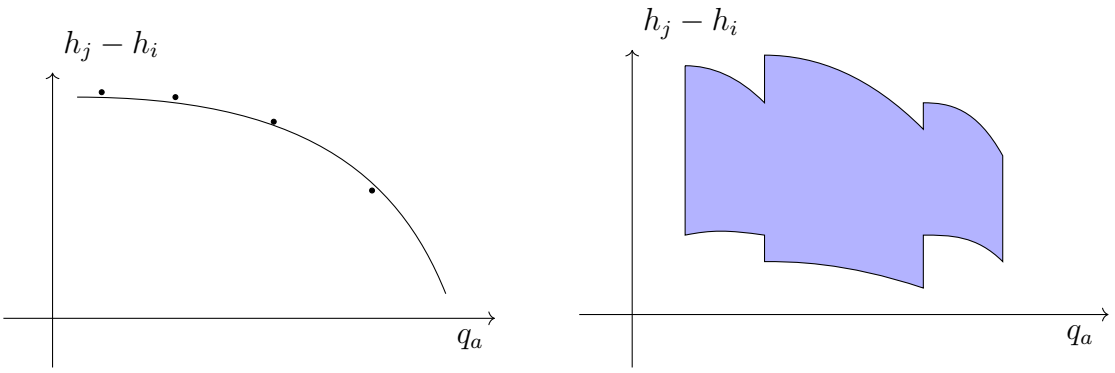
where the parameters Θ_a, ϑ_a^1 and ϑ_a^2 are chosen to approximate empirical pump data. A typical characteristic curve is shown in Figure 2.2 (a). There are also pumps that operate at variable speed $w_a \geq 0$. Basically, this quantity allows to shift the pump's characteristic curve in the plane and thus to manipulate the relation between potential raise and flow through the pump. Equation (2.11) becomes

$$h_j - h_i = w_a^2 \left(\Theta_a - \vartheta_a^1 \left(\frac{q_a}{w_a} \right)^{\vartheta_a^2} \right). \tag{2.12}$$

Also for a variable-speed pump the parameters of its characteristic curve are estimated through empirical data of the pump working at its nominal speed ($w_a = 1$). Equation (2.11) describes fixed-speed pumps that work at their nominal speed only. Instead, second order polynomials are used to fit the characteristic curve of a pump in [VA13].

The obvious differences between the modeling approaches (2.10) and (2.11)/(2.12) can be explained by the discrepancy between network design and network operation. When designing a network, the pump to be installed might be chosen from a certain production series. A typical pressure raise that can be realized by the pumps of such a series is shown schematically in Figure 2.2 (b). A very simple way to roughly model this operation range is actually equation (2.10), provided the variable y_a is bounded appropriately. Things change when considering the task to operate an already existing pump in an existing network. The specific pump is already chosen and has its own specific characteristic curve, which can be modeled quite accurately, for example by equations (2.11) or (2.12), respectively.

Another important quantity in the context of pumps is their power consump-



(a) Characteristic curve of a single pump (b) Operation range of a pump series

Figure 2.2: Pressure raise in networks with pumps

tion, which often appears in the objective function. The power consumption can be modeled as proportional to the product of the pressure raise of a pump and the flow through it, which adds a nonlinear equality constraint as in [GHHV12]. Otherwise, an equation similar to (2.12) or again a polynomial of degree two is fitted to empirical power consumption data at nominal speed.

Pumps are part of the active elements in a network and thus they allow for some discrete decision, that is, turning them on or off. Being turned on or off has implications on two aspects, one of which is the already mentioned semi-continuity of the flow variables. The same is valid in an analogous fashion also for the speed of pumps w_a. The second aspect instead concerns the potential-flow coupling equations (2.10) - (2.12), that are to be imposed only when the pump is working ($x_a = 1$). When $x_a = 0$, the pump is not working, and in this case it is usually regarded to act as a closed valve. The potential values at the two ends of the arc are then uncoupled and do not follow any equation, usually ensured by big-M constraints, see also the next section on valves. In order to avoid these combinatorial aspects, the authors in [BGS08] make some modeling modifications. A single pump $a = (i, j)$ in a graph often models an aggregation of real pumps in the network. The individual pumps in such a pumping station $a = (i, j)$ have a common pressure raise $h_j - h_i$, but the flow q_a through the pumping station is the sum of the individual flows. Approximately, the pressure raise of a pumping station becomes a degree of freedom independent of the aggregated flow value. To accurately measure the power consumption of such a pumping station, additional considerations about its efficiency are made. But all in all, the benefit is that there are no equations that have to be turned on or off, thus avoiding the use of big-M constraints.

2.1.4 Valves

Another set of active elements is given by valves. As for pumps, a valve is modeled by an arc $a = (i, j)$ with negligible length. There are models for different types of valves with no unified terminology in the literature. The authors in [MGM12] for example consider a total of four different types of valves. Valves can be used to actively block the flow completely or to reduce the pressure or the flow in its direction by a variable amount. They are usually modeled by a binary variable x_a. As an example consider the (rather simple) valve type that can be used to separate two pipes, that is, to avoid that fluid can pass between the two of them. If $\underline{q}_a$ and $\overline{q}_a$ denote the upper and lower bounds of the flow, an open ($x_a = 1$) or closed ($x_a = 0$) valve state can be modeled by the big-M constraints

$$\begin{aligned} \underline{q}_a x_a \leq & q_a \leq \overline{q}_a x_a \\ -M_a \cdot (1 - x_a) \leq & h_i - h_j \leq M_a \cdot (1 - x_a). \end{aligned}$$

A closed valve forces the flow to be zero and decouples the two potentials, while an open valve forces the potential values to be equal. Again in [BGS08], the authors model one specific valve type without auxiliary binary variables. Namely, the valve type considered allows to reduce the pressure by a variable amount, but only in the direction of the flow. This behavior is guaranteed by the inequality

$$(h_i - h_j)q_a \geq 0.$$

2.1.5 Tanks

Tanks can make the operation of the network more flexible. In a dynamic setting, where the demand at consumer nodes can vary in time, water can be stored in a tank during a period of low demand and be extracted from it to satisfy peak demands. In the static setting of [GHHV12] tanks are modeled as nodes in the graph that have a variable demand, which can also be negative so as to account for a positive initial tank filling. The water in a tank is usually not pressurized, which means the pressure head is zero, so that the potential value h_i represents the elevation head only. Accordingly, in [GHHV12] the potential value of the tanks is kept fixed as for reservoirs. In a dynamic setting, variable potential values can be used to describe the filling level of the tank at different points in time, which can then be linked to the tank's variable demand of a time period. A simplified version of the model for

tank $i \in \mathcal{N}$ in [BGS08] is given, for example, by

$$\sum_{a \in \delta_i^-} q_a^n - \sum_{a \in \delta_i^+} q_a^n = e_i^n \tag{2.13}$$

$$e_i^n = \frac{1}{\tau_n} \left(h_i^{n+1} - h_i^n \right) A_i, \tag{2.14}$$

with e_i^n denoting the variable volumetric tank inflow and A_i being the cross-sectional area of the tank. A quite detailed tank model that includes binary variables is found in [VA13]. Unless the discretized water hammer equations (2.8)-(2.9) are used, the tank filling equations constitute the most important point where subsequent time periods are coupled. This fact is exploited, for example, in [GNSK$^+$15], in order to apply an algorithm based on Lagrangian decomposition relaxation. In Corollary 2.2, we will provide a simple theoretic result for certain networks with tanks modeled by (2.13) - (2.14). Note that it is also possible, see, e.g., [MGM12], to model tanks as arcs in the graph, although this is far less common and essentially equivalent.

2.2 Convex substructures

As mentioned earlier, the optimization problems treated in this chapter are in the problem class of non-convex MINLPs. However, a recurring feature is the possibility of solving certain subproblems by convex optimization methods.

Consider a static network as in Section 2.1 with no active elements and with fixed pipe diameters, that is, a network with a flow variable q_a for each arc and a potential variable h_i for each node. Denote the set of source nodes by $\mathcal{N}^{src}$ and assume further that the potential values are fixed to some value h_i^{src} at source nodes $i \in \mathcal{N}^{src}$. Finally, assume that the flow conservation constraints (2.1) hold at each non-source node for a given set of fixed demands, and that each arc satisfies a potential-flow coupling equation like (2.3), where the only requirement about the function $\Phi_a(\cdot)$ is strictly increasing monotonicity. Formally, this gives rise to the feasibility problem

$$\sum_{a \in \delta_i^-} q_a - \sum_{a \in \delta_i^+} q_a = d_i \qquad \forall\, i \in \mathcal{N} \setminus \mathcal{N}^{src} \tag{2.15}$$

$$h_i - h_j = \Phi_a(q_a) \qquad \forall\, a \in \mathcal{A} \tag{2.16}$$

$$h_i = h_i^{src} \qquad \forall\, i \in \mathcal{N}^{src} \tag{2.17}$$

$$q_a \in \mathbb{R} \qquad \forall\, a \in \mathcal{A} \tag{2.18}$$

$$h_i \in \mathbb{R} \qquad \forall\, i \in \mathcal{N}. \tag{2.19}$$

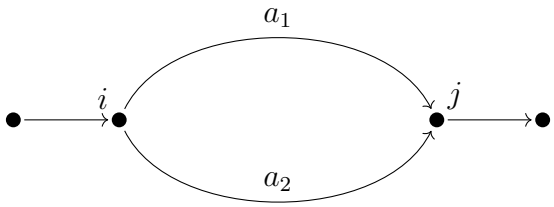


Figure 2.3: Simple example of a network containing a circle

The relation between this system of equations and one of its reformulations as a strictly convex optimization problem has originally been studied in [CCH$^+$78], and the main result therein has been slightly generalized in [Rag13][1]. It turns out that the solution space of the above set of $|\mathcal{A}| + |\mathcal{N}|$ equations in the same number of variables is a single point[2], and we state the latter result here for completeness:

Theorem 2.1. *If the functions $\Phi_a(\cdot)$ are strictly monotonically increasing, system* (2.15) - (2.19) *has a unique solution $(q, h) \in \mathbb{R}^{|\mathcal{A}|+|\mathcal{N}|}$.*

A slightly different version of the problem is studied in [HF15] in the context of gas networks; the flow conservation constraint is imposed at every node (also source nodes) but the node potentials are not fixed there. The number of variables and equations stays the same, and the existence of a solution is given for balanced flows only, that is, $\sum_{i\in\mathcal{N}} d_i = 0$. In that case, one flow conservation constraint at a node becomes redundant and the solution is a one-dimensional subspace. More precisely, the vector of flow variables q is unique and the vector of potential values h is located on a straight line in $\mathbb{R}^{|\mathcal{A}|+|\mathcal{N}|}$. It is easy to deduce that if one fixes the potential at any node, for example a source node, also the potential values become unique here.

The most crucial assumption leading to the striking argument in a potential proof of Theorem 2.1 is the monotonicity of the function $\Phi_a(\cdot)$. More precisely, in absence of the potential-flow coupling constraint (as in classical network flow models), a duplicity of flows can arise in the presence of circles only. By a circle in a graph we mean the existence of two different (not necessarily directed) paths between two nodes. As an illustrating example consider a simple (sub-)graph[3] as in Figure 2.3, see also [PT]. Flow conservation implies that the relation between the flow values q_{a_1} and q_{a_2} is a line with negative slope, say $q_{a_2} = q_0 - q_{a_1}$ for some $q_0 \in \mathbb{R}$. Taking into account also the potential-flow coupling constraints (2.3) we

[1] In that paper, $\Phi_a(\cdot)$ is assumed to be also continuously differentiable, but one can check that this assumption is not used in the proof of the theorem.

[2] Strictly speaking, in [Rag13], only the uniqueness of the solution is shown. However, its existence can be derived in a similar way as in [HF15].

[3] Actually, this is a multigraph, the two arcs a_1 and a_2 are to be thought of as two different paths from node i to node j.

get

$$\Phi_{a_1}(q_{a_1}) = \Phi_{a_2}(q_0 - q_{a_1}). \tag{2.20}$$

The function on the right-hand side of (2.20) is now strictly decreasing in q_{a_1} and as such has exactly one intersection point with the strictly increasing function on the left-hand side. Such considerations, leading to Theorem 2.1, and resulting methods are implemented in widely used software packages like EPANET [EPA], designed for numerically calculating flow and potential values in pressurized water networks.

As already mentioned, problem (2.15)-(2.19) represents a network with only pipes with fixed diameters and relaxed bounds on the variables. However, even when we allow for discrete variable diameters, it occurs as a leaf problem with relaxed bounds in a search tree, i.e., when all integer variables have been fixed, cf. Definition 1.3 and Remark 1.7. In such a situation, a subproblem of this type can in principle be solved to global optimality by a local NLP solver. The fact that the bounds are relaxed does not create a problem in the exploitation of this property, because their satisfaction can easily be checked after having obtained a unique solution (as done, e.g., in [Rag13]). Also when the potential values are not unique but located on a line as in the slightly different version of (2.15)-(2.19) studied in [HF15], it is easy to check whether there is one solution inside the bounds. As well in other situations the uniqueness of the solutions in subproblems can occur. Consider, for example, a network with working pumps that follow equation (2.11). Here, the pressure gain $h_j - h_i$ on arc $a = (i, j)$ is strictly decreasing in the flow q_a, meaning that the pressure loss is strictly increasing. At least this is true in the domain of q_a, so when relaxing the bounds maybe the function has to be extended conveniently, in such a way that it stays strictly increasing on $\mathbb{R}$. However, also a network with such pumps has a unique solution in its leafs, i.e., when the discrete decisions about which pumps are actually working and which are switched off have been taken.

The situation becomes different when pumps follow models (2.10) or (2.12). What actually happens here is that a degree of freedom is added. In the equation for a fixed-speed pump (2.11) the pressure raise is uniquely determined by the unique flow through the pump. This is not true for variable-speed pumps, where the pressure raise can be influenced by variation of the speed w_a. The same happens for model (2.10), where the pressure raise can be influenced by variation of the variable y_a independently of the flow q_a. Uniqueness or just convexity get lost here. Staying with the previous example, imagine the arc a_2 in Figure 2.3 to be a pump obeying

equation (2.10). Equation (2.20) then becomes

$$\Phi_{a_1}(q_{a_1}) = \Phi_{a_2}(q_0 - q_{a_1}) + \beta_{a_2} y_{a_2} \tag{2.21}$$

for some $\beta_{a_2} \in \mathbb{R}$, which is a non-convex subset of $\mathbb{R}^3$ in general. A remedy, just like in [HF15], is to discretize the degree of freedom, i.e., to allow the variables y_a or w_a to take values in a discrete set only. Then, when they are fixed to one of these values in a leaf of a search tree, the functions describing the pressure loss are strictly increasing in the flow q_a and uniqueness or convexity can be made use of.

We believe that there is a lot of potential in the property of uniqueness, or convexity, which has partially been exploited in water network optimization as we will see later. It might also be a key feature when developing techniques for discretized dynamic problems in water network optimization. For example, one can prove uniqueness of the solution when tanks are coupled by an equation of type (2.14) in a passive network with relaxed bounds considered over several discrete time periods. We have not seen a formalization of such a result yet, and to conclude this section, we give one as a corollary of Theorem 2.1. We denote by $\mathcal{N}^T$ the set of nodes that represent tanks[4] and the initial tank filling of $i \in \mathcal{N}^T$ by h_i^{init}. We then study the extension of (2.15) - (2.19) to several discrete time periods, coupled by (2.13) - (2.14):

Corollary 2.2. *For $N \geq 1$ and under the assumptions of Theorem 2.1, the system*

$$\sum_{a \in \delta_i^-} q_a^n - \sum_{a \in \delta_i^+} q_a^n = d_i^n \qquad \forall\, i \in \mathcal{N} \setminus (\mathcal{N}^{src} \cup \mathcal{N}^T), n \in [N] \tag{2.22}$$

$$h_i^n - h_j^n = \Phi_a(q_a^n) \qquad \forall\, a \in \mathcal{A}, n \in [N] \tag{2.23}$$

$$h_i^n = h_i^{src} \qquad \forall\, i \in \mathcal{N}^{src}, n \in [N] \tag{2.24}$$

$$\sum_{a \in \delta_i^-} q_a^n - \sum_{a \in \delta_i^+} q_a^n = e_i^n \qquad \forall\, i \in \mathcal{N}^T, n \in [N] \tag{2.25}$$

$$e_i^n = \frac{1}{\tau_n}(h_i^{n+1} - h_i^n) \qquad \forall\, i \in \mathcal{N}^T, n \in [N] \tag{2.26}$$

$$h_i^1 = h_i^{init} \qquad \forall\, i \in \mathcal{N}^T \tag{2.27}$$

$$q_a^n \in \mathbb{R} \qquad \forall\, a \in \mathcal{A}, n \in [N] \tag{2.28}$$

$$h_i^n \in \mathbb{R} \qquad \forall\, i \in \mathcal{N} \setminus \mathcal{N}^T, n \in [N] \tag{2.29}$$

$$h_i^n \in \mathbb{R} \qquad \forall\, i \in \mathcal{N}^T, n \in [N+1] \tag{2.30}$$

$$e_i^n \in \mathbb{R} \qquad \forall\, i \in \mathcal{N}^T, n \in [N] \tag{2.31}$$

has a unique solution $(q, h, e) \in \mathbb{R}^{(|\mathcal{A}|+|\mathcal{N}|)\cdot N + |\mathcal{N}^T|\cdot(N+1)}$.

[4] We assume $\mathcal{N}^{src} \cap \mathcal{N}^T = \emptyset$.

Proof. We use induction over N. If $N = 1$, by Theorem 2.1 we know that the subsystem composed by (2.22) - (2.24) and (2.27) has a unique solution. Then (2.25) uniquely determines the values $e_i^1 \; \forall \, i \in \mathcal{N}^T$, that together with (2.26) uniquely determine $h_i^2 \; \forall \, i \in \mathcal{N}^T$. Now assume $N \geq 2$ and that the Corollary is valid for $N-1$, i.e., the system of equations composed of (2.22) - (2.31) for $n \in [N-1]$ admits a unique solution. In particular, the values h_i^N are uniquely determined $\forall \; i \in \mathcal{N}^T$. This fact together with (2.22) - (2.24) for $n = N$ uniquely determines the values q_a^N $\forall \; a \in \mathcal{A}$ and $h_i^N \; \forall \; i \in \mathcal{N} \setminus (\mathcal{N}^{src} \cup \mathcal{N}^T)$ by Theorem 2.1. Just like before, uniqueness of e_i^N and $h_i^{N+1} \; \forall \; i \in \mathcal{N}^T$ can be deduced. □

2.3 Solution approaches

In this section we show in more detail what kind of approaches have been presented in the literature for solving water network design problems on the one hand, and water network operation problems on the other. All of the following are based on the general-purpose MINLP techniques presented in Section 1.2.2 - 1.2.5.

2.3.1 Design in the literature

On the design side, one specific formulation together with a set of literature and real-world instances has been studied by several authors. As mentioned earlier, active elements like pumps, valves, and tanks are disregarded. In return, we can choose the diameter for each pipe from a discrete set. The formulation basically consists of the flow conservation constraint (2.1) for each non-source node and the Hazen-Willliams equation (2.5) for each pipe. The diameter D_a on a single pipe a is a variable constrained to belong to a discrete set $\{D_{a,1}, \ldots, D_{a,r_a}\}$. The solutions to the Hazen-Williams equation for some discrete diameters have been shown in Figure 2.1 (b). To each diameter $D_{a,\ell}$ is associated a positive unit length cost $C_{a,\ell}$ in such a way that costs increase with the diameter. Finally, there are lower and upper bounds on the flows, cf. (2.2), as well as lower and upper bounds on the node potentials, and the potentials at source nodes are fixed. In an MINLP formulation, the membership of the diameter to a discrete set can be represented by introducing additional binary variables $x_{a,\ell}$ and using SOS-1 type equations. We give here an explicit MINLP formulation, because we will need it again in Section 2.6:

$$\min \quad \sum_{a \in \mathcal{A}} L_a \cdot \sum_{\ell=1}^{r_a} C_{a,\ell} x_{a,\ell} \tag{2.32}$$

$$\text{s.t.} \quad \sum_{a \in \delta_i^-} q_a - \sum_{a \in \delta_i^+} q_a = d_i \qquad \forall \; i \in \mathcal{N} \setminus \mathcal{N}^{src} \tag{2.33}$$

$$q_a|q_a|^{0.852}\frac{10.7\cdot l_a}{k_a\cdot D_a^{4.82}} = h_i - h_j \qquad \forall\, a\in\mathcal{A} \qquad (2.34)$$

$$\sum_{\ell=1}^{r_a} x_{a,\ell} = 1 \qquad \forall\, a\in\mathcal{A} \qquad (2.35)$$

$$\sum_{\ell=1}^{r_a} x_{a,\ell}D_{a,\ell} = D_a \qquad \forall\, a\in\mathcal{A} \qquad (2.36)$$

$$-\frac{\pi}{4}\overline{v}_a\cdot\sum_{\ell=1}^{r_a} x_{a,\ell}D_{a,\ell}^2 \le q_a \le \frac{\pi}{4}\overline{v}_a\cdot\sum_{\ell=1}^{r_a} x_{a,\ell}D_{a,\ell}^2 \qquad \forall\, a\in\mathcal{A} \qquad (2.37)$$

$$\underline{h_i} \le h_i \le \overline{h}_i \qquad \forall\, i\in\mathcal{N}\setminus\mathcal{N}^{src} \qquad (2.38)$$

$$h_i = h_i^{src} \qquad \forall\, i\in\mathcal{N}^{src} \qquad (2.39)$$

$$x_{a,\ell}\in\{0,1\} \qquad \forall\, \ell\in[r_a], a\in\mathcal{A} \qquad (2.40)$$

$$q_a\in\mathbb{R} \qquad \forall\, a\in\mathcal{A} \qquad (2.41)$$

$$h_i\in\mathbb{R} \qquad \forall\, i\in\mathcal{N}. \qquad (2.42)$$

As seen in Section 2.2, if the diameters were fixed, this would result in a convex (feasibility) problem. Convexity does not hold for a variable diameter, especially not when it is discrete, cf. Figure 2.1 (b), and thus (2.32) - (2.42) is a non-convex MINLP. As mentioned, there is a set of 9 instances for the design problem, out of which 4 are smaller literature instances and 5 are larger real-world ones representing water networks of three Italian cities, one of which is counted three times due to three different diameter sets. These instances can be obtained at [LabOR], their characteristics are subsumed in Table 2.1. The set of available diameters is actually the same for each pipe in each instance, i.e., r_a is constant over $a\in\mathcal{A}$.

The nonlinear branch-and-bound algorithm implemented in BONMIN has been applied to these instances in [BDL+11]. Remember that this acts as a heuristic solver for non-convex MINLPs, cf. Section 1.2.2. Several amendments to the model, on the one hand, and to the algorithm itself, on the other hand, were made. For

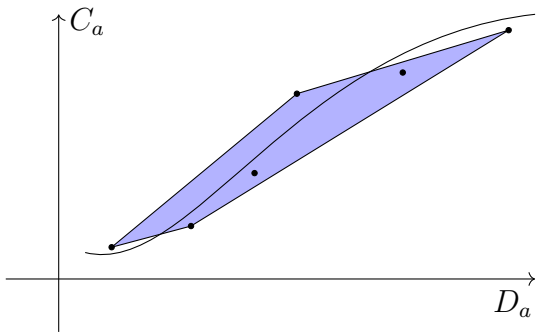
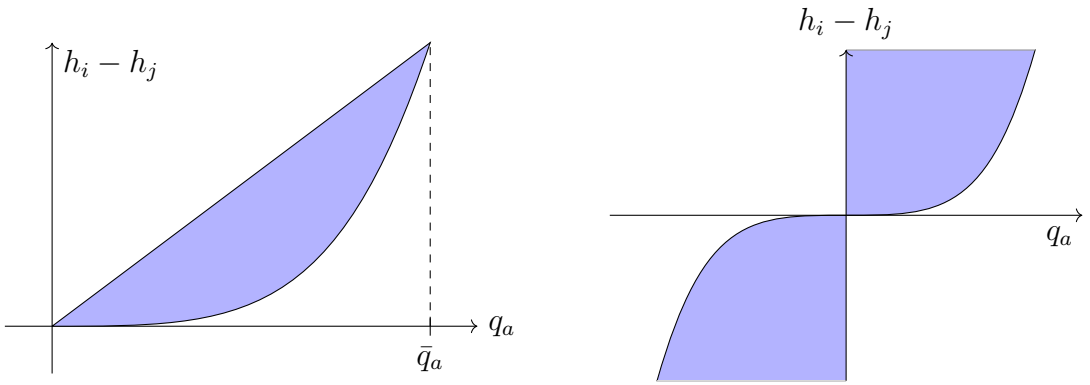


Figure 2.4: Fitted polynomial and convex hull of diameter costs



(a) Convex relaxation of a positive head loss (b) Head loss of a pipe-valve-sequence

Figure 2.5: Relaxations of the potential-flow coupling constraint

example, a fitted polynomial $C_a(D_a)$ is used as objective function instead of the sum $\sum_{\ell=1}^{r_a} x_{a,\ell} C_{a,\ell}$ in order to get a smooth function. Moreover, this function usually produces tighter bounds because the optimal cost value for pipe a of some NLP-relaxation is a point on the graph of $C_a(D_a)$ instead of just a point in the convex hull of the points $(D_{a,1}, C_{a,1}), \ldots, (D_{a,r_a}, C_{a,r_a})$, see Figure 2.4. However, the polynomial is not chosen to be an exact fit in the arguments $\{D_{a,1}, \ldots, D_{a,r_a}\}$, which is why the fitted objective is only used to guide the search in the tree, while the real objective is used to calculate the cost of integer feasible solutions. This is a nice example of how problem specific solution paradigms can give rise to enhancements of general-purpose solvers. Indeed, the option of working with two objective functions was added to the implementation of BONMIN afterwards. Also the implementation of proper SOS-1 branching in BONMIN, which can be used instead of the binary requirement of the variables $x_{a,\ell}$, was stimulated by the water network design application.

A modified LP/NLP-based branch-and-bound framework that exploits the convexity structure presented in Section 2.2 is proposed in [Rag13]. This algorithm is then exact for the non-convex design problem. There are three crucial points in the approach. First, each arc in the network is cloned as many times as there are diameters available on it. One therefore gets much more potential-flow coupling constraints, but each of them is given by a univariate function (because the diameter on a cloned arc is fixed, see Figure 2.1). Second, for each arc, a record of the flow direction is then kept by explicitly introducing an additional binary variable indicating the direction. The gain of both reformulations above is that the resulting non-convex constraints can easily be relaxed to two convex constraints, because only an either concave or convex part of the function $\Phi_a(\cdot)$ has to be taken care of. For example, for a positive flow q_a with upper bound $\overline{q}_a$, these two constraints are

precisely

$$h_i - h_j \geq \Phi_a(q_a) \quad \text{and} \quad h_i - h_j \leq \frac{\Phi_a(\overline{q}_a)}{\overline{q}_a} \cdot q_a.$$

This situation is depicted in Figure 2.5 (a). In this way, in [Rag13] a convex MINLP is obtained that is a relaxation of the original MINLP (2.32) - (2.42). A branch-and-cut scheme as in an LP/NLP-based branch-and-bound algorithm is applied to such a convex MINLP relaxation. Instead of solving the associated NLP of a leaf problem, and this is the third crucial point, the unique solution to the feasibility problem stated in Section 2.2 is found. If this unique solution additionally satisfies flow and potential bounds, the node is feasible for the original non-convex MINLP. In this way a stronger condition, directly related to the original MINLP, is tested instead of just solving the sub-NLP of its convex relaxation. If the bounds are violated, the node is eliminated from the search tree by some cut. Note that a globally optimal solution to the non-convex MINLP is found since the objective function depends on the integer variables only.

We now want to present the computational results that were reported in conclusion with the two algorithms presented above applied to the described water network design instances. We also include the results of applying plain SCIP, i.e., spatial branch & bound, to these instances, without any exploitation of problem structure, reported in [Vig12, Sec. 8.2], and updated in [Vig13b]. The results of these three different approaches are shown in Table 2.2. All reported results were obtained on different computers and with respect to some time limit and in one case with respect to an additional memory limit in terms of branch-and-bound nodes. The machine characteristics are given in the respective column. The best lower and upper bounds found are reported. For the BONMIN-based approach in the second column we obviously have no lower bounds due to the heuristic nature of the approach. The algorithm in [BDL$^+$11] reached the time limit in all except the first instance. The results of the exact approach of [Rag13], for brevity called LP/CVXNLP, are given in columns 3 and 4. Here, an optimal solution is found in all of the four smaller literature instances, as a side effect certifying the optimality of the heuristic solutions in column 2. For the larger instances, the algorithm LP/CVXNLP is not able to terminate within the specified limits, but in any case finds feasible solutions, that, except for the instance pescara, are not better than the solution found by the BONMIN-based algorithm. Taking into account the spatial branch-and-bound algorithm of SCIP in columns 5 and 6, there appear some inconsistencies. The reported optimal solution of the small instance blacksburg is smaller than the optimal solution found by LP/CVXNLP. In addition, the optimal solutions of foss_poly_0 and foss_iron are below the lower bounds given by LP/CVXNLP. We

can only explain these inconsistencies with numerical issues in either of the two algorithms. One source of these issues might be the reformulation techniques used in SCIP, see Section 1.2.4, which sometimes produce slight infeasibilities [Vig13a]. The approach LP/CVXNLP is the only approach that makes use of the convexity property described in Section 2.2. A clear advantage over the other two approaches presented in the above summary of computational results cannot be seen. However, given the heterogeneous experimental setups, the reported results are not suited for making a truly fair comparison.

2.3.2 Operation in the literature

The full-scale problem of optimal water network operation appears to be a rather hard task. By full-scale we mean being based on a time (and space) discretized formulation. Until recently, to the best of our knowledge, there had not been any successful solution approaches for this complete form in the literature. Three simplifications had been proposed, which we will present in the following. After that, we will come to a more recent approach that combines optimization and simulation techniques and does not simplify the full-scale problem. Unfortunately, there does not seem to be a unified test set, which makes a concise comparison, not only of the performance of the algorithms, but also of their practicability, difficult.

The first simplification is obtained in [GHHV12] by dropping the time dimension and regarding a static operation problem. Such a problem may arise as subproblem in the full-scale task, possibly useful in heuristic approaches to the latter. At the basis is a network model with fixed-speed pumps following equation (2.11), valves that can be closed or reduce the pressure in the direction of the flow, and tanks with a fixed initial filling level. The pipes have a fixed diameter and the potential-flow coupling equation is given by (2.4). The objective function is the sum of the cost of water purchased at source nodes and the cost of the power consumption of pumps. An innovation of the study is the introduction of the concept of real and imaginary flows. It is observed that due to valves, it is possible that actually no water is present at certain nodes. Enforcing the potential-flow coupling constraint on an arc that is incident to such a node, which means inducing a flow on that arc, is wrong. If no water is present at a node, no flow emerging from it can be induced. The concept of real and imaginary flows is modeled with the help of additional binary variables. Some interesting preprocessing steps for that model are also presented, one of which can be applied to sequences of pipes and valves. Due to the presence of valves that can reduce the pressure in the direction of the pipe, the pressure loss of an entire pipe-valve-sequence is not described by the Darcy-Weisbach equation itself but by some relaxation of it, see Figure 2.5 (b). With some extra effort,

this union of two convex sets can be modeled by convex constraints. Again, as in [Rag13], the potential-flow coupling function's property of being "half-concave" and "half-convex" is exploited. The resulting MINLP is solved by using SCIP, applying the approach to two real-world networks. It is tested on different scenarios in the two networks, given by different initial tank fillings and the forecast demands of different time windows of a day. The larger of the two networks consists of 88 nodes and 128 arcs, resulting in a program with about a hundred binary variables (without presolve). The most difficult scenario requires about half an hour of computing time. When applying the presolving steps, the computation times reduce drastically. All of the tested scenarios are solved within less than two minutes of computation time, on average much faster. A natural question to ask is how the solution method in [GHHV12], that is, the spatial branch-and-bound algorithm of SCIP, works on the full-scale operation problem. The authors reported that already going up to the full-scale problem with only two or three time periods is troublesome with the - at that time - current version of SCIP [Gle13].

Another simplification of the full-scale operational problem is obtained by piecewise linearly approximating the nonlinear functions in the time discretized model, as shown in [MGM12] and [GKL$^+$11], or in more detail in [Mor13]. The model therein mainly consists of flow conservation constraints (2.1) adapted to time-dependent variables, potential-flow coupling constraints (2.8) and (2.9) and pump equations (2.12). As mentioned, four different types of valves are modeled and two types of constraints on each tank, that are modeled as arcs, are imposed. The authors also impose a terminal filling level of each tank and are able to model the so-called *breathing*. This technical requirement imposes that a tank has to be filled and emptied completely for a given number of times over the considered time horizon. Finally, there are additional linear constraints that account for minimum runtimes and downtimes of pumps. Preprocessing techniques, see Section 1.2.5 and [GMMS12], are used to approximate all nonlinear functions within a controlled error bound of 10^{-2}. Some results on the comparison between the different models for piecewise linear functions especially in the water network context are reported in [GMMS12]. The winning method therein is the incremental method, which is also the method of choice in [MGM12] and [GKL$^+$11]. The tests are conducted on three networks of varying size, optimized over a time horizon of either four hours divided into 12 time steps or one day divided into 24 hourly steps. It is interesting to note that there is some flexibility of what is represented by the objective function, according to the underlying application. So, in some tests, the objective function is chosen to be the minimization of the number of tanks with a filling level below a certain limit, so as to maximize the supply guarantee of the network. In other cases, the overall power consumption is minimized. As we have seen earlier, also the purchase cost of water

at source nodes can be taken into account in the objective function. The largest considered network consists of 25 arcs, resulting in an approximating MILP with almost 11,000 binary variables, for which an MILP solver finds an optimal solution within 694 CPU seconds. Further computational results can be found in [Mor13, Sec. 8.1].

A MILP approach through piecewise linear approximations is also presented in [VA13]. Besides the Darcy-Weisbach equation there are second-order polynomials that represent the characteristic curve of pumps with fixed-speed only, while the empirical power consumption of a pump is fitted by (linear) polynomials of degree one. The objective function is again a combination of the cost of purchased water and the power consumption of pumps. The nonlinear univariate functions are piecewise linearly approximated by a modification of the convex combination method. This modification was proposed in [VN09] and observes that in SOS-2 situations as for piecewise linear approximations, the required number of auxiliary binary variables is actually only logarithmic in the number of approximating line segments. The formulation is further strengthened by adding valid inequalities. In addition, the authors actually use a piecewise linear relaxation as described in [GMMS12] rather than an approximation, see again Figure 1.4 (b). The test network consists of 30 arcs, and the time horizon of one day is divided into 5 time steps. The reported results consider linear relaxations based on a number of approximation intervals ranging from 2 to 8. An MILP solver terminates at an optimal solution in the range of less than 30 seconds.

Computational results obtained with the very detailed model developed in [BGS08] are reported in the follow-up paper [BGS09]. As stated at the end of Sections 2.1.3 and 2.1.4, there are ways to approximate the MINLP model without introducing binary variables. For their large network with 1,481 nodes and 1,935 arcs arising from the drinking water distribution network in the city of Berlin, the authors apply purely nonlinear programming techniques without any kind of branching, thus no search trees are explored. Due to the non-convexities that are still present, this leads again to only locally optimal solutions of the problem with discrete aspects neglected.

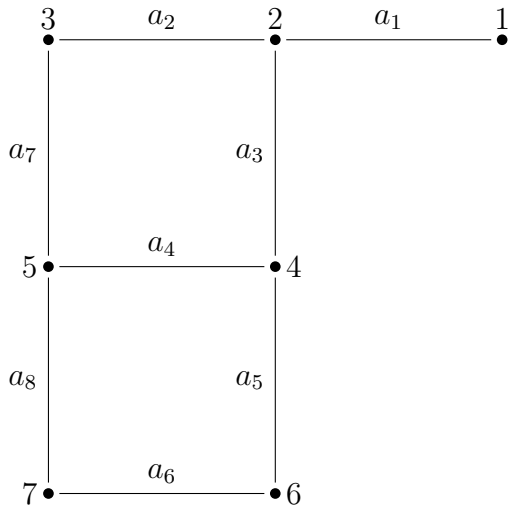
In [NSGAE15], an approach for the full-scale operational problem with fixed speed pumps only, combining Mathematical Programming techniques with simulation tools, is presented. In a master problem, the hydraulic constraints are disregarded completely and only requirements on the compatibility of the discrete decisions related to pumps and tanks are imposed. This leads to an MILP that can be solved by an MILP solver. However, whenever an integer feasible solution is found, the hydraulic simulator EPANET is called and verifies whether the hydraulic constraints are satisfied by this discrete configuration, or not. In principle, this ap-

proach is an exact approach for the considered problem. We note that implicitly, it makes use of a property that is closely related to the result of Corollary 2.2. The approach is thus somewhat related to LP/CVXNLP from the previous section in the context of water network design. One difference, that is of more technical nature, is that a simulation tool instead of an NLP solver is used to verify the discrete configuration, or equivalently, to solve the leaf problems as NLPs. A disadvantage could be seen in the fact that the hydraulic constraints are disregarded completely and thus cannot contribute to the computation of a dual bound during the tree-search. In the MINLP framework of LP/CVXNLP instead, relaxed versions of the hydraulic constraints are present throughout the whole algorithm. In [NSGAE15], computational results on two networks are reported, the larger one with 388 nodes and 432 pipes. There is not a single case in which the algorithm terminates before hitting the imposed time limit of one hour, and thus no provably optimal solutions are provided. Nevertheless, the proposed combination of MILP and simulation software is appealing, and we believe that the incorporation of MINLP techniques in order to improve dual bounds during the search process is a promising way to go.

2.4 Piecewise linearizations of the potential-flow coupling equation

We have surveyed a class of challenging optimization problems and the attempts to solve them. The problems exhibit some differences among them, but all have in common the underlying nonlinear network flow model. The potential-flow coupling constraint is present in almost all formulations (and if not, its dynamic extension (2.8)-(2.9) is), and this constitutes one of the main difficulties.

In this section, we discuss a very simple computational example. Consider the water network design instance shamir from Section 2.3.1. The network is depicted in Figure 2.6. Node 1 is the only source node and water is transported from there to all the other (consumer) nodes. Originally being a water network design instance, the shamir network does not have any active elements. In that figure are also listed the diameters on each pipe, that we consider as fixed for the moment. This diameter set actually represents the optimal diameter set of the water network design problem, which on this small instance can be determined by any of the three methods presented in Section 2.3.1. Since this diameter set is optimal, it is in particular feasible, i.e., leads to a feasible (and unique) solution (q, h) that satisfies the flow conservation constraints, the potential-flow coupling constraints (that is, system (2.15) - (2.16)) and the bounds on flow and pressure, and can be determined by any (also local) NLP solver. The example can be seen as a subproblem of a water network



	D_a $[mm]$
a_1	457.2
a_2	254.0
a_3	406.4
a_4	101.6
a_5	406.4
a_6	254.0
a_7	254.0
a_8	25.4

i	$\underline{h}_i$ $[m]$
1	201
2	180
3	190
4	185
5	180
6	195
7	190

Figure 2.6: The shamir network

optimization instance and helps to illustrate the physical bottleneck in many such subproblems. This bottleneck is constituted by the lower pressure bounds. For example, decreasing the diameter on arc a_6 to the next available discrete one from the water network design instance would result in a network flow that does not violate the capacity on that arc, but violates the lower bound on the pressure of node 7. This again can be certified by any NLP solver. The same effect can be obtained by reducing the diameter on several other arcs.

2.4.1 MILP- vs. NLP-feasibility

In the following, we will distinguish between NLP- and MILP-feasibility of certain subproblems of an original MINLP in the context of water networks. To be more precise, assume that the underlying non-convex MINLP is of the form (1.18) - (1.21) and that its MILP-approximation is given by

$$\begin{aligned} \min \quad & c^T x && (2.43)\\ \text{s.t.} \quad & Ax + Wz \leq b, && (2.44)\\ & x \in \mathbb{R}^{n-p} \times \mathbb{Z}^p, && (2.45)\\ & z \in \{0,1\}^{\tilde{p}}, && (2.46) \end{aligned}$$

Now when fixing the original discrete variables, that is, the last p components of the vector x to some value $\hat{x}_{(p)}$, (1.18) - (1.21) becomes an NLP, corresponding to one of its leaf problems, while (2.43) - (2.46) remains an MILP. By NLP- and MILP-feasibility, we mean the feasibility of these two subproblems, respectively, with respect to a certain $\tilde{x}_{(p)}$. A question that arises is the one about the relation

between NLP- and MILP-feasible solutions, i.e., are there solutions that have, for example, both of these attributes? When as a solution we mean the collection of discrete decision values, i.e., $\tilde{x}_{(p)}$, plus flow and potential values, the answer to that question is rather negative. Indeed, when approximating all nonlinear functions by piecewise linear ones, the only points that lie on the graph of both functions are the breakpoints. That means that only a solution that takes values exclusively in the set of breakpoints would be both NLP- and MILP-feasible. Inside the continuous domain of the flow and potential variables, however, this is a fairly small set of points.

Instead, if as solution we intend only the discrete decision values $\tilde{x}_{(p)}$ (because from a practical point of view the flow and pressure variables are not really decision variables), the situation simplifies. Therefore, we turn to the shamir network from above. Imagine to have an MILP-approximation of the potential-flow coupling equations for the shamir network with diameters fixed to the optimal diameter set, as above. Since the potential-flow coupling function is convex in $|q_a|$, the energy loss will be overestimated by this approximation, cf. Figure 1.4 (a). The worse the approximation is, the higher will be this overestimation. In fact, when fixing the diameters to the optimal diameter set and using five linearization points per pipe in an MILP model for our example, any MILP solver certifies the infeasibility of the optimal (and thus NLP-feasible) diameter set. The bottleneck are again the lower pressure bounds, this time at nodes $3, 5, 6$ and 7. So, MILP approximations lead to conservative solutions in general, because they overestimate the energy loss in a network. The worse the approximation is, the more conservative gets the solution. If one keeps the approximation coarse as to keep the number of auxilary binary variables low, fewer diameter configurations become feasible, which is why it becomes harder to find feasible solutions in the search tree. Refining the approximation augments the number of binaries and thus slows down the MILP solver. The implicit use of conservative solutions thus leads to this side effect of MILP-approximations of water network optimization problems in practice.

Another characteristic of the underlying nonlinear network flow model, which here leads to purely algorithmic considerations and does not aim at the discrepancy between NLP- and MILP-feasibility, is given by the property described in Section 2.2. We have seen that in some situations a nonlinear network flow in a subproblem (like the shamir network above) is unique, provided it is feasible at all. It seems sensible to compute this solution by a local NLP solver in polynomial time, instead of exploiting other methods. Take, for example, the classic formulation for the water network design problem (2.32) - (2.42) described in Section 2.3.1. It turns out that an MILP approach via piecewise linear approximations performs terribly bad, see Section 2.4.3. Indeed, even when linearizing the Hazen-Williams equation with only a few

linearization points, no MILP-feasible solution is found by an MILP solver within a reasonable time limit on medium size instances. This is because the leaf problems in the search tree are either MILP-infeasible or have exactly one solution (note that as well the linearized version of the potential-flow coupling constraint is strictly increasing). Certifying MILP-infeasibility or searching this unique solution with an MILP formulation cannot be a good idea. The same conceptual problem occurs if the subproblem with an at most unique solution is solved by spatial branching. An NLP-feasible solution will be found in at most one of the many branches. An additional difficulty in design problems often arises from the fact that the objective function is constant in such a subproblem, since it only depends on the diameter choices. This results in exploring a sub-tree without the guidance of the objective function. We conclude that in this situation the most promising thing to do is to exploit the uniqueness of the leaf problems by some combination of integer branching and nonlinear programming techniques (see Section 2.2 and references therein).

2.4.2 The role of pumps

At this point, it is possible to illustrate one aspect of the role of pumps. Abstracting from the original physical setting of the shamir instance, imagine that there is a pump associated with arc a_3, modeled by the equation

$$h_4 - h_2 = -\frac{\text{sign}(q_{a_3})|q_{a_3}|^{1.852} \cdot 10.7 \cdot L_{a_3}}{k_{a_3}^{1.852} D_{a_3}^{4.87}} + y, \tag{2.47}$$

where $y \in \mathbb{R}_+$, cf. equation (2.10). Obviously, the network with fixed optimal diameters still has an NLP-feasible solution. But also decreasing the diameter on arc a_6 leads to an NLP-feasible solution now, namely with $y > 0$. Adding another such virtual pump on arc a_2 turns feasible also the aforementioned MILP approximation of the optimal diameter set with five linearization points, that before was infeasible. In this case, pumps could also be placed on arcs a_2 and a_5 or just on arc a_1 in order to achieve feasibility. In other words, pumps are somehow able to compensate for the approximation error made by MILP approximations. This could lead to computational advantages by producing more quickly diameter configurations that, tested a posteriori by solving convex leaf problems, show no need of pump usage. Of course, this depends on the cost of the pumps that is, however, somehow difficult to evaluate in a static framework like the design one. Namely, taking decisions on pump usage in the design context might imply operation problems that are feasible only if the pump is always used. Thus, the cost saved for installing pipes might be spent for the continuous use of pumps. We will analyze this latter issue in more detail in Section 2.5.

An interesting paradox is encountered in the context of pumps that are modeled by (2.47). Imagine to have some water network optimization problem for which all subproblems with fixed integer variables possess the convexity or uniqueness structure described in Section 2.2. Again, take for example the classic water network design problem and imagine to have pumps modeled by equation (2.47) in the network. In order to be able to use the aforementioned algorithmic combination of integer branching and exploitation of uniqueness, one would have to discretize the variable y as in [HF15], so that in a leaf problem with fixed discrete decisions, it becomes a constant. Otherwise, the introduced pump destroys the convexity of the subproblems, cf. (2.21). This is not true for an MILP-approximation of the problem. Because everything is linear in such a model, the additive y does not destroy any convexity. Discretizing the variable y instead would again lead to the same situation as in an MILP approximation of the water network design problem without pumps: a search tree where subproblems solvable in polynomial time are instead attacked by integer branching.

Another aspect related to the role of pumps is more of algorithmic nature. As seen above, they actually have the ability to augment the number of feasible integer configurations in a search tree. It is in general easier to find feasible solutions. This algorithmic advantage can be seen directly. In an equation like (2.47), y acts as a slack variable. It becomes easier to find a solution somewhere above the graph of the Hazen-Williams function instead of exactly on that graph. Another piece of evidence of this can be found in [GHHV12]. One of the several effects of presolving therein is that some Darcy-Weisbach constraints are replaced by their relaxations depicted in Figure 2.5 (b). Instead of being constrained to find a feasible point on the graph, we can find it somewhere above (or below) the graph, and the performance of the algorithm is drastically improved by presolving. Also pumps with variable speed in an operational setting somehow allow to augment the set of feasible solutions. All in all, it depends of course on the cost of the above slacks, whether an overall procedure is improved or not.

2.4.3 Computational experiments

Another way of approximating (sub-)problems in a manner that the space of feasible solutions is augmented is to use piecewise linear relaxations instead of piecewise linear approximations. In this way, it should become easier to find feasible solutions. In [GMMS12] it is reported that the switch from piecewise linear approximation to piecewise linear relaxation does not result in an overall speed-up for the problem considered in that paper, i.e., a water network operation problem minimizing the power consumption of pumps. In this concluding section, we want to report the

results of a small experiment with piecewise linear approximations and relaxations of the water network design problem described in Section 2.3.1. We proceed as in Section 6.3 in [BDL+11] as for how to obtain a piecewise linear approximation. Without going too much into detail, this amounts to approximating the inverse of the Hazen-Williams equation for each available diameter, and then deactivating the constraints for those diameters that are not chosen, with big-M constraints. In the inverse of the Hazen-Williams equation, the flow is expressed as a function of the head loss. After having computed a (non-positive) lower and a (non-negative) upper bound on the head loss, we distribute K breakpoints on each half axis. By taking the inverse, we can use this segmentation for each diameter, because the lower and upper bounds on the head loss are independent of the chosen diameter. We model the approximations by either the convex combination method or the incremental method, and optionally build piecewise linear relaxations as described in [GMMS12]. Table 2.3 reports results for $K = 2, 4, 10$ on 8 out of 9[5] instances introduced in Section 2.3.1. We report the computing times in seconds (columns time) and the best integer solution found by CPLEX 12.6 through AMPL (columns ub) within a time limit of two hours (indicated by ∞, when reached) on a single core of a 3.1 GHz quad-core machine with 1.96 GB RAM. We checked the diameter configurations of each obtained integer solution for feasibility with the simulation software EPANET. Integer solutions that were certified feasible by EPANET are marked with an asterisk in front of the corresponding objective value in Table 2.3.

The results show that it is in general very hard to find feasible solutions. The piecewise linear approximation never finds a feasible solution within the time limit for medium or large size instances. The piecewise linear relaxation does find one in some cases on medium size instances for fairly few breakpoints when using the incremental method, and the smaller size instances are usually solved faster by the piecewise linear relaxation. However, the piecewise linear approximation provides a conservative solution, in the sense that the diameter configuration is usually feasible for the nonlinear problem. For the relaxation that is not true in general. In fact, as shown by the results in Table 2.3, when the algorithm provides an optimal solution to the piecewise linearly relaxed model, it is not always feasible. Instead, when hitting the time limit, the integer solutions found by the algorithm using the relaxation were all feasible in our experiment. However, there is clearly no controlled way of setting a time limit that would guarantee finding an integer solution that is feasible. All in all, we can see that the incremental method is generally a bit faster than the convex combination method, provided an instance is solved. This confirms the observation made in [GMMS12].

[5]The new york instance originates from the slightly different practical problem of doubling pipes in an existing network, and we do not consider it here.

2.5 Unified modeling of design and operation

As mentioned before, the inherent difference of the design and the operation problems is the contrast between static and dynamic modeling, which is also intimately related to the absence and presence of active network elements. Dynamic modeling can make sense under two conditions. First of all, considering a dynamic model is useful only if exogenous parameters that have impact on the decision variables, like the demand patterns of consumers, naturally change over time. In such a situation, neglecting time completely and taking for example mean values to express these parameters results in an accuracy loss. Of course, also a discretization leads to mean values, but these mean values are based on single time windows. The second condition is that it has to make physical sense to take different decisions at different points in time. A pump, for example, can be switched on and off over time. For design problems in the form that we discuss here, this is hardly the case. The variables in the model (2.32) - (2.42) represent the decision to build a pipe with some diameter, which is not something that is easily reversed within a reasonable time horizon. In other words, decisions regarding passive elements are inherently static decisions.

Now if we want to put passive and active elements together in a unified model, that is, tackling design decisions and operational decisions at once, we have two choices. Either we consider the decisions regarding active elements (in an approximative fashion) as static as well, or we find a model that can handle static and dynamic decisions at the same time. In the first case, there arises the problem of how to correctly measure the cost of active elements. The cost of a pump, for example, is not only its installation cost, but also its variable working cost, and it is not clear how to establish it in a static model. Another problem is that, in theory, the decision of installing a pump implies that subsequently it is always working, which makes it somewhat needless to pose the operational question of when to switch it on and off.

In the second case, that is, when putting static and dynamic decisions in a unified model, an obvious problem is how to compare fixed and time-dependent costs. The weight of fixed costs decreases when considering longer time periods, which, for example, would make a bigger pipe more profitable in the context of water networks. Anyway, this raises the question of which is the correct period to consider, which then can have varying answers according to different points of view of what "correct" means.

Despite these difficulties, such a unified approach taking the first of the two choices outlined above, i.e., using a static model with operational components, is presented in [HF15] and [HFK15] in the context of gas networks. The basic structure

of that static model can in principle be applied to water networks as well. The authors consider gas transmission networks, with flow conservation constraints at the nodes and arc equations of type (2.10). Therein, the operational component $\beta_a y_a$ is forced to be zero for pipes, positive for compressors (the equivalent of pumps in gas networks) or negative for pressure regulators. The striking point is that variable y_a is modeled as a discrete variable. This leads to the fact that the leaf problems in a search tree that branches on the integer variables are convex optimization problems, as shown in Section 2.2. The first paper [HF15] is of operational type. Fixed entry and exit flows into and out of the network at certain nodes are given, and the task is to determine if the network is able to satisfy this scenario, also called *nomination* in this context. Thus, a feasibility problem without the minimization of power consumption or anything else is solved. The convex leaf problems are therefore solved by an NLP solver via different relaxation strategies. For example, one possibility is to relax the bounds on the variables q and h and minimize the bound violation. A leaf is then feasible if the optimal value of the relaxed problem is equal to zero. Networks with up to almost five hundred pipes are handled, although in the computations variables y_a's are relaxed as continuous. Hence the procedure is not able to prove infeasibility, but only the feasibility of a scenario. However, in the considered tests infeasibility did not occur. The follow-up paper [HFK15] treats the so-called topology optimization problem for the same type of gas networks. Based on the same equations, the model is now allowed to extend a network (that is infeasible for some scenario) by choosing on each arc exactly one of a discrete number of parallel network elements with different characteristics in a cost-minimal way. In principle, these elements can contain also pumps, but as a special case is contained the problem of choosing on each arc exactly one pipe out of a discrete set of pipes with varying parameters, i.e., an equivalent of the classic water network design problem. Again the convex leaf problems are solved through a relaxation. If such a relaxed leaf problem is infeasible for the original MINLP, a cutting plane is derived that is based on information from the nicely interpreted dual problem of the relaxed leaf problem. Some tests were conducted on networks with only pipes, and it is shown that the cutting planes derived from the relaxed leaf problems can significantly reduce the computing times. This again underlines the potential that lies in the exploitation of the convexity property of the leaf problems. In the next section, we show how to adapt the valid inequalities leading to the aforementioned cutting planes from gas to water networks.

2.6 Nonlinear valid inequalities for water network design problems

We now come to the derivation of nonlinear valid inequalities for the water network design problem (2.32) - (2.42). The derivation and line of exposition are similar to the one of analogue inequalities in the context of the topology optimization problem of gas networks proposed in [HFK15], and the level of detail therein is more profound. The step from gas to water networks bears slight differences in some constraints describing the problem and thus requires some care. Highlighting this transfer is the main purpose and contribution of this section. We start by considering a leaf problem of the MINLP (2.32) - (2.42), whose binary assignments satisfy (2.35). That is, we assume that $\forall\ a \in \mathcal{A}$, there is a $\ell_0(a) \in [r_a]$ such that $x_{a,\ell_0(a)} = 1$, which implies $x_{a,\ell} = 0\ \forall\ \ell \neq \ell_0(a)$. The left- and right-hand sides of (2.37) then simplify, and for brevity we denote these quantities by

$$\underline{q}_a := -\frac{\pi}{4}\overline{v}_a D^2_{a,\ell_0(a)} \tag{2.48}$$

$$\overline{q}_a := \frac{\pi}{4}\overline{v}_a D^2_{a,\ell_0(a)}. \tag{2.49}$$

Further, we denote the Hazen-Williams function on arc a by $\Phi_a(\cdot)$, that is,

$$\Phi_a(q_a) := \frac{10.7 \cdot L_a}{k_a \cdot D^{4.82}_{a,\ell_0(a)}} q_a |q_a|^{0.852}. \tag{2.50}$$

$\Phi_a(\cdot)$ can be easily checked to satisfy the assumptions of Theorem 2.1. This section is therefore based on much of what has been said in Section 2.2. The following NLPs are usually non-convex, because they contain the nonlinear functions $\Phi_a(\cdot)$ inside equality constraints. Yet, their feasible region is actually a convex set due to Theorem 2.1. The leaf problem in question becomes the following feasibility problem.

$$\sum_{a\in\delta_i^-} q_a - \sum_{a\in\delta_i^+} q_a = d_i \qquad \forall\ i \in \mathcal{N} \setminus \mathcal{N}^{src} \tag{2.51}$$

$$h_i - h_j = \Phi_a(q_a) \qquad \forall\ a \in \mathcal{A} \tag{2.52}$$

$$h_i = h_i^{src} \qquad \forall\ i \in \mathcal{N}^{src} \tag{2.53}$$

$$\underline{q}_a \leq q_a \leq \overline{q}_a \qquad \forall\ a \in \mathcal{A} \tag{2.54}$$

$$\underline{h}_i \leq h_i \leq \overline{h}_i \qquad \forall\ i \in \mathcal{N}. \tag{2.55}$$

The difference between the above system and (2.15) - (2.19) is just that in the latter,

the bounds on the variables are disregarded. We will go over to a reformulation of (2.51) - (2.55) in which we relax the bounds on the variables, but account for their violation by slack variables, that in turn are minimized in the objective function. In [HFK15], this reformulation is called the domain relaxation problem,

$$\min \sum_{i \in \mathcal{N} \setminus \mathcal{N}^{src}} \Delta_i + \sum_{a \in \mathcal{A}} \Delta_a \tag{2.56}$$

$$\text{s.t.} \quad \sum_{a \in \delta_i^-} q_a - \sum_{a \in \delta_i^+} q_a = d_i \qquad \forall\, i \in \mathcal{N} \setminus \mathcal{N}^{src} \tag{2.57}$$

$$h_i - h_j = \Phi_a(q_a) \qquad \forall\, a \in \mathcal{A} \tag{2.58}$$

$$h_i = h_i^{src} \qquad \forall\, i \in \mathcal{N}^{src} \tag{2.59}$$

$$q_a - \Delta_a \leq \overline{q}_a \qquad \forall\, a \in \mathcal{A} \tag{2.60}$$

$$h_i - \Delta_i \leq \overline{h}_i \qquad \forall\, i \in \mathcal{N} \setminus \mathcal{N}^{src} \tag{2.61}$$

$$q_a + \Delta_a \geq \underline{q}_a \qquad \forall\, a \in \mathcal{A} \tag{2.62}$$

$$h_i + \Delta_i \geq \underline{h}_i \qquad \forall\, i \in \mathcal{N} \setminus \mathcal{N}^{src} \tag{2.63}$$

$$\Delta_a \geq 0 \qquad \forall\, a \in \mathcal{A} \tag{2.64}$$

$$\Delta_i \geq 0 \qquad \forall\, i \in \mathcal{N} \setminus \mathcal{N}^{src} \tag{2.65}$$

$$q_a \in \mathbb{R} \qquad \forall\, a \in \mathcal{A} \tag{2.66}$$

$$h_i \in \mathbb{R} \qquad \forall\, i \in \mathcal{N}. \tag{2.67}$$

The relation between systems (2.15) - (2.19), (2.51) - (2.55) and (2.56) - (2.67) can easily be deduced by Theorem 2.1.

Observation 2.3. *System* (2.56) - (2.67) *has a unique solution* $(q, h, \Delta) \in \mathbb{R}^{|\mathcal{A}|+|\mathcal{N}|} \times \mathbb{R}_+^{|\mathcal{A}|+|\mathcal{N} \setminus \mathcal{N}^{src}|}$. *Furthermore, the leaf problem* (2.51) - (2.55) *is feasible if and only if in that unique solution, we have* $\Delta = 0$.

Observation 2.3 implies that the domain relaxation problem, that is at first sight a non-convex NLP, can be solved to global optimality by a local NLP solver, as already noted in Section 2.2. For example, NLP solvers like IPOPT will return a KKT point of (2.56) - (2.67) in order to certify (local) optimality. We will therefore study the KKT conditions (cf. Section 1.2.1) of (2.56) - (2.67), satisfied by any KKT point, in order to obtain some of their algebraic properties. In the following, we denote the KKT multipliers of (2.57) - (2.65), in the order of appearance of these constraints, by

$$(\mu, \lambda, \lambda^+, \lambda^-, \tilde{\lambda}) \in \mathbb{R}^{|\mathcal{N} \setminus \mathcal{N}^{src}|+|\mathcal{A}|} \times \mathbb{R}^{\mathcal{N}^{src}} \times \mathbb{R}_+^{3 \cdot (|\mathcal{A}|+|\mathcal{N} \setminus \mathcal{N}^{src}|)}.$$

In gas networks, as was anticipated in Section 2.2, the node potentials are usually

not fixed at source nodes, but in turn the balance equations are imposed. In that case, the domain relaxation problem, and especially its associated KKT multipliers, change slightly. The stationarity conditions (1.26) for the domain relaxation problem can be written as in (2.68) - (2.73). Note that there is actually no multiplier μ_i for a source node $i \in \mathcal{N}^{src}$, but we augment the following system by these variables and fix them to zero. In that way, we can write (2.69) in a unified fashion.

$$\mu_i = 0 \qquad \forall\, i \in \mathcal{N}^{src} \tag{2.68}$$

$$\mu_j - \mu_i + \lambda_a^+ - \lambda_a^- = \mu_a \Phi_a'(q_a) \qquad \forall\, a \in \mathcal{A} \tag{2.69}$$

$$\sum_{a \in \delta_i^+} \mu_a - \sum_{a \in \delta_i^-} \mu_a = \lambda_i^+ - \lambda_i^- \qquad \forall\, i \in \mathcal{N} \setminus \mathcal{N}^{src} \tag{2.70}$$

$$\sum_{a \in \delta_i^+} \mu_a - \sum_{a \in \delta_i^-} \mu_a = \lambda_i \qquad \forall\, i \in \mathcal{N}^{src} \tag{2.71}$$

$$\lambda_i^+ + \lambda_i^- + \tilde{\lambda}_i = 1 \qquad \forall\, i \in \mathcal{N} \setminus \mathcal{N}^{src} \tag{2.72}$$

$$\lambda_a^+ + \lambda_a^- + \tilde{\lambda}_a = 1 \qquad \forall\, a \in \mathcal{A}. \tag{2.73}$$

Some of these equations will be useful in the proof of the next Lemma. In [HFK15], a nice interpretation of the above system as a dual nonlinear network flow with flow variables μ_a is provided. We can now prove the validity of a first nonlinear valid inequality.

Lemma 2.4. *Let* $(q^*, h^*, \Delta^*, \mu^*, \lambda^*, \lambda^{+*}, \lambda^{-*}, \tilde{\lambda}^*)$ *be a KKT point of* (2.56) - (2.67). *Then the nonlinear inequality*

$$\sum_{a \in \mathcal{A}} \mu_a^* \Phi_a(q_a) \leq \sum_{i \in \mathcal{N} \setminus \mathcal{N}^{src}} (\lambda_i^{+*} \overline{h}_i - \lambda_i^{-*} \underline{h}_i) + \sum_{i \in \mathcal{N}^{src}} \lambda_i^* h_i^{src} \tag{2.74}$$

is valid for the leaf problem (2.51) - (2.55).

Proof. From (2.52), we get

$$\sum_{a \in \mathcal{A}} \mu_a^* \Phi_a(q_a) = \sum_{a \in \mathcal{A}} \mu_a^* (h_i - h_j),$$

where the right-hand side can be rearranged by summing over the nodes instead of summing over the arcs and then using (2.70) and (2.71):

$$\begin{aligned} \sum_{a \in \mathcal{A}} \mu_a^* (h_i - h_j) &= \sum_{i \in \mathcal{N}} h_i \left(\sum_{a \in \delta_i^+} \mu_a^* - \sum_{a \in \delta_i^-} \mu_a^* \right) \\ &= \sum_{i \in \mathcal{N} \setminus \mathcal{N}^{src}} h_i \left(\lambda_i^{+*} - \lambda_i^{-*} \right) + \sum_{i \in \mathcal{N}^{src}} h_i \lambda_i^*. \end{aligned}$$

Taking into account the sign of λ^{+*} and λ^{-*} together with (2.53) and (2.55) gives (2.74). □

The inequality (2.74) could in theory be used as a valid inequality inside a tree-search method for the water network design problem. It bears, however, two difficulties, the first one being the fact that it is only locally valid, i.e., valid for a leaf problem. One could argue that globally valid inequalities are more desirable than local ones in general, but this is not always true, and not even the point here. Instead, the way the inequality is derived makes its use as a locally valid inequality useless: its derivation hypothesizes that the domain relaxation (2.56) - (2.67) of a leaf problem (2.51) - (2.55) has been solved in order to get a valid inequality for the leaf problem itself. However, by Observation 2.3, once we have solved the domain relaxation problem, there is no need anymore to solve the leaf problem. We will address the issue of local validity later on and show how to lift the inequality to a globally valid one. The second difficulty is more related to the computational efficiency of the use of a nonlinear inequality and was also noted in [HFK15]. The inequality (2.74) actually represents a non-convex constraint, and thus, if added to the original MINLP, would trigger the need for (additional) spatial branching. More details about this are given in [Hum14, Chapter 4]. Therefore, in [HFK15], a linear underestimator of the nonlinear function involved in the inequality is sought, that in turn can be used more efficiently inside a tree-search algorithm for a problem like (2.32) - (2.42). Before that, yet another step is taken, that modifies (2.74) to another valid nonlinear inequality, but makes the underestimation of the latter easier and also algebraically elegant. We will go the same way but, due to the aforementioned difference between gas and water networks, first have to introduce some additional notation.

It turns out that the presence of multiple source nodes makes the formalization of some arguments more difficult, which is why we assume that there is one distinguished $i_0 \in \mathcal{N}^{src}$. The case $|\mathcal{N}^{src}| = 1$ is then contained as a special case in the following derivation. With every vector of flows $q \in \mathbb{R}^{|\mathcal{A}|}$ in the network that satisfies (2.51), $\forall\, i \in \mathcal{N}^{src} \setminus \{i_0\}$ we define the quantities

$$\tilde{q}_i := -\left(\sum_{a \in \delta_i^-} q_a - \sum_{a \in \delta_i^+} q_a \right). \tag{2.75}$$

For $i \in \mathcal{N}^{src} \setminus \{i_0\}$, the absolute value of this quantity represents the amount of flow that is inserted into the network at source nodes i. The whole network flow can then be interpreted as follows: the whole amount of flow in the network, given by $F := \sum_{i \in \mathcal{N} \setminus \mathcal{N}^{src}} d_i$, is provided at source node i_0, and part of it is transported to the other source nodes. In particular, the quantity $\tilde{q}_i$ is transported on some

imaginary arc $\tilde{a}_i = (i_0, i)$ with fixed pressure loss $h^{src}_{i_0} - h^{src}_i$ to source node i. In this interpretation, the network has fixed demands at each node $i \in \mathcal{N}$: demand d_i at node $i \in \mathcal{N} \setminus \mathcal{N}^{src}$, negative but fixed demand $-F$ at i_0, and an imaginary and fixed demand of zero at all other source nodes. In particular, we can write out balance equations also at source nodes:

$$\sum_{a \in \delta^-_{i_0}} q_a - \left(\sum_{a \in \delta^+_{i_0}} q_a + \sum_{i \in \mathcal{N}^{src} \setminus \{i_0\}} \tilde{q}_i \right) = -F \tag{2.76}$$

$$\left(\sum_{a \in \delta^-_i} q_a + \tilde{q}_i \right) - \sum_{a \in \delta^+_i} q_a = 0 \qquad \forall\, i \in \mathcal{N}^{src} \setminus \{i_0\}. \tag{2.77}$$

For ease of notation, we further define the linear function

$$\psi(q) := \sum_{i \in \mathcal{N}^{src} \setminus \{i_0\}} \tilde{q}_i (h^{src}_{i_0} - h^{src}_i).$$

With this additional notation, we can formulate and prove the following lemma.

Lemma 2.5. *Let* $(q^*, h^*, \Delta^*, \mu^*, \lambda^*, \lambda^{+*}, \lambda^{-*}, \tilde{\lambda}^*)$ *be a KKT point of* (2.56) - (2.67). *Then the identity*

$$\sum_{a \in \mathcal{A}} (q_a - q^*_a) \Phi_a(q_a) = \psi(q^*) - \psi(q) \tag{2.78}$$

is fulfilled by any feasible solution of the leaf problem (2.51) - (2.55).

Proof. Definition (2.75) and its interpretation allows us to use the fact that the difference flow $q_a - q^*_a$ consists of circulations only, provided we consider the imaginary arcs as well. That is, $\forall\, i \in \mathcal{N} \setminus \mathcal{N}^{src}$ we know from the flow conservation constraints (2.51) and (2.57) that

$$\sum_{a \in \delta^-_i} (q_a - q^*_a) - \sum_{a \in \delta^+_i} (q_a - q^*_a) = 0.$$

For source node i_0 instead, by (2.76) we can write

$$\sum_{a \in \delta^-_{i_0}} (q_a - q^*_a) - \sum_{a \in \delta^+_{i_0}} (q_a - q^*_a) - \sum_{i \in \mathcal{N}^{src} \setminus \{i_0\}} (\tilde{q}_i - \tilde{q}^*_i) = 0,$$

while, $\forall\ i \in \mathcal{N}^{src} \setminus \{i_0\}$, we have

$$\sum_{a\in\delta_i^-}(q_a - q_a^*) + (\tilde{q}_i - \tilde{q}_i^*) - \sum_{a\in\delta_i^+}(q_a - q_a^*) = 0.$$

If we multiply the above equations by h_i and sum over all $i \in \mathcal{N}$, we get

$$\sum_{i\in\mathcal{N}}\left(\sum_{a\in\delta_i^-}(q_a - q_a^*) - \sum_{a\in\delta_i^+}(q_a - q_a^*)\right)h_i + \sum_{i\in\mathcal{N}^{src}\setminus\{i_0\}}(\tilde{q}_i - \tilde{q}_i^*)h_i^{src} - h_{i_0}^{src}\sum_{i\in\mathcal{N}^{src}\setminus\{i_0\}}(\tilde{q}_i - \tilde{q}_i^*) = 0.$$

By changing the summation over the nodes on the left-hand side to summation over the arcs leads to

$$\sum_{a\in\mathcal{A}}(h_j - h_i)(q_a - q_a^*) + \sum_{i\in\mathcal{N}^{src}\setminus\{i_0\}}(\tilde{q}_i - \tilde{q}_i^*)(h_i^{src} - h_{i_0}^{src}) = 0,$$

which with (2.52) can be rearranged to (2.78). □

Combining Lemmas 2.4 and 2.5 gives the following immediate corollary.

Corollary 2.6. *Let* $(q^*, h^*, \Delta^*, \mu^*, \lambda^*, \lambda^{+*}, \lambda^{-*}, \tilde{\lambda}^*)$ *be a KKT point of* (2.56) - (2.67). *Then for any* $\zeta \geq 0$ *and* $\xi \in \mathbb{R}$, *the nonlinear inequality*

$$\sum_{a\in A}\big(\zeta\mu_a^* + \xi(q_a - q_a^*)\big)\Phi_a(q_a) \leq \zeta\left(\sum_{i\in\mathcal{N}\setminus\mathcal{N}^{src}}\big(\lambda_i^{+*}\overline{h}_i - \lambda_i^{-*}\underline{h}_i\big) + \sum_{i\in\mathcal{N}^{src}}\lambda_i^* h_i^{src}\right) + \xi\big(\psi(q^*) - \psi(q)\big) \tag{2.79}$$

is valid for the leaf problem (2.51) - (2.55).

One of the reasons for the modification of the valid inequality (2.74) to (2.79) is that each term on the left-hand side of (2.74) is not bounded by below, but in fact one can show that $\lim_{q_a\to+\infty}\Phi_a(q_a) = +\infty$ and $\lim_{q_a\to-\infty}\Phi_a(q_a) = -\infty$. Instead, each term on the left-hand side of (2.79) tends to either $+\infty$ or $-\infty$ for both decreasing and increasing q_a, depending on the sign of the remaining coefficients. In addition, some algebraic properties based on the stationarity conditions help to simplify the proposed underestimator significantly. We will show this in the next section.

2.6.1 The nice algebra of nonlinear network flows

We now derive the underestimator of (2.79) in a straightforward way. By imposing the same slope that is used in [HFK15] on this one-dimensional linear function, we can calculate the intercept in such a way that the resulting function is the tightest underestimator among all linear functions with that particular slope on the domain of the leaf problem. Using the properties (2.68) - (2.73), the resulting linear valid inequality can be rearranged to a constant inequality. Everything is collected in the following proposition.

Proposition 2.7. *Let* $(q^*, h^*, \Delta^*, \mu^*, \lambda^*, \lambda^{+*}, \lambda^{-*}, \tilde{\lambda}^*)$ *be a KKT point of* (2.56) - (2.67)*, set* $\mu_i^* = 0 \ \forall \ i \in \mathcal{N}^{src}$ *and* $\forall \ a \in \mathcal{A}$ *define*

$$\tau_a := \inf_{\underline{q}_a \leq q_a \leq \overline{q}_a} \left\{ \left(\zeta\mu_a^* + \xi(q_a - q_a^*)\right)\Phi_a(q_a) - \xi(h_i^* - h_j^*)q_a - \zeta(\mu_i^* - \mu_j^* - \lambda_a^{+*} + \lambda_a^{-*})q_a \right\}.$$

Then the constant inequality

$$\sum_{a\in\mathcal{A}} \tau_a \leq \zeta \Bigg(\sum_{i\in\mathcal{N}\setminus\mathcal{N}^{src}} \left(\lambda_i^{+*}\overline{h}_i - \lambda_v^{-*}\underline{h_i}\right) + \sum_{i\in\mathcal{N}\setminus\mathcal{N}^{src}} \mu_i^* d_i + \sum_{a\in\mathcal{A}} (\lambda_a^{+*}\overline{q}_a - \lambda_a^{-*}\underline{q}_a) + \sum_{i\in\mathcal{N}^{src}} \lambda_i^* h_i^{src} \Bigg) + \xi \left(\psi(q^*) + \sum_{i\in\mathcal{N}\setminus\mathcal{N}^{src}} h_i^* d_i - h_{i_0}^{src} F \right) \tag{2.80}$$

is valid for the leaf problem (2.51) - (2.55).

Proof. We first establish two identities that will be used further down. In particular, using similar changes of summation as before and the balance equations for all nodes (cf. (2.76) - (2.77)), we can write

$$\begin{aligned}
\xi \sum_{a\in\mathcal{A}} (h_j^* - h_i^*)q_a &= \xi \sum_{i\in\mathcal{N}} h_i^* \left(\sum_{a\in\delta_i^-} q_a - \sum_{a\in\delta_i^+} q_a \right) \\
&= \xi \cdot \sum_{i\in\mathcal{N}\setminus\mathcal{N}^{src}} h_i^* d_i + \xi h_{i_0}^{src} \left(\sum_{i\in\mathcal{N}^{src}\setminus\{i_0\}} \tilde{q}_i - F \right) - \xi \cdot \sum_{i\in\mathcal{N}^{src}\setminus\{i_0\}} h_i^{src}\tilde{q}_i \\
&= \xi \cdot \sum_{i\in\mathcal{N}\setminus\mathcal{N}^{src}} h_i^* d_i - \xi h_{i_0}^{src} F + \xi \cdot \sum_{i\in\mathcal{N}^{src}\setminus\{i_0\}} (h_{i_0}^{src} - h_i^{src})\tilde{q}_i \\
&= \xi \cdot \sum_{i\in\mathcal{N}\setminus\mathcal{N}^{src}} h_i^* d_i - \xi h_{i_0}^{src} F + \xi\psi(q).
\end{aligned} \tag{2.81}$$

Then, remembering that $\mu_i^* = 0 \; \forall \; i \in \mathcal{N}^{src}$, we get

$$\zeta \sum_{a \in \mathcal{A}} (\mu_j^* - \mu_i^*) q_a = \zeta \cdot \sum_{i \in \mathcal{N} \setminus \mathcal{N}^{src}} \mu_i^* d_i. \tag{2.82}$$

Now by definition, we know that on the for a leaf problem valid domain of q_a, we have

$$\tau_a \leq \big(\zeta \mu_a^* + \xi(q_a - q_a^*)\big)\Phi_a(q_a) - \xi(h_i^* - h_j^*)q_a - \zeta(\mu_i^* - \mu_j^* - \lambda_a^{+*} + \lambda_a^{-*})q_a.$$

Summation over the arcs leads to

$$\begin{aligned}\sum_{a \in \mathcal{A}} \tau_a \leq & \sum_{a \in \mathcal{A}} \big(\zeta \mu_a^* + \xi(q_a - q_a^*)\big)\Phi_a(q_a) + \xi \sum_{a \in \mathcal{A}} (h_j^* - h_i^*)q_a \\ & + \zeta \sum_{a \in \mathcal{A}} (\mu_j^* - \mu_i^*)q_a + \zeta \sum_{a \in \mathcal{A}} (\lambda_a^{+*} - \lambda_a^{-*})q_a.\end{aligned}$$

The first sum of the right-hand side can be bounded by (2.79), while the second and third sum can be substituted by (2.81) and (2.82), so that we get

$$\begin{aligned}\sum_{a \in \mathcal{A}} \tau_a \leq & \zeta \left(\sum_{i \in \mathcal{N} \setminus \mathcal{N}^{src}} \big(\lambda_i^{+*} \overline{h}_i - \lambda_i^{-*} \underline{h}_i\big) + \sum_{i \in \mathcal{N}^{src}} \lambda_i^* h_i^{src} \right) + \xi\big(\psi(q^*) - \psi(q)\big) \\ & + \xi \cdot \sum_{i \in \mathcal{N} \setminus \mathcal{N}^{src}} h_i^* d_i - \xi h_{i_0}^{src} F + \xi \psi(q) + \zeta \cdot \sum_{i \in \mathcal{N} \setminus \mathcal{N}^{src}} \mu_i^* d_i + \zeta \sum_{a \in \mathcal{A}} (\lambda_a^{+*} - \lambda_a^{-*}) q_a.\end{aligned}$$

Bounding the last sum by use of the lower and upper bounds on q_a that are valid in the leaf problem, and rearranging gives (2.80). □

We now come to the point of lifting the locally valid inequality to a global one. This turns out to be rather easy, at least formally.

Observation 2.8. *In the hypotheses of Lemmata 2.4 and 2.5, Corollary 2.6 and Proposition 2.7, it is not required that the given KKT point comes from the domain relaxation of the leaf problem for which the inequalities or identities are valid. In fact, it can be a KKT point of the domain relaxation of any leaf problem of the MINLP* (2.32) - (2.42).

With Observation 2.8 in mind, lifting the inequality (2.80), that is still only locally valid, to a globally valid one is straightforward. We just define the quantities in (2.48) - (2.50) for any $\ell \in [r_a]$:

$$\underline{q}_{a,\ell} := -\frac{\pi}{4} \overline{v}_a D_{a,\ell}^2$$

$$\overline{q}_{a,\ell} := \frac{\pi}{4}\overline{v}_a D_{a,\ell}^2$$
$$\Phi_{a,\ell}(q_a) := \frac{10.7 \cdot L_a}{k_a \cdot D_{a,\ell}^{4.82}} q_a |q_a|^{0.852}.$$

Thus, we can define $\tau_{a,\ell}$ and substitute all quantities in (2.80), that depend on the index ℓ, by a weighted sum of the binary variables $x_{a,\ell}$.

Corollary 2.9. *Let $(q^*, h^*, \Delta^*, \mu^*, \lambda^*, \lambda^{+*}, \lambda^{-*}, \tilde{\lambda}^*)$ be a KKT point of any leaf problem of the MINLP (2.32) - (2.42). Further, $\forall\ a \in \mathcal{A}$ and $\ell \in [r_a]$ define*

$$\tau_{a,\ell} := \inf_{\underline{q}_{a,\ell} \le q_a \le \overline{q}_{a,\ell}} \left\{ \left(\zeta\mu_a^* + \xi(q_a - q_a^*)\right) \Phi_{a,\ell}(q_a) - \xi(h_i^* - h_j^*)q_a + \zeta(\mu_i^* - \mu_j^* - \lambda_a^{+*} + \lambda_a^{-*})q_a \right\}.$$

Then the linear inequality

$$\begin{aligned}\sum_{a\in\mathcal{A}} \sum_{\ell=1}^{r_a} \tau_{a,\ell} x_{a,\ell} \le & \zeta \left(\sum_{i \in \mathcal{N}\setminus\mathcal{N}^{src}} \left(\lambda_i^{+*}\overline{h}_i - \lambda_v^{-*}\underline{h}_i\right) + \sum_{i\in\mathcal{N}\setminus\mathcal{N}^{src}} \mu_i^* d_i + \sum_{i\in\mathcal{N}^{src}} \lambda_i^* h_i^{src} \right) \\ & + \zeta \sum_{a\in\mathcal{A}} \left((\lambda_a^{+*} + \lambda_a^{-*}) \frac{\pi}{4} \overline{v}_a \cdot \sum_{d=1,\dots,r_a} D_{a,\ell}^2 x_{a,\ell} \right) \\ & + \xi \left(\psi(q^*) + \sum_{i\in\mathcal{N}\setminus\mathcal{N}^{src}} h_i^* d_i - h_{i_0}^{src} F \right) \end{aligned} \tag{2.83}$$

is valid for (2.32) - (2.42).

Finally, the question of how this inequality can actually be used in practice arises. A natural approach would be to solve (2.32) - (2.42) with a spatial branch-and-bound algorithm, cf. Section 1.2.4, and solve the domain relaxation problem at any leaf problem that is encountered during the search process by a local NLP solver, so as to get a KKT point. In [HFK15] for example, SCIP and IPOPT are used to do precisely that. A hurdle in order to numerically obtain the cutting plane (2.83) from that KKT point can then be seen in the computation of the coefficients $\tau_{a,\ell}$. In [HFK15] is shown that under certain conditions, ζ and ξ can be chosen in such a way that the infimum in the definition of $\tau_{a,\ell}$ is always attained at q_a^* and the $\tau_{a,\ell}$ have a closed-form formula. The cut is therefore added whenever these conditions are fulfilled. We conducted preliminary experiments with these cutting planes on the water network design instances presented in Section 2.3.1. These experiments have so far not been very successful [Wie14]. Intuitively, the underestimation of the original nonlinear valid inequality is too rough. Further ideas of how to obtain additional linear underestimators have been established, and possible future work consists in the implementation of these ideas.

2.7 Outlook

We see two general potential future research directions, one of which concerns more the modeling part, whereas the other one is of algorithmic nature. The first aspect is the unification of design and operation problems. Despite the difficulties mentioned in Section 2.5, this is an area that could be tackled in the future. Therefore, robust and unified ways of modeling the costs of active and passive network elements are necessary. Additionally, it seems not yet clear which is the right dynamic model for the different network elements. For example, is the discretized version of the water hammer equations (2.8)-(2.9) really necessary to describe the water dynamics in pipes or is a static Hazen-Williams equation in every time step enough? A model that balances well accuracy and tractability should be determined. Furthermore, we think that the exploitation of convex leaf problems as described in Section 2.2 bears a lot of potential, because it has the power of reducing the complexity of the problem (maybe at the cost of discretizing some continuous parameters). First steps in this direction on the operational side have been made, as described at the end of Section 2.3.2, and the further combination of such properties with MINLP techniques seems fruitful.

3 Mixed Integer Programming with indicator constraints

In computer science, every programming language provides a syntax for conditional constructs like the simple, well known *if-then*. These basic structures have found their way into MIP some time ago. Think back at the TSPTW model (1.32) - (1.37) in Section 1.5.1, where we saw a fist example in (1.38). The condition therein, here generalized to one involving a more general function $g(\cdot)$,

$$[z = 1] \implies [g(x) \leq 0],$$

translates nothing else then the logical implication "if variable z is equal to one, then impose the constraint that $g(x)$ has o be non-negative." In other words, the so-called indicator variable z indicates whether the constraint $g(x) \leq 0$ is *switched on or off*. CPLEX for example supports this simple syntax since around 2006, and the software LogMIP [VG99] provides even more functionality for directly expressing logical expressions in a Mathematical Programming model. Of course, once a Mathematical Programming software gives its users the possibility of expressing such constructs, it has to implement means to handle them algorithmically. The case of indicator constraints in Mixed Integer Programming is the topic of this chapter: we will analyze ways to reformulate indicator constraints with mixed integer variables.

Indicator constraints can be found in the literature in a number of applications. This happens either because they model explicit logical arguments like "if facility j is inactive, then no client i can be assigned to it", as in the facility location [CFN77]; or because the Mathematical Programming formulation is constructed by imposing a specific order through (otherwise implicit) logical implications like "either job i is executed on machine k before job j or vice versa", as in Job Shop Scheduling introduced in Section 1.5.2. That is, besides *if-then*, also structures of *if-then-else*-type are encountered. Of course, the TSPTW model (1.32) - (1.37) is another example. Devising efficient and computationally effective methods to deal with logical implications is one of the most fundamental needs to enhance the Mathematical Programming solvers' capability of facing real-world optimization problems [Lod10].

It may happen that a single indicator variable z controls more than one constraint. In order to elegantly express this circumstance, on can collect the functions involved in all of these constraints, say $g_i(\cdot),\ i = 1, \ldots, \ell$, and denote the intersection of their sublevel sets by S, i.e.,

$$S = \{x \in \mathbb{R}^{\tilde{n}} \mid g_i(x) \leq 0,\ i = 1, \ldots, \ell\}. \tag{3.1}$$

Then, the optimization problem we consider in this chapter is of the general form

$$\begin{aligned}
\min \quad & f(x, z) && && (3.2)\\
\text{s.t.} \quad & h_i(x, z) \leq 0 && \forall\ i = 1, \ldots, m_1 && (3.3)\\
& \left.\begin{aligned} [z_k = 0] &\implies [x \in S_0^k] \\ [z_k = 1] &\implies [x \in S_1^k] \end{aligned}\right\} && \forall\ k = 1, \ldots, K && (3.4)\\
& x \in \mathbb{R}^{\tilde{n}-p} \times \mathbb{Z}^p && && (3.5)\\
& z \in \{0, 1\}^K. && && (3.6)
\end{aligned}$$

System (3.2) - (3.6) can be interpreted in the following way. One wants to minimize a function $f(\cdot)$ subject to a set of constraints. So-called global constraints are expressed by the functions $h_i(\cdot)$. Moreover, K pairs of logical implications are given, each one involving a binary indicator variable z_k. This variable indicates that either x is constrained to belong to the set S_0^k or to the set S_1^k, that we both assume to be of the form (3.1). In other words, each z_k indicates whether all of the constraints defining either of the two sets are imposed or not. System (3.2) - (3.6) is thus flexible enough to model indicator constraints of type *if-then-else*. In fact, a single indicator variable above cannot only switch on some constraints by being equal to one, but also by being equal to zero. All in all, the essence of a system like (3.2) - (3.6) is the fact that imposing one or more constraints of the form $g_i^k(x) \leq 0$ is linked to discrete decisions.

In order to formulate system (3.2) - (3.6) as a Mathematical Programming problem one needs to remove the logical implications and write explicit constraints. We have already seen the most straightforward way of doing that in several examples in Section 1.5. It is given by the big-M method, where constraints are activated or deactivated by multiplying the binary indicator variable by a very large (precomputed) constant. We will revise this method in more detail in Section 3.1. Alternatively, one can use Disjunctive Programming techniques, which are the main topic of the present chapter. A general introduction to this has already been given in Section 1.4, and in Section 3.2 we will recall the most important aspects for indicator constraints. If (3.2) - (3.6), reduced by (3.4), is an MINLP or even an MILP, then both

techniques can again lead to an MINLP or an MILP. Either of the two techniques bear some drawbacks, in the case of Disjunctive Programming given by the fact that the obtained reformulations lead to dealing with large NLPs, defined in lifted spaces, i.e., with an increased number of variables. This implies difficulties with practically solving these NLPs in general. In this chapter we review the relevant literature on mathematical optimization with logical implications that attempts to avoid the issue of dealing with large NLPs. In particular, we review some existing results that allow to work in the original space of variables for two relevant special cases of (3.2) - (3.6) in Sections 3.3.1 and 3.3.2. We also shed particular light on the resulting structures when the involved functions are affine in Section 3.3.3. Then, we significantly extend these special cases by considering more general sets S_k^j in Section 3.4.1. In Section 3.4.2 we will go over to a system that even allows for indicator variables that are not necessarily binary. The latter two sections represent the main theoretical contribution of this chapter.

The results reviewed and proposed in Sections 3.3.3 - 3.4.2, if applied in a straightforward way, give the possibility of writing down different, potential MI(N)LP formulations of (3.2) - (3.6). Computational experiments comparing Disjunctive Programming formulations in the original space of variables with big-M ones, presented in Section 3.5, show that the former are computationally viable and promising. However, we highlight in Section 3.6 that the obtained formulations can be seen as parametrized by the bounds of the involved variables. In order to benefit from this algorithmically, in Section 3.6.1 we will review some recently proposed algorithmic principles that can be applied in this context. Thereby inspired, in Section 3.6.2 we will present an experimental, MILP-based algorithm for a special case of (3.2) - (3.6) and present computational results for the Job Shop Scheduling problem. Here, the connection to Constraint Programming, where propagation techniques are crucial and sophisticated, will be drawn. We will thus investigate the incorporation of such techniques for bound tightening purposes into the presented algorithm.

In some sense, this chapter can be seen in the light of the attempt to incorporate "new" types of constraints into MIP, or at least to reformulate them by MIP. Especially in Section 3.2, the connection to so-called disjunctive constraints, that are, for example, an omnipresent tool in CP for scheduling problems, will become clear. This chapter is based on joint work with Pierre Bonami, Andrea Lodi and Andrea Tramontani, and its main parts have been published in [BLTW15].

3.1 BigM constraints

The first modeling technique for expressing an indicator constraint, as was anticipated several times now, is the widely known big-M method. It is very straightforward and works as follows. Let us assume that the discrete decision linked to the constraint $g(x) \leq 0$ is modeled by the binary variable z. Then, in a Mathematical Programming formulation, one can impose the constraint

$$g(x) \leq M \cdot (1 - z), \tag{3.7}$$

where M is a very large positive constant. If $z = 1$, then constraint $g(x) \leq 0$ is imposed. Conversely, if $z = 0$, then constraint (3.7) is satisfied by any value of $x \in \mathcal{F}$, where $\mathcal{F}$ describes the feasible set for x (e.g., the feasible region of the underlying MI(N)LP), and provided that $M \geq \sup_{x \in \mathcal{F}} g(x)$. Such a requirement for the definition of M already leads to a major difficulty encountered in Mathematical Programming with indicator constraints: the quantity $\sup_{x \in \mathcal{F}} g(x)$ might not be easily (or not even at all) computable. This might be the case, for example, if the set $\mathcal{F}$ itself is unbounded. In such cases, setting a reasonably high value of M will usually do the job in practice, but there is no theoretical guarantee that the system with a big-M constraint (3.7) is actually equivalent to the original one. In fact, when the set $\mathcal{F}$ is unbounded, the indicator constraints might not be representable as an MILP[1] and there might be no representation too by an MINLP with a convex feasible region[2]. If, instead, one assumes that this difficulty does not occur, i.e., it is possible, in practice, to compute $\sup_{x \in \mathcal{F}} g(x)$ or at least an upper bound on this value, then the big-M method leads to a valid reformulation of (3.2) - (3.6). For example, when we deal with affine functions and bounded variables, it is possible to obtain a valid big-M constraint.

Example 3.1. *Assume that $g(x) = a_0 + a^T x$ and that the variables x are bounded, $\underline{x}_i \leq x_i \leq \overline{x}_i$. It is easy to see that a safe choice for the big-M is then*

$$M := a_0 + \sum_{a_i > 0} a_i \overline{x}_i + \sum_{a_i < 0} a_i \underline{x}_i. \tag{3.8}$$

We give two examples of the application of the big-M choice in (3.8)*. In the TSPTW model* (1.32) - (1.37)*, one can simply set*

$$M_{ij} := u_i - l_j + p_i + t_{ij}.$$

[1]The reader is referred to [JL84] for a definition of sets that are MILP representable.
[2]The reader is referred to [HL14] for an example.

Analogously, in the JSS model (1.43) - (1.49)*, we set*

$$M_{ij}^k := \overline{s}_{ik} - \underline{s}_{jk} + t_i$$
$$\tilde{M}_{ij}^k := \overline{s}_{jk} - \underline{s}_{ik} + t_j,$$

where the bounds $\overline{s}$ *and* $\underline{s}$ *are calculated as in* (1.57) - (1.59) *(based on the disjunctive graph of the original JSS instance without any binary fixings).*

However, even under the assumptions of Example 3.1, the big-M method has two main drawbacks. The first one is trivially rooted in numerical risks associated with choosing a big-M value such that $1/M$ comes close to the machine precision, or, in any case, to the tolerances that are used by any Mathematical Programming solver working in floating point arithmetics (see, e.g., [KAA+11]). The other drawback affects on a more algorithmic level the current generation of MI(N)LP solvers, i.e., the solution method that they implement. More precisely, MI(N)LP solvers heavily rely on the iterative solution of the continuous relaxation of the given MI(N)LP, as we have seen in Sections 1.1.2 and 1.2.2. At the same time, it is well-known that the big-M formulations involving constraints (3.7) are characterized by continuous relaxations whose dual bounds depend on the value of M, but which are typically very weak, i.e., very far away from the optimal solution value (see, e.g., [Lod10]). This is because, in the continuous relaxation, a value of the binary variable z very close to one is enough to satisfy a constraint, thus deactivating the original implication $g(x) \leq 0$. In [BBF+14] can be found a recently proposed way of improving on both of the above difficulties by strengthening the M values within the branch-and-bound tree by domain propagation and separation of local cutting planes. We will explain this approach in more detail in Section 3.6.1.

3.2 Disjunctive Programming for indicator constraints

We announced the second way of reformulating (3.2) - (3.6) as an MI(N)LP to be given by Disjunctive Programming techniques. In this section, we show how to cast indicator constraints in the terminology of what has been presented in Section 1.4. We will make a change of notation that allows us to immediately see the relation of indicator constraints to Disjunctive Programming, but it will also facilitate the notation of more general systems than (3.2) - (3.6) in Section 3.4.2. We consider a system involving the union of sets,

$$\min \quad f(x,z) \tag{3.9}$$
$$\text{s.t.} \quad h_i(x,z) \leq 0 \qquad \forall\ i = 1,\dots,m_1 \tag{3.10}$$

$$(x, z_k) \in \bigcup_{j \in \mathcal{J}} \Gamma_j^k \qquad \forall\ k = 1, \ldots, K \tag{3.11}$$

$$x \in \mathbb{R}^{\tilde{n}-p} \times \mathbb{Z}^p \tag{3.12}$$

$$z \in \{0,1\}^K, \tag{3.13}$$

where $\Gamma_j^k := S_j^k \times \{j\}$. Unlike (3.2) - (3.6), where z_k are binary variables, in the Disjunctive Programming problem (3.9) - (3.13) we can consider an arbitrary set of disjunctive terms indexed in $\mathcal{J}$. Disjunctions with two disjunctive terms, leading to binary variables, are contained as a special case. The theory of Disjunctive Programming, especially Theorem 1.11 allows us to manage the union of bodies Γ_j^k as in (3.11), provided they are convex sets. Then, we can replace $\cup_{j\in\mathcal{J}}\Gamma_j^k$ by $\text{conv}\left(\cup_{j\in\mathcal{J}}\Gamma_j^k\right)$ and (possibly) end up with an MI(N)LP. Note that Theorem 1.11 can be applied to slightly more general disjunctive sets than the one in (3.11), because the convex sets Γ_k^j, whose union is taken therein, already have a specific structure. However, this is not true for the sets S_k^j.

The above constitutes the second way of dealing with logical implications, and it is somewhat opposite to the big-M method. Indeed, the big-M drawback of weak continuous relaxations is obviously overcome because there is no tighter convex relaxation of a single disjunction than its convex hull. As anticipated in Section 1.4, two difficulties arise, though. First, we do not know if the closure of the convex hull has an algebraic representation. This is related to the non-differentiability issue mentioned therein, and also the reason why up to now, we always stressed that we only possibly end up with MI(N)LPs. There has been and still is active research going on regarding this issue of the general Theorem 1.11, see, e.g., [GL03, Fur14]. Second, the convex hull in Theorem 1.11 is described as the projection of a higher-dimensional set, and this can be prohibitive in practice. The focus of the present chapter is on reviewing and providing new results that deal with this second difficulty. This is done by projecting out additional variables, which is, of course, a hard task in its most general form, but possible in several special cases that are presented in the next sections.

It is worth noting that a somewhat parallel representation of optimization problems including logical implications in terms of boolean variables is given by the notion of Generalized Disjunctive Programming (GDP), also at the basis of the aforementioned software LogMIP. For a recent review on the GDP modeling paradigms, we refer the reader to [GT13], and to [LG00, GL03] for the algorithmic aspects. In GDP terms, system (3.9) - (3.13) can be equivalently expressed as

$$\min \quad f(x) + \sum_{k=1}^{K} c_k$$

$$
\begin{aligned}
\text{s.t.} \quad & h_i(x,z) \leq 0 && \forall\, i = 1,\dots,m_1 \\
& \bigvee_{j \in \mathcal{J}_k} \begin{bmatrix} Y_j^k \\ g_j^k(x) \leq 0 \\ c_k = c_j^k \end{bmatrix} && \forall\, k = 1,\dots,K \\
& \Omega(Y) = \texttt{true} \\
& 0 \leq x \leq U \\
& c \in \mathbb{R}^K \\
& Y_j^k \in \{\texttt{true}, \texttt{false}\}
\end{aligned}
$$

where $\Omega(Y)$ denotes logical propositions, often but not necessarily assumed to be expressed in Conjunctive Normal Form [Pfa11]. Although technically equivalent, in the remainder of the chapter we will work with notation (3.9) - (3.13) and abandon the above GDP representation.

3.3 Single disjunctions

In this section we review two special cases in which the convex hull description of Theorem 1.11, applied to (3.9) - (3.13), can be projected onto the space of the original variables. More precisely, we assume that the appearing disjunctions are *unrelated* and we consider a single disjunction at a time (thus, we can drop the index k throughout the section). The considered special cases have $|\mathcal{J}| = 2$ and are such that one term of the disjunction is either a single point (Section 3.3.1) or a box (Section 3.3.2). Further, in the following (including Section 3.4) we assume that the subset of variables needed to express the disjunction is of dimension n. In other words, for $j \in \mathcal{J}$, we assume the sets S_j to be subsets of $\mathbb{R}^n$. Note that this might be a subspace of the original space $\mathbb{R}^{\tilde{n}}$ in (3.9) - (3.13). The results in this section and those in Section 3.4 can be used in a program like (3.9) - (3.13) by embedding everything into $\mathbb{R}^{\tilde{n}}$.

3.3.1 Constraint vs. Nucleus

We first consider the case in which either a subset of variables (those involved in the disjunction) is forced to zero, or a set of constraints is imposed. This occurs in many applications, for example in uncapacitated facility location with quadratic costs [GLW07], stochastic service systems design [Elh06], scheduling with controllable processing times [AAG09] or the unit commitment problem [FG06]. In the notation of (3.9) - (3.13), this special case can be written as

$$S_0 = \{x \in \mathbb{R}^n \mid x = 0\}$$

$$S_1 = \{x \in \mathbb{R}^n \mid \mathbf{l}_1 \leq x \leq \mathbf{u}_1, g_{1,i}(x) \leq 0,\ i = 1, \ldots, \ell\}.$$

In this situation, Theorem 1.11 translates into the following, as has been proven in [GL12]

Theorem 3.2. *Let $\mathcal{J} = \{0, 1\}$ and for $j \in \mathcal{J}$ define $\Gamma_j := S_j \times \{j\}$, with S_j specified as above and non-empty. We then have that if Γ_1 is a convex set,* $\text{conv}(\Gamma_0 \cup \Gamma_1) = \text{cl}(\Gamma)$*, where*

$$\Gamma = \begin{Bmatrix} (x, z) \in \mathbb{R}^{n+1} \\ \tilde{g}_{1,i}(x, z) \leq 0, \quad \forall\ i = 1, \ldots, \ell \\ z\mathbf{l}_1 \leq x \leq z\mathbf{u}_1 \\ 0 < z \leq 1 \end{Bmatrix}.$$

In this first step, a fundamental property that persists throughout all the results in the following sections can be seen. The indicator variable z of the initial indicator constraint (or disjunction) takes over the role of the weight or multiplier inside the perspective function $\tilde{g}_i(\cdot)$. We make three remarks. First, Theorem 3.2 can also be formulated for the situation in which on the zero side, the variables are not forced to zero but to an arbitrary single point. Then, everything can be shifted to 0. Second, Theorem 3.2 is also extendable to the situation in which on the zero side, the set S_0 is not a single point, but a ray [GL12]. The two corresponding sets would be

$$\hat{S}_0 = \{(x, y) \in \mathbb{R}^{n+1} \mid x = 0, y \geq 0\},$$

$$\hat{S}_1 = \{(x, y) \in \mathbb{R}^{n+1} \mid \mathbf{l}_1 \leq x \leq \mathbf{u}_1,\ g_{1,i}(x) \leq 0,\ i = 1, \ldots, \ell, g_{1,\ell+1}(x) \leq y\}.$$

Third, if the form of the functions $g_{1,i}(\cdot)$ is known in more detail, in some cases one can avoid the need to take the closure and thus the differentiability issue, for example for polynomial functions [GL12]. The remarkable computational advantages of Theorem 3.2 are discussed, for example, in [GL12, FG06].

3.3.2 Single indicator constraint

We now consider a slightly more general case where the set S_0 is not a point but a box. Namely,

$$S_0 = \{x \in \mathbb{R}^n \mid \mathbf{l}_0 \leq x \leq \mathbf{u}_0\},$$

$$S_1 = \{x \in \mathbb{R}^n \mid \mathbf{l}_1 \leq x \leq \mathbf{u}_1, g_1(x) \leq 0\}.$$

We note that the special case of Section 3.3.1 with $\ell = 1$ is obtained by setting $\mathbf{l}_0 = \mathbf{u}_0 = 0$. Again, the above case occurs in several applications, for example, in telecommunication for the so-called delay-constrained routing problem [BAO06].

In order to formulate an extension of Theorem 3.2 (under an additional condition), we need the following definition, that has originally been given in [HBCO12].

Definition 3.3. *A function $g : \mathbb{R}^n \to \mathbb{R}$ is called independently non-decreasing (resp. non-increasing) in the i-th coordinate, if $\forall\ x \in dom(g)$ and $\forall\ \lambda > 0$, we have $g(x + \lambda \mathbf{e}_i) \geq g(x)$ (resp. $g(x + \lambda \mathbf{e}_i) \leq g(x)$). We say that g is independently monotone in the i-th coordinate, if it is either independently non-decreasing or non-increasing. Finally, g is called isotone, if it is independently monotone in every coordinate.*

For any subset $I \subseteq [n]$ we denote by $\bar{I}$ the complement of I in $[n]$, i.e., $\bar{I} := [n] \setminus I$. For an isotone function $g(\cdot)$ we denote by $J_1(g)$ (resp. $J_2(g)$), the set of indices in which $g(\cdot)$ is independently non-decreasing (resp. independently non-increasing). If a function $g(\cdot)$ is constant in coordinate i, it is both independently non-decreasing and non-increasing. In such a case, we assume that the index i is arbitrarily assigned to exactly one of the sets $J_1(g)$ and $J_2(g)$. In this way, we always get a partition of the index set, i.e., for any isotone function g, $J_1(g) \cup J_2(g) = [n]$ and $J_1(g) \cap J_2(g) = \emptyset$.

From now on, we assume the existence of an underlying set of closed convex functions $\{g_j : \mathbb{R}^n \to \mathbb{R} \mid j \in \mathcal{J}\}$ with associated bounds $\mathbf{l}_j, \mathbf{u}_j$ and define for all $I \subseteq [n]$, $x \in \mathbb{R}^n$, $z > 0$ and $j, j' \in \mathcal{J}$ the function $h_I^{j,j'} : \mathbb{R}^{n+1} \mapsto \mathbb{R}^n$, with

$$\left(h_I^{j,j'}\right)_i (x, z) := \begin{cases} \left(\mathbf{l}_j\right)_i, & i \in I \cap J_1(g_j) \\ \left(\mathbf{u}_j\right)_i, & i \in I \cap J_2(g_j) \\ x_i - \frac{(1-z)\left(\mathbf{u}_{j'}\right)_i}{z}, & i \in \bar{I} \cap J_1(g_j) \\ x_i - \frac{(1-z)\left(\mathbf{l}_{j'}\right)_i}{z} & i \in \bar{I} \cap J_2(g_j) \end{cases} \tag{3.14}$$

$\forall\ i \in [n]$, and $q_I^{j,j'} = g^j \circ h_I^{j,j'}$. We can then state the main result of [HBCO12] in our slightly different notation.

Theorem 3.4. *Let $\mathcal{J} = \{0, 1\}$ and for $j \in \mathcal{J}$ define $\Gamma_j := S_j \times \{j\}$, with S_j specified as above and non-empty. If $g_1(\cdot)$ is isotone,* $\mathrm{conv}(\Gamma_0 \cup \Gamma_1) = \mathrm{cl}(\Gamma')$, *where*

$$\Gamma' = \left\{ \begin{array}{ll} (x, z) \in \mathbb{R}^{n+1} & \\ z q_I^{1,0}(\frac{x}{z}, z) \leq 0, & \forall\ I \subseteq [n] \\ z\mathbf{l}_1 + (1-z)\mathbf{l}_0 \leq x \leq z\mathbf{u}_1 + (1-z)\mathbf{u}_0 & \\ 0 < z \leq 1 & \end{array} \right\}.$$

Hijazi et al. [HBCO12] have shown that formulating the delay-constrained routing problem [BAO06] by means of Theorem 3.4 leads to a significant computational advantage over a straightforward big-M formulation.

It is worth discussing the trade-off between applying Theorem 3.4 versus Theorem 1.11. On the one side, the advantage of Theorem 3.4 is the fact that we project back onto the original space of variables. In fact, we can express $\text{conv}(\Gamma_0 \cup \Gamma_1) = \text{cl}(\Gamma')$ as a subset of a $(n+1)$-dimensional space. A direct application of Theorem 1.11 would lead to expressing the convex hull as the projection of a subset of a $(3n+5)$-dimensional space. On the other side, this gain does not come for free. In Theorem 3.4, we need exponentially many constraints, indexed in $I \subseteq [n]$. In particular, including all the simple bound constraints, we have $2^n + 2n + 2$ constraints opposed to $4n + 9$ that would result from a direct application of Theorem 1.11. Similar considerations apply to the results presented in the following sections. In essence, an upper bound on the number of constraints needed to describe the convex hull is always exponential, although only in the number of variables involved in the constraint. In the following, we will see that it is sometimes possible to detect redundant constraints among the exponentially many ones. Moreover, these many constraints could be potentially added in a cutting-plane fashion, for example on top of a big-M formulation. In the next section, we will se that in the case of affine functions, exactly one of the many inequalities is precisely a big-M constraint and thus alone is sufficient in order to get a valid formulation. In Section 3.5, we will also show some concrete examples that illustrate the growth of the number of constraints when applying the results presented in this chapter, depending on the application, and in particular on the number of variables that are involved in the disjunction.

3.3.3 Linear constraints

We now assume that $g_1(\cdot)$ is an affine function. Such functions are easily seen to be isotone, and thus Theorem 3.4 is still valid. However, the constraints needed to describe the convex hull have a form that allows to avoid the need to take the closure. For any affine function $g_j(x) = a_{j0} + \sum_{i=1}^n a_{ji}x_i$, $j' \in \mathcal{J}$ and $I \subseteq [n]$ we define the linear function

$$H_I^{j,j'}(x,z) := \sum_{i\in\bar{I}} a_{ji}x_i + z\left(a_{j0} + \sum_{i\in I, a_{ji}>0} a_{ji}\left(\mathbf{l}_j\right)_i + \sum_{i\in I, a_{ji}<0} a_{ji}\left(\mathbf{u}_j\right)_i\right) - (1-z)\left(\sum_{i\in\bar{I}, a_{ji}>0} a_{ji}\left(\mathbf{u}_{j'}\right)_i + \sum_{i\in\bar{I}, a_{ji}<0} a_{ji}\left(\mathbf{l}_{j'}\right)_i\right). \quad (3.15)$$

It is easy to check that, for example, $H_I^{1,0}(x,z) = zq_I^{1,0}(\frac{x}{z}, z)$. Then, Theorem 3.4 reads as the following, as was noted in [Hij10].

Corollary 3.5. *Consider the situation of Theorem 3.4. We then have that if* $g_1(\cdot)$

is affine, $\operatorname{conv}(\Gamma_0 \cup \Gamma_1) = \Gamma''$*, where*

$$\Gamma'' = \left\{ \begin{array}{ll} (x,z) \in \mathbb{R}^{n+1} & \\ H_I^{1,0}(x,z) \leq 0, & \forall\ I \subseteq [n] \\ z\mathbf{l}_1 + (1-z)\mathbf{l}_0 \leq x \leq z\mathbf{u}_1 + (1-z)\mathbf{u}_0 & \\ 0 \leq z \leq 1 & \end{array} \right\}.$$

We make several observations regarding this corollary. The first and the fourth have also been made in [Hij10].

Observation 3.6. *In this special case of* g_1 *being an affine function, we note that in Corollary 3.5 we can restrict to subsets* $I \subseteq \tilde{N} := \{i \in [n] \mid a_{1i} \neq 0\}$*.*

Observation 3.7. *Consider the case where* S_1 *is described by several linear constraints, for example* $S_1 = \{x \in \mathbb{R}^n \mid Dx \leq b, \mathbf{l}_1 \leq x \leq \mathbf{u}_1\}$ *with* $D \in \mathbb{R}^{m \times n}$*. If the coefficients of* D *are monotonous for every column, i.e., if for all* $j \in [n]$*, and for any pair* $i, i' \in [m]$*,* $d_{ij} \cdot d_{i'j} \geq 0$ *holds, then Corollary 3.5 can be naturally extended.*

Observation 3.8. *Consider the context of Corollary 3.5 and assume further that*

$$S_1 \subseteq S_0. \tag{3.16}$$

This is equivalent to saying $\operatorname{proj}_x \Gamma_1 \subseteq \operatorname{proj}_x \Gamma_0$ *and holds, for example, if* $\mathbf{l}_0 = \mathbf{l}_1$ *and* $\mathbf{u}_0 = \mathbf{u}_1$*, which is the case in many applications. Furthermore, it is easy to see that the non-redundant facets of* Γ'' *among the set of inequalities* $H_I^{1,0}(x,z) \leq 0$ *indexed in* $I \subseteq \tilde{N}$ *are binding on at least one point* $(x',1) \in \Gamma_1$*, i.e.,* $H_I^{1,0}(x',1) = 0$*. Now, if* $\frac{\partial H_I^{1,0}}{\partial z}(x,z)$ *was negative, then we would have* $H_I^{1,0}(x',0) > 0$*, which contradicts the fact that* $\operatorname{proj}_x \Gamma_1 \subseteq \operatorname{proj}_x \Gamma_0$ *and that* $H_I^{1,0}(x,z)$ *is a valid inequality for* Γ_0*. Thus, in order to describe* $\operatorname{conv}(\Gamma_0 \cup \Gamma_1)$ *in Corollary 3.5, under assumption (3.16) we can further restrict to subsets* $I \in \hat{N}$*, where*

$$\hat{N} := \left\{ I \subseteq \tilde{N} \;\middle|\; \frac{\partial H_I^{1,0}}{\partial z}(x,z) \geq 0 \right\}.$$

The derivatives are also easy to compute. Namely,

$$\begin{aligned} \frac{\partial H_I^{1,0}}{\partial z}(x,z) = a_{1,0} \;&+ \sum_{i \in I, a_{1i} > 0} a_{1i} \left(\mathbf{l}_1\right)_i \;+ \sum_{i \in I, a_{1i} < 0} a_{1i} \left(\mathbf{u}_1\right)_i \\ &+ \sum_{i \in \bar{I}, a_{1i} > 0} a_{1i} \left(\mathbf{u}_0\right)_i \;+ \sum_{i \in \bar{I}, a_{1i} < 0} a_{1i} \left(\mathbf{l}_0\right)_i. \end{aligned}$$

Observation 3.9. *If we assume that the bounds* $\mathbf{u}_1$ *and* $\mathbf{l}_1$ *are equal to some globally known bounds* $\overline{x}$ *and* $\underline{x}$ *on the variables* x*, then it is an easy exercise to check that* $H^{1,0}_{\emptyset}(x,z) \leq 0$ *is a big-M constraint with the big-M calculated precisely as in* (3.8)*. This shows that, in the light of what was said at the end of Section 3.3.2, only one of the exponentially many constraints in Corollary 3.5 is enough to get a valid formulation, while the rest could be used in a cutting-plane fashion.*

To the best of our knowledge, there is no computational investigation so far on the quality of the bound achievable by the reformulation based on Corollary 3.5. We will show in Section 3.5.1 some results on it by solving supervised classification problems (cf. Section 1.5.4) with the standard big-M formulation and with the disjunctive one.

3.4 Pairs of related disjunctions

In this section we consider the case in which two disjunctions of the type introduced in Section 3.3.2 are related to each other. Of course, we are still interested in characterizing the cases in which the convex hull description of Theorem 1.11 can be projected onto the space of the original variables. Namely, in Section 3.4.1 we show how to deal with the case in which two disjunctions are *complementary*, i.e., precisely one of their associated indicator variables must take value one. This is a significant generalization of the result in Section 3.3.2, also because it can be interpreted as if both terms of a binary disjunction are actual constraints given as the level sets of a single function. This can be seen as an actual *if-the-else* construct. However, we will see that an additional technical condition is required. Moreover, in Section 3.4.2 we consider the case in which at most one of the indicator variables associated with a pair of related disjunctions can take value one and we examine its relationship with a ternary disjunction.

3.4.1 Complementary indicator constrains

We now consider a pair of disjunctions of the type discussed in Section 3.3.2, which are complementary to each other in the sense that the indicator constraint described by the first disjunctive term, $g_0(x) \leq 0$, is deactivated when the indicator constraint of the second disjunctive term, $g_1(x) \leq 0$, is active and vice versa. To model this situation we consider the sets

$$S_0 = \{x \in \mathbb{R}^n \mid \mathbf{l}_0 \leq x \leq \mathbf{u}_0\},$$

$$S_1 = \{x \in \mathbb{R}^n \mid \mathbf{l}_1 \leq x \leq \mathbf{u}_1, g_1(x) \leq 0\},$$

$$\overline{S}_0 = \{x \in \mathbb{R}^n \mid \mathbf{l}_0 \leq x \leq \mathbf{u}_0, g_0(x) \leq 0\},$$

$$\overline{S}_1 = \{x \in \mathbb{R}^n \mid \mathbf{l}_1 \leq x \leq \mathbf{u}_1\}.$$

We denote by z the indicator variable associated with the first two sets: when it is equal to one, the constraint $g_1(x) \leq 0$ is active. Conversely, the indicator variable $\overline{z}$ is associated with the second pair of sets: when it is equal to one, the constraint $g_0(x) \leq 0$ is active. The complementarity is assured by the fact that the two indicators have to sum up to one. Therefore, we define

$$H^{=} := \{(z, \overline{z}) \in \mathbb{R}^2 \mid z + \overline{z} = 1\} \text{ and } L^{=} := \mathbb{R}^n \times H^{=}.$$

Then, we can prove the following result.

Proposition 3.10. *Let $\mathcal{J} = \{0, 1\}$ and for $j \in \mathcal{J}$ define $\Gamma_j := S_j \times \{j\} \times [0, 1]$, with S_j specified as above and non-empty, and $\overline{\Gamma}_j := \overline{S}_j \times [0, 1] \times \{1 - j\}$, with $\overline{S}_j$ specified as above and non-empty. If $g_0(\cdot)$ and $g_1(\cdot)$ are isotone functions with $J_1(g_0) = J_2(g_1)$, then*

$$\operatorname{conv}\left((\Gamma_0 \cup \Gamma_1) \cap \left(\overline{\Gamma}_0 \cup \overline{\Gamma}_1\right) \cap L^{=}\right) = \operatorname{conv}(\Gamma_0 \cup \Gamma_1) \cap \operatorname{conv}\left(\overline{\Gamma}_0 \cup \overline{\Gamma}_1\right) \cap L^{=} = \operatorname{cl}(\Gamma^*),$$

where

$$\Gamma^* = \left\{ \begin{array}{l} (x, z, \overline{z}) \in \mathbb{R}^{n+2} \\ \left.\begin{array}{l} z q_I^{1,0}(\frac{x}{z}, z) \leq 0 \\ \overline{z} q_I^{0,1}(\frac{x}{\overline{z}}, \overline{z}) \leq 0 \end{array}\right\} \forall\ I \subseteq [n] \\ z\mathbf{l}_1 + (1 - z)\mathbf{l}_0 \leq x \leq z\mathbf{u}_1 + (1 - z)\mathbf{u}_0 \\ 0 < z, \overline{z} < 1 \\ z + \overline{z} = 1 \end{array} \right\}.$$

Proof. We will first show the second equality above. First of all, we note that

$$\Gamma_0 \cup \Gamma_1 = \big((S_0 \times \{0\}) \cup (S_1 \times \{1\})\big) \times [0, 1]$$

and thus

$$\operatorname{conv}(\Gamma_0 \cup \Gamma_1) = \operatorname{conv}\big((S_0 \times \{0\}) \cup (S_1 \times \{1\})\big) \times [0, 1].$$

To the convex hull on the right-hand-side one can apply Theorem 3.4 and then lift it to its cartesian product with $[0, 1]$, and the same applies to $\overline{\Gamma}_0 \cup \overline{\Gamma}_1$. By intersecting the two resulting sets and $L^{=}$, one gets exactly the set $\operatorname{cl}(\Gamma^*)$ after noting that the constraints

$$z\mathbf{l}_1 + (1 - z)\mathbf{l}_0 \leq x \leq z\mathbf{u}_1 + (1 - z)\mathbf{u}_0 \text{ and } \overline{z}\mathbf{l}_0 + (1 - \overline{z})\mathbf{l}_1 \leq x \leq \overline{z}\mathbf{u}_0 + (1 - \overline{z})\mathbf{u}_1$$

are identical after substituting $\overline{z}$ by $1 - z$.

Now we come to the first equality and prove it by showing two set inclusions. First, the left-hand-side set is trivially included in the right-hand-side one. The converse set inclusion requires a bit more work. Note that we can write

$$(\Gamma_0 \cup \Gamma_1) \cap \left(\overline{\Gamma}_0 \cup \overline{\Gamma}_1\right) \cap L^{=} = \left(\overline{S}_0 \times \{0\} \times \{1\}\right) \cup \left(S_1 \times \{1\} \times \{0\}\right).$$

Using this identity and Theorem 1.11, we know $\operatorname{conv}\left((\Gamma_0 \cup \Gamma_1) \cap \left(\overline{\Gamma}_0 \cup \overline{\Gamma}_1\right) \cap L^{=}\right)$ to be the projection onto $(x, z, \overline{z})$ of the closure of the set

$$\left\{\begin{array}{l} (x, x^0, x^1, z, z^0, z^1, \overline{z}, \overline{z}^0, \overline{z}^1, \lambda^0, \lambda^1) \in \mathbb{R}^{3n+8} \\ x = x^0 + x^1,\ z = z^0 + z^1,\ \overline{z} = \overline{z}^0 + \overline{z}^1 \\ \lambda^0 + \lambda^1 = 1,\ \tilde{g}_0(x^0, \lambda^0) \leq 0,\ \tilde{g}_1(x^1, \lambda^1) \leq 0 \\ \lambda^0 \mathbf{l}_0 \leq x^0 \leq \lambda^0 \mathbf{u}_0,\ \lambda^1 \mathbf{l}_1 \leq x^1 \leq \lambda^1 \mathbf{u}_1 \\ z^0 = 0,\ z^1 = \lambda^1,\ \overline{z}^0 = \lambda^0,\ \overline{z}^1 = 0,\ \lambda^0, \lambda^1 > 0 \end{array}\right\}.$$

By eliminating $z^0 = 0$ and $\overline{z}^1 = 0$, identifying $z = z^1 = \lambda^1$ and $\overline{z} = \overline{z}^0 = \lambda^0$, and substituting $x^0 = x - x^1$ and x^1 by y, this set becomes

$$\hat{\Gamma} = \left\{\begin{array}{l} (x, y, z, \overline{z}) \in \mathbb{R}^{2n+2} \\ z g_1(y/z) \leq 0 \\ \overline{z} g_0\left((x - y)/\overline{z}\right) \leq 0 \\ x - \overline{z}\mathbf{u}_0 \leq y \leq x - \overline{z}\mathbf{l}_0 \\ z\mathbf{l}_1 \leq y \leq z\mathbf{u}_1 \\ 0 < z, \overline{z} < 1 \end{array}\right\}.$$

It remains to show that $\operatorname{cl}(\Gamma^*) \subseteq \operatorname{proj}_{(x,z,\overline{z})} cl(\hat{\Gamma})$. Therefore, let $(x, z, \overline{z}) \in \Gamma^*$ and define $y \in \mathbb{R}^n$ as

$$\begin{aligned} y_i &= \max\{z\,(\mathbf{l}_1)_i\,, x_i - \overline{z}\,(\mathbf{u}_0)_i\} \quad \forall\ i \in J_1(g_1), \\ y_i &= \min\{z\,(\mathbf{u}_1)_i\,, x_i - \overline{z}\,(\mathbf{l}_0)_i\} \quad \forall\ i \in J_2(g_1). \end{aligned}$$

Then, one can check that there is a set $I \subseteq [n]$ such that $h_I^{1,0}(\frac{x}{z}, z) = y/z$. Furthermore, taking into account that $J_2(g_1) = J_1(g_0)$, we can also check that $h_{\overline{I}}^{0,1}(\frac{x}{\overline{z}}, \overline{z}) = \frac{x-y}{\overline{z}}$. Because $z q_I^{1,0}(\frac{x}{z}, z) = z g_1(\frac{y}{z}) \leq 0$ and $\overline{z} q_{\overline{I}}^{0,1}(\frac{x}{\overline{z}}, \overline{z}) = \overline{z} g_0\left(\frac{x-y}{\overline{z}}\right) \leq 0$, we deduce that $g_1(\frac{y}{z}) \leq 0$ and $g_0\left(\frac{x-y}{\overline{z}}\right) \leq 0$. The lower and upper bounds on y are easily checked to hold and we have that $(x, y, z, \overline{z}) \in \hat{\Gamma}$.

We now come to the closure. Consider any point $(x, z, \overline{z}) \in \operatorname{cl}(\Gamma^*)$ and let $(x_k, z_k, \overline{z}_k)$

be a sequence of points in Γ^* such that $\lim_{k\to\infty}(x_k, z_k, \overline{z}_k) = (x, z, \overline{z})$. For every $k \in \mathbb{N}$ one can define y_k as above to get a point $(x_k, y_k, z_k, \overline{z}_k) \in \hat{\Gamma}$, and y_k converges to some $y \in \mathbb{R}^n$. Thus, $\lim_{k\to\infty}(x_k, y_k, z_k, \overline{z}) = (x, y, z, \overline{z}) \in \mathrm{cl}(\hat{\Gamma})$. With this, the proof is complete. □

The proof of Proposition 3.10 is a straightforward extension of the proof of Theorem 3.4 given in [HBCO12]. We note that it can now also be recovered from the results in the very recent work [Vie15a]. The interpretation of Proposition is somewhat surprising. Under the technical condition that the functions $g_0(\cdot)$ and $g_1(\cdot)$ be isotone, and that in addition $J_2(g_1) = J_1(g_0)$, it shows that computing the convex hull of the intersection of the two disjunctions does not allow any improvement with respect to intersecting the individual convex hulls of the two single disjunctions considered in isolation. Without this technical condition, this may not be true (see Example 3.14). Of course, Proposition 3.10 also shows that everything can be done in the original space of variables.

Because the two disjunctions above are complementary, i.e., $\overline{z} = 1 - z$, the whole situation could be described as a single disjunction with a single indicator variable, that is, as a subset of $\mathbb{R}^{n+1}$. For example,

$$(x, z) \in \left(\overline{S}_0 \times \{0\}\right) \cup \left(S_1 \times \{1\}\right).$$

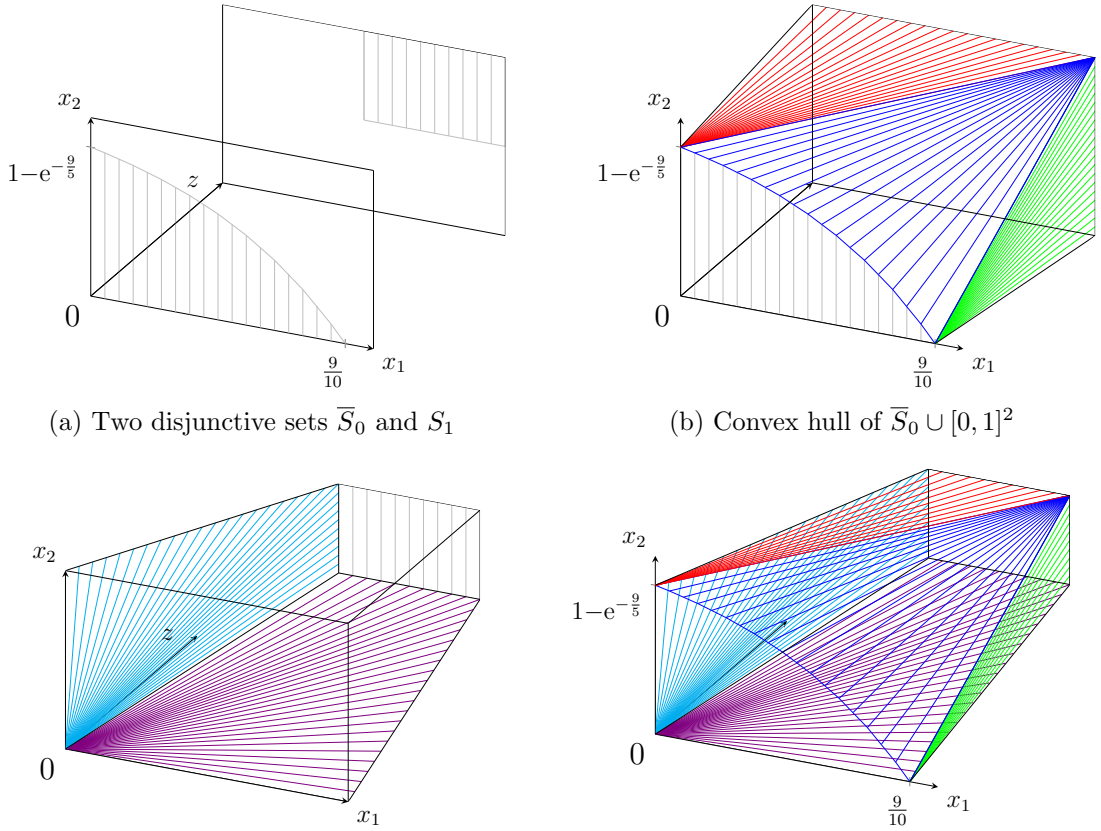
Its convex hull is then just the projection of the closure of Γ^* from Proposition 3.10 onto the first $n + 1$ variables. The fact that the whole situation can be managed by projecting out $\overline{z}$ was already somewhat anticipated in the proof of Proposition 3.10. We formalize this observation in the following corollary.

Corollary 3.11. *Let* $\mathring{\Gamma}_0 := \overline{S}_0 \times \{0\}$ *and* $\mathring{\Gamma}_1 := S_1 \times \{1\}$*, with* $\overline{S}_0$ *and* S_1 *specified as above and non-empty. Thus, if* $J_2(g_1) = J_1(g_0)$*, then* $\mathrm{conv}(\mathring{\Gamma}_0 \cup \mathring{\Gamma}_1) = \mathrm{cl}(\mathring{\Gamma})$*, where*

$$\mathring{\Gamma} = \left\{ \begin{array}{l} (x, z) \in \mathbb{R}^{n+1} \\ \left.\begin{array}{l} z q_I^{1,0}(\frac{x}{z}, z) \le 0 \\ (1-z) q_I^{0,1}(\frac{x}{1-z}, 1-z) \le 0 \end{array}\right\} \forall\ I \subseteq [n] \\ z\mathbf{l}_1 + (1-z)\mathbf{l}_0 \le x \le z\mathbf{u}_1 + (1-z)\mathbf{u}_0 \\ 0 < z < 1 \end{array} \right\}.$$

The condition about the index sets of the isotone functions required for Proposition 3.10 and Corollary 3.11 is interpreted in the following observation.

Observation 3.12. *If* $J_2(g_1) = J_1(g_0)$*, then there is a vertex* v *of the rectangle* $[\mathbf{l}_0, \mathbf{u}_0]$ *such that* $g_0(v) \le 0$*. In fact, because* $\overline{S}_0 \neq \emptyset$*, there is a* $x^0 \in [\mathbf{l}_0, \mathbf{u}_0]$ *with*



(a) Two disjunctive sets $\overline{S}_0$ and S_1

(b) Convex hull of $\overline{S}_0 \cup [0,1]^2$

(c) Convex hull of $[0,1]^2 \cup S_1$

(d) Intersection of the two convex hulls

Figure 3.1: Illustrative construction of the convex hull of two disjunctive sets

$g_0(x^0) \leq 0$. *If we set*

$$v_i := \begin{cases} (\mathbf{l}_0)_i, & i \in J_1(g_0) \\ (\mathbf{u}_0)_i, & i \in J_2(g_0) \end{cases},$$

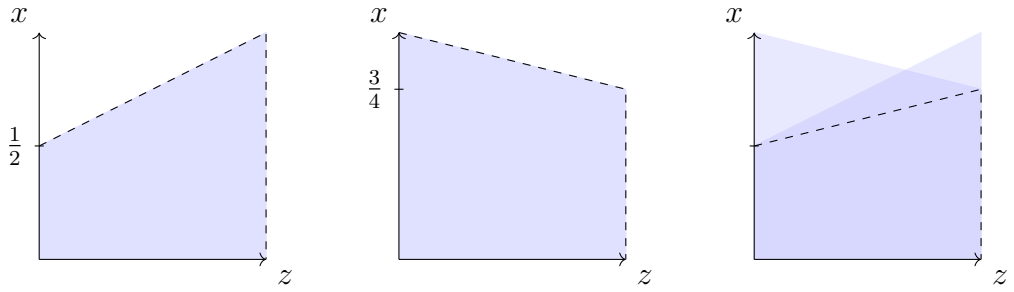
then $g_0(v) \leq g_0(x^0) \leq 0$. *Furthermore, because also* $S_1 \neq \emptyset$, *there is a* $x^1 \in [\mathbf{l}_1, \mathbf{u}_1]$ *with* $g_1(x^1) \leq 0$. *If we then define* $w \in [\mathbf{l}_1, \mathbf{u}_1]$ *as the opposing vertex, i.e.,*

$$w_i := \begin{cases} (\mathbf{u}_1)_i, & v_i = (\mathbf{l}_0)_i \\ (\mathbf{l}_1)_i, & v_i = (\mathbf{u}_0)_i \end{cases},$$

due to $J_2(g_1) = J_1(g_0)$, *we get that* $g_1(w) \leq g_1(x^1) \leq 0$. *Thus, the resulting convex hull contains at least two such opposing vertices of the underlying hyperractangles.*

We now give an example of Corollary 3.11 involving functions in two variables, giving rise to three-dimanesional sets.

Example 3.13. *Let* $\mathbf{l}_0 = \mathbf{l}_1 = 0 \in \mathbb{R}^2$ *and* $\mathbf{u}_0 = \mathbf{u}_1 = 1 \in \mathbb{R}^2$, *and consider the functions* $g_0(x_1, x_2) = \exp\left(2x_1 - \frac{9}{5}\right) + x_2 - 1$ *and* $g_1(x_1, x_2) = \max\left\{\frac{1}{2} - x_1, \frac{1}{2} - x_2\right\}$.



(a) (b) (c)

Figure 3.2: Counterexample for the case in which $J_2(g_1) \neq J_1(g_0)$

One can show that they are both isotone and that $J_1(g_0) = J_2(g_1) = \{1, 2\}$ *and* $J_2(g_0) = J_1(g_1) = \emptyset$. *The two disjunctive sets*

$$\{x \in \mathbb{R}^2 \mid \mathbf{l}_0 \leq x \leq \mathbf{u}_0, g_0(x) \leq 0\} \times \{0\} \text{ and } \{x \in \mathbb{R}^2 \mid \mathbf{l}_1 \leq x \leq \mathbf{u}_1, g_1(x) \leq 0\} \times \{1\}$$

are shown in Figure 3.1 (a). Figures 3.1 (b) and (c), respectively show the non-redundant boundaries of the convex hull of

$$\{x \in \mathbb{R}^2 \mid \mathbf{l}_0 \leq x \leq \mathbf{u}_0, g_0(x) \leq 0\} \times \{0\} \bigcup \{x \in \mathbb{R}^2 \mid \mathbf{l}_1 \leq x \leq \mathbf{u}_1\} \times \{1\}$$

on the one hand, and those of the convex hull of

$$\{x \in \mathbb{R}^2 \mid \mathbf{l}_1 \leq x \leq \mathbf{u}_1, g_1(x) \leq 0\} \times \{1\} \bigcup \{x \in \mathbb{R}^2 \mid \mathbf{l}_0 \leq x \leq \mathbf{u}_0\} \times \{0\}$$

on the other. The convex hulls in Figures 3.1 (b) and (c) are obtained by Theorem 3.4 and Corollary 3.5, respectively. Figure 3.1 (d) then shows the intersection of the latter two, giving the convex hull of the set

$$\{x \in \mathbb{R}^2 \mid \mathbf{l}_0 \leq x \leq \mathbf{u}_0, g_0(x) \leq 0\} \times \{0\} \bigcup \{x \in \mathbb{R}^2 \mid \mathbf{l}_1 \leq x \leq \mathbf{u}_1, g_1(x) \leq 0\} \times \{1\}.$$

Now, we switch from $\mathbb{R}^2$ to $\mathbb{R}$ in order to see what happens when the condition $J_2(g_1) = J_1(g_0)$ is not fulfilled.

Example 3.14. *Let* $l^0 = l^1 = 0$ *and* $u^0 = u^1 = 1$, *and consider the two functions* $H^0(x) = x - \frac{1}{2}$ *and* $H^1(x) = x - \frac{3}{4}$, *both of which are non-decreasing in their first and only coordinate. Figures 3.2 (a) and (b) show the convex hull when considering one indicator constraint at a time. In Figure 3.2 (c) we see that their intersection does not give the convex hull of the two complementary indicator constraints.*

Informally, the condition $J_2(g_1) = J_1(g_0)$ can be described as the fact that one function is non-increasing in those coordinates where the other one is non-decreasing

and vice versa. This holds true in a number of applications, starting from the well-known (and already mentioned) Job Shop Scheduling problem, where for each pair of jobs i, j and each machine k, either i is executed on k before j or vice versa.

In Section 3.5.2 we will compare the quality of the bound provided by the disjunctive reformulation obtained by applying Corollary 3.11 above and the straightforward big-M formulation for JSS.

3.4.2 Almost complementary indicator constraints

In this section we will start again from the setting of Section 3.4.1, but weaken the requirement on the relation between the two disjunctions. In particular, we relax the constraint $z + \bar{z} = 1$ by imposing $z + \bar{z} \leq 1$ instead. This means that in addition to the two possibilities of activating one constraint at a time, we also allow that none of them is active. By slightly modifying the notation from before, we will consider the three sets

$$S_0 = \{x \in \mathbb{R}^n \mid \mathbf{l}_0 \leq x \leq \mathbf{u}_0\},$$

$$S_1 = \{x \in \mathbb{R}^n \mid \mathbf{l}_1 \leq x \leq \mathbf{u}_1, g_1(x) \leq 0\},$$

$$S_{-1} = \{x \in \mathbb{R}^n \mid \mathbf{l}_{-1} \leq x \leq \mathbf{u}_{-1}, g_{-1}(x) \leq 0\}.$$

Again, we denote by z the indicator variable that activates the constraint $g_1(x) \leq 0$, and by $\bar{z}$ the one that activates the constraint $g_{-1}(x) \leq 0$. Moreover, we extend (3.16) as

$$S_1 \subseteq S_0 \text{ and } S_{-1} \subseteq S_0. \tag{3.17}$$

If we define

$$H^{\leq} := \{(z, \bar{z}) \in \mathbb{R}^2 \mid z + \bar{z} \leq 1\} \text{ and } L^{\leq} := \mathbb{R}^n \times H^{\leq},$$

and for $j \in \mathcal{J} = \{0, 1\}$, by slight abuse of notation,

$$\Gamma_j := S_j \times \{j\} \times [0, 1] \text{ and } \bar{\Gamma}_j := S_{-j} \times [0, 1] \times \{j\},$$

then the situation can be modeled as the intersection of two disjunctions and $L^{\leq}$, namely

$$(x, z, \bar{z}) \in (\Gamma_0 \cup \Gamma_1) \cap \left(\bar{\Gamma}_0 \cup \bar{\Gamma}_1\right) \cap L^{\leq}. \tag{3.18}$$

Observation 3.15. *It is easy to check that* (3.18) *is equivalent to the ternary disjunction*

$$(x, z, \bar{z}) \in (S_0 \times \{0\} \times \{0\}) \cup (S_1 \times \{1\} \times \{0\}) \cup (S_{-1} \times \{0\} \times \{1\}).$$

From an application perspective, such a ternary case holds, for example, in the context of the TSPTW (cf. Section 1.5.1). There, for each pair of cities i and j, either city j is visited immediately after city i (say, S_1), or immediately before (say, S_{-1}), or the visit happens not adjacent to i (say, S_0).

In the case of (3.18), we do not know a description of the convex hull in the original space of variables, but using Theorem 3.4, we can compute a superset.

Corollary 3.16. *Let S_0, S_1 and S_{-1} be specified as above and non-empty. Thus, we have that if g_{-1} and g_1 are isotone functions, then*

$$\operatorname{conv}\left((\Gamma_0 \cup \Gamma_1) \cap \left(\bar{\Gamma}_0 \cup \bar{\Gamma}_1\right) \cap L^{\leq}\right) \subseteq \operatorname{conv}(\Gamma_0 \cup \Gamma_1) \cap \operatorname{conv}\left(\bar{\Gamma}_0 \cup \bar{\Gamma}_1\right) \cap L^{\leq} = \operatorname{cl}(\breve{\Gamma}),$$

where

$$\breve{\Gamma} = \left\{ \begin{array}{l} (x, z, \bar{z}) \in \mathbb{R}^{n+2}, \\ \left.\begin{array}{l} z q_I^{1,0}(\frac{x}{z}, z) \leq 0 \\ \bar{z} q_I^{-1,0}(\frac{x}{\bar{z}}, \bar{z}) \leq 0 \end{array}\right\} \forall\ I \subset [n], \\ z\mathbf{l}_1 + (1-z)\mathbf{l}_0 \leq x \leq z\mathbf{u}_1 + (1-z)\mathbf{u}_0 \\ \bar{z}\mathbf{l}_{-1} + (1-\bar{z})\mathbf{l}_0 \leq x \leq \bar{z}\mathbf{u}_{-1} + (1-\bar{z})\mathbf{u}_0 \\ 0 < z, \bar{z} < 1 \\ z + \bar{z} \leq 1 \end{array} \right\}.$$

Proof. The set inclusion is a simple observation, while the set equality follows by Theorem 3.4. □

In Section 3.5.3 we will discuss some computational results comparing the straightforward big-M formulation of the TSPTW with the disjunctive one based on Corollary 3.16 above.

3.4.2.1 Ternary disjunctions

Of course, a ternary case like the one described in the previous section can be managed by means of a unique indicator variable, say $\tilde{z}$, which will lead to (3.9) - (3.13) with $\mathcal{J} = \{-1, 0, 1\}$. It is natural to ask, on the one side, if it is possible to write the counterpart of Corollary 3.16 for this special case, and, on the other hand, which is the relationship between these two ways of representing essentially the same disjunction. The answer to the first question is positive under the technical

condition of the functions $g_{-1}(\cdot)$ and $g_1(\cdot)$ being affine (discussed later), while that to the second one is once again a bit surprising: we will show in the following that working with two distinct indicators (as in Corollary 3.16) is stronger than with one indicator variable only.

To deal with the ternary case with a unique indicator variable we need to extend the definition of the set $\hat{N}$ of Observation 3.8 to pairs of indices $j, j' \in \mathcal{J} = \{-1, 0, 1\}$:

$$\hat{N}^{j,j'} := \Big\{ I \subseteq \tilde{N} \;\Big|\; a_{j,0} + \sum_{i \in I, a_{j,i} > 0} a_{j,i} \left(\mathbf{l}_j\right)_i + \sum_{i \in I, a_{j,i} < 0} a_{j,i} \left(\mathbf{u}_j\right)_i + \sum_{i \in \bar{I}, a_{j,i} > 0} a_{j,i} \left(\mathbf{u}_{j'}\right)_i + \sum_{i \in \bar{I}, a_{j,i} < 0} a_{j,i} \left(\mathbf{l}_{j'}\right)_i \geq 0. \Big\}$$

Then, we can prove the following result.

Proposition 3.17. *Let $\mathcal{J} = \{-1, 0, 1\}$ and for $j \in \mathcal{J}$ define $\Gamma_j := S_j \times \{j\}$, with S_j specified as above and non-empty. Moreover, let us assume that (3.17) holds and that $g_{-1}(\cdot)$ and $g_1(\cdot)$ are affine functions. Then,* $\mathrm{conv}(\Gamma_0 \cup \Gamma_1 \cup \Gamma_{-1}) = \Gamma'''$*, where*

$$\Gamma''' = \left\{ \begin{array}{ll} (x, \tilde{z}) \in \mathbb{R}^{n+1} & \\ H_I^{1,0}(x, \tilde{z}) \leq 0, & \forall\ I \in \hat{N}^{1,0} \\ H_I^{-1,0}(x, -\tilde{z}) \leq 0, & \forall\ I \in \hat{N}^{-1,0} \\ \tilde{z}\mathbf{l}_1 + (1 - \tilde{z})\mathbf{l}_0 \leq x \leq \tilde{z}\mathbf{u}_1 + (1 - \tilde{z})\mathbf{u}_0 & \\ -\tilde{z}\mathbf{l}_{-1} + (1 + \tilde{z})\mathbf{l}_0 \leq x \leq -\tilde{z}\mathbf{u}_{-1} + (1 + \tilde{z})\mathbf{u}_0 & \\ -1 \leq \tilde{z} \leq 1 & \end{array} \right\}.$$

Proof. We start by defining the two relaxed sets

$$\tilde{\Gamma}_1 = \left\{ \begin{array}{ll} (x, \tilde{z}) \in \mathbb{R}^{n+1} & \\ H_I^{1,0}(x, \tilde{z}) \leq 0, & \forall\ I \in \hat{N}^{1,0} \\ \tilde{z}\mathbf{l}_1 + (1 - \tilde{z})\mathbf{l}_0 \leq x \leq \tilde{z}\mathbf{u}_1 + (1 - \tilde{z})\mathbf{u}_0 & \\ -1 \leq \tilde{z} \leq 1 & \end{array} \right\}$$

and

$$\tilde{\Gamma}_{-1} = \left\{ \begin{array}{ll} (x, \tilde{z}) \in \mathbb{R}^{n+1} & \\ H_I^{-1,0}(x, -\tilde{z}) \leq 0, & \forall\ I \in \hat{N}^{-1,0} \\ -\tilde{z}\mathbf{l}_{-1} + (1 + \tilde{z})\mathbf{l}_0 \leq x \leq -\tilde{z}\mathbf{u}_{-1} + (1 + \tilde{z})\mathbf{u}_0 & \\ -1 \leq \tilde{z} \leq 1 & \end{array} \right\},$$

and note that $\Gamma''' = \tilde{\Gamma}_1 \cap \tilde{\Gamma}_{-1}$. By Corollary 3.5 and Observation 3.8, because (3.16) holds for $(1,0)$ and $(-1,0)$, we have $\tilde{\Gamma}_1 \cap (\mathbb{R}^n \times [0,1]) = \text{conv}(\Gamma_0 \cup \Gamma_1)$ and $\tilde{\Gamma}_{-1} \cap (\mathbb{R}^n \times [-1,0]) = \text{conv}(\Gamma_0 \cup \Gamma_{-1})$. Thus, $(\Gamma_0 \cup \Gamma_1) \subseteq \tilde{\Gamma}_1$ and $(\Gamma_0 \cup \Gamma_{-1}) \subseteq \tilde{\Gamma}_{-1}$. By the first of these two set inclusions, for any $I \in \hat{N}^{1,0}$, $H_I^{1,0}(x,\tilde{z}) \leq 0$ is a valid inequality for Γ_0. That is, $H_I^{1,0}(v,0) \leq 0$ for any $v \in \text{proj}_x \Gamma_0$. Since $\text{proj}_x \Gamma_{-1} \subseteq \text{proj}_x \Gamma_0$ and $\frac{\partial H_I^{1,0}}{\partial \tilde{z}}(x,\tilde{z}) \geq 0$, we deduce that $H_I^{1,0}(w,-1) \leq 0$ for all $w \in \text{proj}_x \Gamma_{-1}$, i.e., that $H_I^{1,0}(x,\tilde{z}) \leq 0$ is valid for Γ_{-1}. In a similar way, because (3.16) holds for $(1,0)$, one can show that the bound inequalities in $\tilde{\Gamma}_1$ are valid for Γ_{-1}. Hence, we deduce that $(\Gamma_0 \cup \Gamma_1 \cup \Gamma_{-1}) \subseteq \tilde{\Gamma}_1$.

In an analogue fashion, noting that $\frac{\partial H_I^{-1,0}}{\partial(-\tilde{z})}(x,-\tilde{z}) \geq 0$ for all $I \in \hat{N}^{-1,0}$ and that (3.16) holds for $(-1,0)$, one gets $(\Gamma_0 \cup \Gamma_1 \cup \Gamma_{-1}) \subseteq \tilde{\Gamma}_{-1}$. All in all follows that $(\Gamma_0 \cup \Gamma_1 \cup \Gamma_{-1}) \subseteq \left(\tilde{\Gamma}^1 \cap \tilde{\Gamma}_{-1}\right) = \Gamma'''$, and because Γ''' is clearly a convex set, $\text{conv}\,(\Gamma_0 \cup \Gamma^1 \cup \Gamma_{-1}) \subseteq \Gamma'''$. In order to see the converse set inclusion, let $(x,\tilde{z}) \in \tilde{\Gamma}_1 \cap \tilde{\Gamma}_{-1}$. If $\tilde{z} \in [0,1]$, by Corollary 3.5 we get $(x,\tilde{z}) \in \text{conv}\,(\Gamma_0 \cup \Gamma_1)$. Otherwise, $(x,\tilde{z}) \in \text{conv}\,(\Gamma_0 \cup \Gamma_{-1})$. In either case, $(x,\tilde{z}) \in \text{conv}\,(\Gamma_0 \cup \Gamma_1 \cup \Gamma_{-1})$ and we conclude that $\text{conv}\,(\Gamma_0 \cup \Gamma_1 \cup \Gamma_{-1}) = \Gamma'''$. □

Proposition 3.17 is formulated for affine functions because for general isotone functions it would require a definition of the perspective functions for negative values of the multiplier λ (namely, the indicator $\tilde{z}$), different from the definition given in (1.31). To the best of our knowledge, such a definition has not been proposed yet. This is due to the fact that the perspective function with a negative multiplier looses the convexity-preserving property.

In order to compare the two ways of modeling the ternary disjunction proposed in the present and in the previous section, we maintain the more restrictive assumption of affine functions required in Proposition 3.17, under which Corollary 3.16 is, of course, still valid. Then, we define

$$H := \{(z,\bar{z},\tilde{z}) \in \mathbb{R}^3 \mid \tilde{z} = z - \bar{z}\} \text{ and } L := \mathbb{R}^n \times H,$$

$$\Gamma_0 := S_0 \times \{0\} \times \{0\} \times [-1,1],$$

$$\Gamma_1 := S_1 \times \{1\} \times \{0\} \times [-1,1],$$

$$\Gamma_{-1} := S_{-1} \times \{0\} \times \{1\} \times [-1,1],$$

$$\tilde{\Gamma}_0 := S_0 \times [0,1] \times [0,1] \times \{0\},$$

$$\tilde{\Gamma}_1 := S_1 \times [0,1] \times [0,1] \times \{1\},$$

$$\tilde{\Gamma}_{-1} := S_{-1} \times [0,1] \times [0,1] \times \{-1\}.$$

and consider the two disjunctions

$$(x, z, \bar{z}, \tilde{z}) \in (\Gamma_0 \cup \Gamma_1 \cup \Gamma_{-1}) \cap L \text{ and } (x, z, \bar{z}, \tilde{z}) \in \left(\tilde{\Gamma}_0 \cup \tilde{\Gamma}_1 \cup \tilde{\Gamma}_{-1}\right) \cap L. \quad (3.19)$$

These two are equivalent in terms of indicator constraints, in the sense that either $z, \bar{z}$ (together) or $\tilde{z}$ alone indicate the activity of the involved constraints, and with the trivial transformation $\tilde{z} = z - \bar{z}$ one can switch from one case to the other without losing information. However, in terms of their continuous relaxations, the two disjunctions are not the same. By Corollary 3.16 and Proposition 3.17, we can (after projecting out either $\tilde{z}$ or $z, \bar{z}$, respectively) compute a superset of the convex hull of the first disjunction in (3.19) and the convex hull itself of the second disjunction, and then lift them back into $\mathbb{R}^{n+3}$ and intersect with L. The relation of the two resulting sets is characterized by the following corollary.

Corollary 3.18. $\operatorname{conv}(\Gamma_0 \cup \Gamma_1 \cup \Gamma_{-1}) \cap L \subseteq \dot{\Gamma}$ *and* $\operatorname{conv}(\tilde{\Gamma}_0 \cup \tilde{\Gamma}_1 \cup \tilde{\Gamma}_{-1}) \cap L = \check{\Gamma}$, *where*

$$\dot{\Gamma} = \left\{ \begin{array}{ll} (x, z, \bar{z}, \tilde{z}) \in \mathbb{R}^{n+3} & \\ H_I^{1,0}(x, z) \leq 0, & \forall\ I \in \hat{N}^{1,0} \\ H_I^{-1,0}(x, \bar{z}) \leq 0, & \forall\ I \in \hat{N}^{-1,0} \\ z\mathbf{l}_1 + (1 - z)\mathbf{l}_0 \leq x \leq z\mathbf{u}_1 + (1 - z)\mathbf{u}_0 & \\ \bar{z}\mathbf{l}_{-1} + (1 - \bar{z})\mathbf{l}_0 \leq x \leq \bar{z}\mathbf{u}_{-1} + (1 - \bar{z})\mathbf{u}_0 & \\ 0 < z, \bar{z} < 1 & \\ -1 < \tilde{z} < 1 & \\ z + \bar{z} \leq 1 & \\ \tilde{z} = z - \bar{z} & \end{array} \right\}$$

and

$$\check{\Gamma} = \left\{ \begin{array}{ll} (x, z, \bar{z}, \tilde{z}) \in \mathbb{R}^{n+3} & \\ H_I^{1,0}(x, \tilde{z}) \leq 0, & \forall\ I \in \hat{N}^{1,0} \\ H_I^{-1,0}(x, -\tilde{z}) \leq 0, & \forall\ I \in \hat{N}^{-1,0} \\ \tilde{z}\mathbf{l}_1 + (1 - \tilde{z})\mathbf{l}_0 \leq x \leq \tilde{z}\mathbf{u}_1 + (1 - \tilde{z})\mathbf{u}_0 & \\ -\tilde{z}\mathbf{l}_{-1} + (1 + \tilde{z})\mathbf{l}_0 \leq x \leq -\tilde{z}\mathbf{u}_{-1} + (1 + \tilde{z})\mathbf{u}_0 & \\ 0 < z, \bar{z} < 1 & \\ -1 < \tilde{z} < 1 & \\ \tilde{z} = z - \bar{z} & \end{array} \right\}.$$

In addition, $\dot{\Gamma} \subseteq \check{\Gamma}$. *Thus,* $\operatorname{conv}(\Gamma_0 \cup \Gamma_1 \cup \Gamma_{-1}) \cap L \subseteq \operatorname{conv}(\tilde{\Gamma}_0 \cup \tilde{\Gamma}_1 \cup \tilde{\Gamma}_{-1}) \cap L$.

Proof. The set $\dot{\Gamma}$ follows immediately from $\check{\Gamma}$ in the statement of Corollary 3.16 and

Observation 3.8. The inclusion $\text{conv}(\Gamma_0 \cup \Gamma_1 \cup \Gamma_{-1}) \cap L \subseteq \dot{\Gamma}$ follows from the Corollary. Similarly, the set equality $\text{conv}(\tilde{\Gamma}_0 \cup \tilde{\Gamma}_1 \cup \tilde{\Gamma}_{-1}) \cap L = \check{\Gamma}$ follows from Proposition 3.17. The set inclusion $\dot{\Gamma} \subseteq \check{\Gamma}$ can be seen by the fact that the two sets are described by the same constraints apart from $z + \bar{z} \leq 1$ and the constraints indexed in $I \in \hat{N}^{1,0}$ and $I \in \hat{N}^{-1,0}$, respectively. Even if one added the former constraint, $z + \bar{z} \leq 1$, to the definition of $\check{\Gamma}$, note that the latter set of constraints is more restricting in $\dot{\Gamma}$, because, for example, $\frac{\partial H_I^{1,0}}{\partial z} \geq 0$, but $z \geq \tilde{z}$ for all $(x, z, \bar{z}, \tilde{z}) \in \dot{\Gamma}$. Similarly, $\frac{\partial H_I^{-1,0}}{\partial -z} \geq 0$, but $\bar{z} \geq -\tilde{z}$ for any $(x, z, \bar{z}, \tilde{z}) \in \dot{\Gamma}$. □

The computation in Section 3.5.3 shows that the set inclusion $\dot{\Gamma} \subseteq \check{\Gamma}$ can be strict, i.e., $\dot{\Gamma} \subset \check{\Gamma}$.

3.5 Computation

In this section we computationally compare the quality of straightforward big-M formulations and the disjunctive ones implemented in the original space of variables by making use of the results in the previous sections on several problems that expose logical implications in their definition. We express this quality in terms of the dual gap (as a percentage) between the continuous relaxation and the original problem, cf. Definition 1.2. All problems that are solved in order to obtain the values that are needed to calculate the dual gaps in the following sections are MILPs, LPs, MIQPs or QPs, and we therefore use CPLEX 12.6. In particular, the feasible region in all problems is a polyhedral set.

In addition to the dual gap, we compute and report the percentage gap of the closure of rank-1 lift-and-project cuts [Bon12] of the two distinct formulations. We will study this kind of MILP cutting planes in more detail in Chapter 4, and thus postpone a formal definition[3]. We can think of the percentage gap of this closure as an additional measure of the quality of an MILP formulation. In all problems we consider in the following, the binary variables we apply this closure to are indicator variables and it is thus interesting to analyze the impact of the closure on the stronger formulation obtained by Disjunctive Programming with respect to the big-M one. This is the reason we include it in our comparison. In [Bon12] is also described a way to approximate the closure of strengthened rank-1 lift-and-project cuts, and again we include the respective percentage gap in the comparison.

We do not compare the time spent to solve the continuous relaxation of the disjunctive formulation with that of the big-M one, because we are mostly concerned with assessing their strength in terms of dual gaps for now. Note that in the case of linear functions, as mentioned earlier, there are no differentiability issues, i.e.,

[3] In particular, the closures were computed with a version of Algorithm 4.2.

taking the closures in any of the results in the previous sections just amounts to taking $0 \leq z \leq 1$, which can be implemented in a straightforward way. This is the case for the constraint sets in all following cases.

We also note that in the case of linear indicator constraints, exactly one of the inequalities given by the disjunctive formulation coincides with the big-M constraint with the value of M chosen as tight as possible (see Observation 3.9). In this way, the disjunctive formulation can be seen as the big-M formulation augmented by a number of valid inequalities. In any case, we always need bounds on the variables to be able to write either formulation, and we will indicate in each case how we obtain them.

Throughout Tables 3.1 - 3.4, the continuous relaxation of the big-M formulation is denoted by big-M lp, while the one of the continuous relaxation of the perspective reformulation is denoted by CH lp. Similarly, the rank-1 lift-and-project closures are denoted by P_e big-M and P_e CH, and the strengthened closures by P_e^* big-M and P_e^* CH.

3.5.1 Single indicator constraints: Supervised classification

We first consider the supervised classification problem introduced in Section 1.5.4. Therein, we gave an MIQP formulation that was already a big-M reformulation of the problem. We restate the problem here as an MIQP with indicator constraints,

$$\begin{aligned}
\min \quad & \frac{\omega^T\omega}{2} + \frac{C}{m}\left(\sum_{i=1}^{m}\xi_i + 2\sum_{i=1}^{m}(1-z_i)\right) & \\
\text{s.t.} \quad & [z_i = 1] \implies [y_i(\omega^T x_i + \omega_0) - 1 + \xi_i \geq 0] & \forall\, i \in [m] \\
& 0 \leq \xi_i \leq 2 & \forall\, i \in [m] \\
& (\omega, \omega_0) \in \mathbb{R}^{d+1} & \\
& z \in \{0,1\}^m. &
\end{aligned}$$

The continuous variables $(\omega, \omega_0) \in \mathbb{R}^{d+1}$ are unbounded and, in order to be able to reformulate, we have to apply some preprocessing. This was mentioned before; we have to obtain bounds on the variables involved in the disjunctions. For feasible sets described by linear constraints, ways to do so have been proposed. We choose to follow the approach described in [BBF+14], that in addition can be applied with several levels of intensity. In our computational tests, we compare two of these intensities, say *weak* and *strong*. We thus have two different sets of bounds on the variables (ω, ω_0), one stronger and one weaker one. In any case, this is then precisely the case of a linear single indicator constraint case discussed in Section 3.3.3, because for each object i either a linear constraint is active (say, S_1) or the

variables are restricted into a box (say, S_0).

We note that if each of the indicator constraints above is rewritten as a big-M constraint, this results in a set of m constraints. This is opposed to a set of $m \cdot 2^{d+2}$ constraints resulting from Theorem 3.4. However, we observed that the number of constraints can in practice be reduced significantly by means of Observation 3.8.

In Table 3.1 and Table 3.2 we report the various dual gaps as explained in the previous section obtained with 31 instances from [Bro11] and after applying the aforementioned weak or strong preprocessing, respectively. The results in Table 3.1[4] computationally confirm the theoretical dominance of the disjunctive formulation with respect to the big-M one. However, the improvement is very small, almost negligible, and although it gets slightly higher if the lift-and-project closure is considered, still it does not look very promising to work with the disjunctive formulation. It is worth noting that also in terms of lift-and-project (unstrengthened) closures there is a theoretical dominance of the disjunctive formulation with respect to the big-M one, while the same does not hold for the strengthened versions of the closure because they are only approximations of the theoretical closure [Bon12].

In Table 3.2 we show only those instances from Table 3.1 in which the dual gap is not closed in at least one of the six columns[5]. By considering a much more strengthened version of the initial model (with tighter bounds on (ω, ω_0)) the quality of the bound provided by the disjunctive formulation looks better than that of the big-M one. More precisely, on the instances 4nl_16 and 4nl_17, in which the big-M formulation shows a small but still significant gap, the disjunctive formulation improves significantly, showing no gap. On two out of three of the remaining "difficult" instances, namely 2nl_10 and 3nl_13, the bound provided by the disjunctive reformulation is significantly better. On these last two instances the two formulations did not show any difference in Table 3.1, thus suggesting that the disjunctive reformulation takes advantage of the strengthening of the bounds of the (ω, ω_0) variables. We will come back to this in Section 3.6.

If CPLEX 12.6 is used as a black-box for solving the two MIQP formulations to optimality instead of just computing dual gaps, we do not observe much of a difference in terms of computing times and branch-and-bound nodes between the two formulations. Precisely, in the case of the disjunctive formulation we decide to add all constraints that are not present in the big-M one (see again Observation 3.9) as *user cuts* to CPLEX, instead of imposing them as regular constraints. This allows the solver to decide a proper way of using the constraints, which is a conservative strategy in case no clear assessment on their strength is possible. However, as anticipated, no significant difference is observed.

[4]Shown are only 27 instances, while the mean values are computed over all 31.

[5]The mean values are computed over these 8 instances only.

3.5.2 Complementary indicator constrains: Job Shop Scheduling

We now turn to JSS. A MILP formulation with big-M constraints has been given in (1.43) - (1.49), and we state one with indicator constraints here.

$$\begin{aligned}
\min \quad & \Omega && && (3.20)\\
\text{s.t.} \quad & s_{ik'} \geq s_{ik} + p_{ik} && \forall\, i \in [n], k, k' \in [m] :\ O_{ik} < O_{ik'} && (3.21)\\
& [x^k_{ij} = 1] \implies [s_{kj} \geq s_{ki} + p_{ki}] && \forall\, i < j \in [n],\ k \in [m] && (3.22)\\
& [x^k_{ij} = 0] \implies [s_{ki} \geq s_{kj} + p_{kj}] && \forall\, i < j \in [n],\ k \in [m] && (3.23)\\
& \Omega \geq s_{ik} + p_{ik} && \forall\, i \in [n], k \in [m] && (3.24)\\
& s_{ik} \geq 0 && \forall\, i \in [n], k \in [m] && (3.25)\\
& x^k_{ij} \in \{0, 1\} && \forall\, i < j \in [n], k \in [m]. && (3.26)
\end{aligned}$$

The above is precisely the case of the complementary disjunctions discussed in Section 3.4.1, in particular Corollary 3.11. It is an easy exercise to verify the technical condition $J_1(g) = J_2(g)$ required therein. Note therefore that each indicator constraint could be rewritten by either a single big-M constraint or four constraints resulting from Corollary 3.11, one of which can be shown to be always redundant. As bounds on the variables, we use the ones calculated according to (1.57) and (1.58).

In Table 3.3 we consider 34 JSS instances from the literature [ABZ88, FT63, KH06][6]. The results in therein show that there is some advantage in using the disjunctive formulation with respect to the big-M one, although the improvement in the quality of the dual bound is not sufficient to make any of the 34 instances above significantly easier for a general-purpose MILP solver if used as a black-box. More precisely, within one hour CPU time limit, CPLEX 12.6, used as described in the previous section, solves to optimality the same 17 instances for both formulations and no dominance in performance is shown: the big-M formulation is slightly better in terms of arithmetic mean, while the reverse is true in geometric mean.

However, we believe that combining the disjunctive reformulation, lift-and-project cuts and effective special-purpose propagation (i.e., tailored for JSS) could lead to an effective algorithm for optimally solving difficult JSS instances. Part of this is precisely the topic of Section 3.6.2.

[6]The dual gaps are computed with respect to the best known solution for instances la23 and la26. An asterisk indicates that the computation of the lift-and-project closure hit the time limit of two CPU hours.

3.5.3 Almost complementary indicator constraints: TSPTW

As a last computational example we consider the TSPTW introduced in Section 1.5.1. We recall the MILP formulation, but this time with indicator constraints,

$$\min \quad \sum_{\substack{(i,j)\in\mathcal{A}\\ i\neq q, j\neq p}} t_{ij}x_{ij} \tag{3.27}$$

$$\text{s.t.} \quad \sum_{j\in\delta_i^+} x_{ij} = 1 \qquad \forall\, i \in \mathcal{N}\setminus\{q\} \tag{3.28}$$

$$\sum_{i\in\delta_j^+} x_{ij} = 1 \qquad \forall\, j \in \mathcal{N}\setminus\{p\} \tag{3.29}$$

$$[x_{ij}=1] \implies [a_i + p_i + t_{ij} \le a_j] \qquad \forall\,(i,j)\in\mathcal{A} : i\neq q \tag{3.30}$$

$$l_i \le a_i \le u_i \qquad \forall\, i\in\mathcal{N} \tag{3.31}$$

$$x_{ij} \in \{0,1\} \qquad \forall\,(i,j)\in\mathcal{A} : i\neq q, j\neq p. \tag{3.32}$$

Taking two binary variables corresponding to the same arc, x_{ij} and x_{ji}, one can show that the associated indicator constraints imply the relation $x_{ij} + x_{ji} \le 1$. Thus, the above is precisely the case of almost complementary indicator constraints of Section 3.4.2, and we can compare a big-M formulation with a disjunctive formulation in the original space of variables. Bounds on the variables are taken simply from (3.31). The number of constraints needed for reformulating each indicator constraint is exactly the same as in the case of JSS in the previous section.

In addition to what has been reported in the tables of the two previous sections, we also report the dual gaps of the continuous relaxation of the big-M formulation where a ternary variable is used to model the disjunction as described in Section 3.4.2.1 (column big-Mz lp), and the continuous relaxation of the associated disjunctive reformulation according to Proposition 3.17 (column CHz lp). For these two (weaker) cases, however, we do not compute the closures. We use 50 TSPTW instances from [Asc95]. On the one side, the numbers in Table 3.4[7] confirm the dominance relationships discussed in Section 3.4.2.1, and, on the other side, show a neat advantage obtained by using the disjunctive formulation (column CHx lp) with respect to the big-M one (column big-Mx lp). This is also true for closures and the improvement becomes even more significant.

As in the JSS case, however, there is no instance that is not solved with the big-M formulation, but is instead solved with the disjunctive formulation within the CPU time limit of one hour using CPLEX 12.6. Nevertheless, over the 36 (out

[7] Shown are only 26 instances, but the mean values are computed over all 50. Again, an asterisk indicates that the computation of the lift-and-project closure hit the time limit of two CPU hours.

of 50) instances solved by CPLEX in both cases, there is a neat improvement of both the running times and the number of branch-and-bound nodes. Precisely, by considering only the 21 nontrivial instances for which each of the two formulations is solved in at least 0.5 secs, the running time goes from 107.4 seconds to 72.3 in arithmetic mean, and from 33.1 to 27.4 in geometric mean. In turn, the number of nodes changes from 524,557.7 to 382,049.7 in arithmetic mean, and from 140,002.1 to 96,037.7 in geometric mean. This roughly corresponds to a 30% reduction in the number of nodes, which looks like a promising result.

3.6 Bound tightening

At several points throughout this chapter, the importance of the availability of bounds on the variables in the context of MIPs with indicator constraints has been noted. This importance can be seen directly in the reformulations of indicator constrains with either big-M constraints or disjunctive reformulations. In the case of affine functions, consider for example (3.8) and (3.15). But also in the nonlinear case it can be seen when looking at the definition in (3.14). Thus, the constraints of the reformulations we studied in the previous sections are somehow parametrized in the bounds on the involved variables. Of course, when the binary variable involved in the indicator constraint is zero or one, changing these parameters (the bounds), does not have any effect. The constraint will just be active or inactive. This is not true in the continuous relaxation though. As was stressed in Section 3.1, big-M constraints are characterized by weak continuous relaxations. Therefore it seems desirable to strengthen these relaxations, and it is easy to see that a big-M constraint with a big-M as in (3.8) for example, is dominated by one in which more restrictive bounds are used. The same is true for the constraints that are constructed with (3.14). Also the computational results in Section 3.5.1 suggest that the reformulations benefit from the use of bound tightenings. Even more interesting, Table 3.2, that shows the results in which strong preprocessing was used to obtain more restrictive bounds on the variables of the supervised classification problem, suggests that the perspective reformulation benefits more than the big-M one. To further analyze this computational evidence, we explore bound tightening techniques and their use in MILPs with indicator constraints in this section, and in particular in conjunction with disjunctive reformulations in the case of JSS in Section 3.6.2.

3.6.1 Locally implied bound cuts

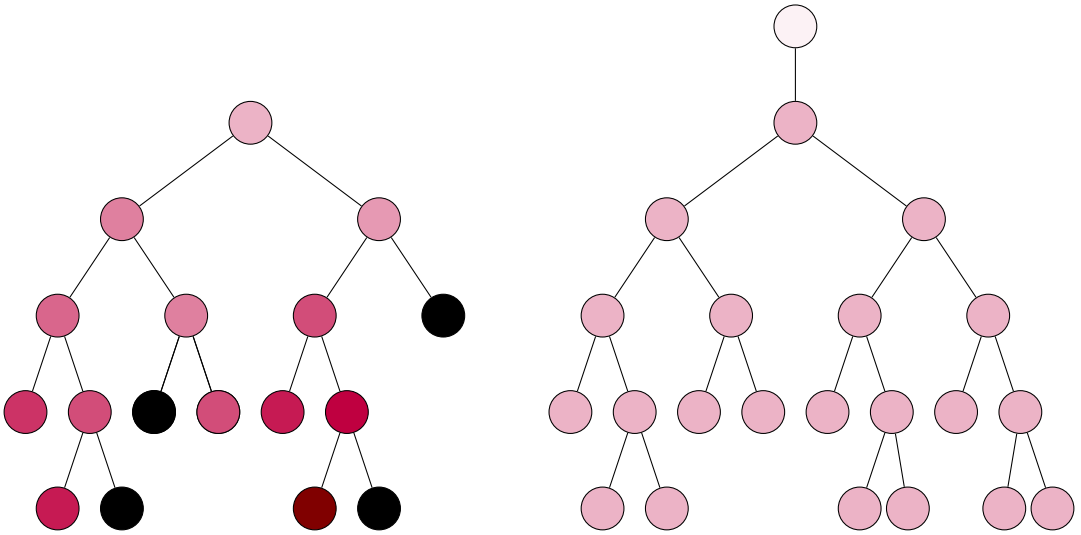
As for MILPs and the more classical way of reformulating indicator constraints, i.e., the big-M method, using tightened bounds on the variables in order to strengthen the

LP-relaxation of a big-M constraints is relatively easy, cf. (3.8). For the purpose of this section, we imagine two conceptually opposed ways of using tightened bounds on the variables inside big-M coefficients of a reformulation of an MILP with indicator constraints. We stress that for the moment, we are exclusively concerned with how to use these bounds, but not how to obtain them. That is, whenever we say *bound tightening*, this has to be understood as any generic algorithmic way of deducing tighter bounds on the variables.

The first of the two aforementioned opposed ways of using tightened bounds can be motivated in the following way. Since an MILP is usually solved by branch & bound, cf. Section 1.1.2, we can imagine to use local bounds on the variables in every node of the search tree, i.e., bounds that are valid in the subproblem associated to the node only. Whenever going down in the search tree, these bounds can only get more restrictive, since the subproblem of any node is more constrained than the one of its parent node. As an example take JSS, where we have seen that bounds on the start variables can be obtained via the heads and tails in the disjunctive graph in Section 1.5.2.1. When going down in the search tree, more conjunctive arcs are added to the disjunctive graphs associated to the nodes' subproblems, and the heads and tails can only increase. This in turn leads to more restrictive bounds on the start variables, cf. (1.57) and (1.58). But also in any other MILP, it remains true that local bounds on the variables can only get tighter when the depth of the tree increases. Together with the fact that tighter bounds on the variables can only strengthen the continuous relaxation of big-M constraints, this suggests the use of these local bounds in the big-M coefficients at every node in the search tree. Even more, it suggests that increased effort on bound tightening at the nodes can be advantageous. The general idea of such a procedure is schematically depicted in Figure 3.3 (a), where tighter continuous relaxations are symbolized by darker colors.

The reasons why an approach exploiting local bounds as described above has traditionally not been used when indicator constraints are present lie in the technicalities of the implementation of an MILP code. A big-M constraint with a big-M that is based on locally valid bounds can be seen as a locally valid cutting plane that has to be separated at a non-root node in the search tree. However, the separation of cutting planes at non-root nodes usually leads to a loss of efficiency in the way the search tree can be handled. For example, the use of warm starting techniques becomes slightly more involved.

For the reasons outlined above, the second and more traditional way of using tightened big-M coefficients in MILP is much less aggressive. Bound tightening is conducted in order to obtain globally valid bounds on the variables (i.e., bound tightening is conducted only at the root node), which are then used to build a big-M formulation, that in turn is solved by branch & bound without altering the



(a) Strengthened big-M coefficients at every node in the tree

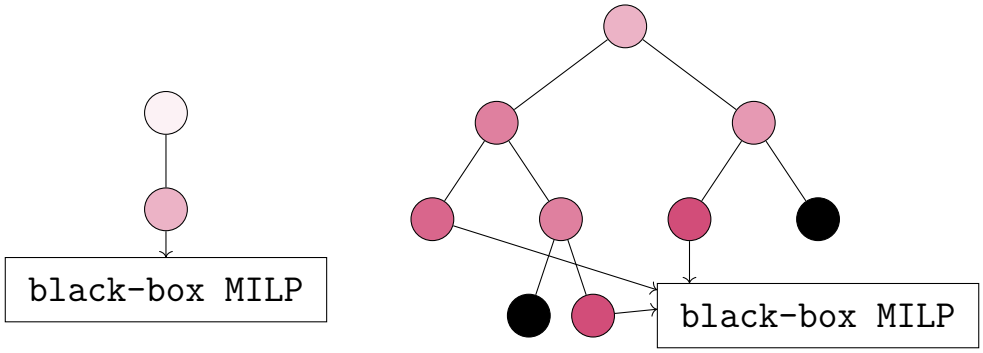
(b) Strengthened big-M coefficients at the root node only

Figure 3.3: The use of strengthened big-M coefficients inside a search tree

coefficients again. This is schematically depicted in Figure 3.3 (b). The trade-off between the two described approaches should become clear from Figures 3.3 (a) and (b), and from what has been said so far. While in the first case, tightened LP relaxations lead to more pruning in the tree, the (longer) list of nodes in the second case can be explored more efficiently.

A recent study of the use of locally valid big-M coefficients together with aggressive bound tightening techniques can be found in [BBF+14]. CPLEX 12.6.1 allows for the use of so-called locally-implied-bound cuts in the context of MIPs with indicator constraints, that are based precisely on tightened big-M coefficients, as outlined above. These cuts work particularly good on instances of supervised classification problems from Sections 1.5.4 and 3.5.1.

An interesting connection to non-convex MINLPs can be drawn here. As we have seen, the continuous relaxation of a reformulation of an indicator constraint depend on the bounds of the variables that are present in the constraint. The same is true for the relaxations that are usually used for non-convex nonlinear constraints, see Section 1.2.4 and in particular Figure 1.3. Just like there, bound tightening techniques are therefore a tool that should be taken into consideration when dealing with MILPs with indicator constraints. In some sense, MILP can learn from MINLP technology here. However, in that respect, both can learn from CP regarding the use of propagation. This is precisely the motivation of the next section.



(a) Black-box MILP solve right after the root node

(b) Black-box MILP solves of nodes with positive depth

Figure 3.4: The use of black-box MILP solvers inside a search tree

3.6.2 A tree-of-trees approach for MILP with indicator constraints

Inspired by the two extremes outlined in the previous section (cf. Figure 3.3), we present here an experimental, generic approach for MILPs with indicator constraints. The extreme sketched in Figure 3.3 (b) can also be seen as applying bound tightening at the root node, and then using an MILP solver as a black-box, see Figure 3.4 (a). Our approach is something in between the two extremes in Figure 3.3. The idea is to use bound tightening inside the search tree up to a certain point, in order to then use an MILP solver as a black-box on the subproblems of the open nodes, see Figure 3.4 (b). Roughly speaking, the original problem is decomposed into several subproblems, each of which has a tightened continuous relaxation, so as to make them tractable enough for an MILP solver. One motivation behind this is of course the fact that we can make use of the sophisticated MILP technology when solving these subproblems. Such an approach could be attributed the descriptive name *tree of trees*, because a series of MILPs is generated in a tree-search fashion, and each of them is then solved as an MILP, i.e., by exploring yet another tree.

We assume to have an underlying MILP with indicator constraints, that is a special version of (3.2) - (3.6),

$$\begin{aligned}
\min \quad & c^T x + d^T z && && (3.33)\\
\text{s.t.} \quad & Ax + Wz \leq g && && (3.34)\\
& [z_{i_k} = 1] \implies [a_k^T x \leq b_k] && \forall\ k = 1, \dots, \bar{K} && (3.35)\\
& [z_{i_k} = 0] \implies [\tilde{a}_k^T x \leq \tilde{b}_k] && \forall\ k = \bar{K}+1, \dots, \tilde{K} && (3.36)\\
& x \in \mathbb{R}^{\tilde{n}} && && (3.37)\\
& z \in \{0,1\}^K. && && (3.38)
\end{aligned}$$

(3.33) - (3.38) can be obtained from (3.2) - (3.6) by assuming that all involved functions are affine, and by assuming that bounds on the variables in the sets S_j^k in (3.2) are independent of j and k, and equal to some global bounds $\underline{x}$ and $\overline{x}$ implied by the global constraints (3.34). Then, each of the binary variables z_i corresponds to complementary indicator constraints as in Section 3.4.1, if

$$i \in \{i_k \mid k = 1, \dots, \bar{K}\} \cap \{i_k \mid k = \bar{K} + 1, \dots, \tilde{K}\}.$$

Otherwise, it corresponds to a single indicator constraint as in Section 3.3.2. Note that for ease of exposition, in (3.37) we assume that there are no integer-constrained variables x. However, the following approach can easily be extended to the case where this restriction does not hold.

Our generic tree-search method for a problem of the form (3.33) - (3.38) is inspired by and similar to the general branch-and-bound procedure for an MILP, cf. Algorithm 1.1. Yet, there are some amendments that we will describe in detail now. A node η in our search tree is given by the original MILP with indicator constraints (3.33) - (3.38), together with a set of binary variables that are fixed to one, denoted by $\mathcal{B}_+^\eta$, and a set of binary variables that are fixed to zero, denoted by $\mathcal{B}_-^\eta$. Hence, a node still corresponds to a subproblem of the original one. Further, it is attributed lower and upper bounds on the variables x, denoted by $\underline{x}^\eta$ and $\overline{x}^\eta$, respectively. All these sets and quantities are initially inherited from the parent node. Finally, every node is equipped with a lower bound on the optimal objective value of its associated subproblem, denoted by ℓ and initially set to the optimal objective value of the parent node. Every node has a depth, which is the succeeding integer of the depth of the parent node. The root node ρ has $\mathcal{B}_+^\rho = \mathcal{B}_-^\rho = \emptyset$, $\ell = -\infty$, depth equal to zero and its associated bounds on the variables are equal to the global ones $\underline{x}$ and $\overline{x}$.

In order to measure the quality of the dual bound coming from the continuous relaxation of a reformulation of the subproblem associated to a node η_1 with respect to the one associated to another node η_2, we define their so-called relative tightness as

$$\begin{aligned}\tau(\eta_1, \eta_2) :=& \alpha \cdot \frac{(|\mathcal{B}_+^{\eta_1}| + |\mathcal{B}_-^{\eta_1}|) - (|\mathcal{B}_+^{\eta_2}| + |\mathcal{B}_-^{\eta_2}|)}{\tilde{K} - (|\mathcal{B}_+^{\eta_2}| + |\mathcal{B}_-^{\eta_2}|)} \\ &+ (1 - \alpha) \cdot \left(1 - \frac{\sum\limits_{\substack{k=1 \\ z_{i_k} \notin (\mathcal{B}_+^{\eta_1} \cup \mathcal{B}_-^{\eta_1})}}^{\bar{K}} |M_k^{\eta_1}| + \sum\limits_{\substack{k=\bar{K}+1 \\ z_{i_k} \notin (\mathcal{B}_+^{\eta_1} \cup \mathcal{B}_-^{\eta_1})}}^{\tilde{K}} |\tilde{M}_k^{\eta_1}|}{\sum\limits_{\substack{k=1 \\ z_{i_k} \notin (\mathcal{B}_+^{\eta_1} \cup \mathcal{B}_-^{\eta_1})}}^{\bar{K}} |M_k^{\eta_2}| + \sum\limits_{\substack{k=\bar{K}+1 \\ z_{i_k} \notin (\mathcal{B}_+^{\eta_1} \cup \mathcal{B}_-^{\eta_1})}}^{\tilde{K}} |\tilde{M}_k^{\eta_2}|} \right),\end{aligned}$$

where $\alpha \in [0,1]$ is some parameter, and M_k^η and $\tilde{M}_k^\eta$ denote the big-M coefficients of the indicator constraints (3.35) and (3.36), respectively, calculated as in (3.8) with the bounds of node η, i.e., $\underline{x}^\eta$ and $\overline{x}^\eta$. The relative tightness of two nodes is thus a weighted sum of two more specialized quantities. The first precisely measures the portion of the non-fixed binary variables in the second node η_2, that are instead fixed in η_1. The second quantity measures the relative change of the magnitude of the big-M coefficients corresponding to all indicator constraints with non-fixed binary variables in η_1. The relative tightness of two nodes is a measure for the relative quality of their continuous relaxations because, on the one hand, the number of fixed binary variables obviously contributes to this quality, as is true in any MILP with binary variables. On the other hand, we have seen that the magnitude of the big-M coefficients is another contributor of the quality of the LP-relaxation of a big-M reformulation of an indicator constraints. Of course we are not only interested in big-M reformulations, but especially in disjunctive reformulations. But also in this case, in light of Observation 3.9, the magnitude of the big-M coefficients is a valid indicator of the quality of the LP-relaxations. Finally, we define the absolute tightness of a node η as $\bar{\tau}(\eta) := \tau(\eta, \rho)$, where ρ is the root node.

The main difference between Algorithm 1.1 and our approach is the fact that the subproblems of some nodes are solved as LPs, while others are solved as MILPs. We denote by $\mathcal{T} = \mathcal{L} \cup \mathcal{M}$ the list of all open nodes, partitioned into an LP-list and an MILP-list. Nodes in $\mathcal{L}$ will be solved as LPs (so-called LP-nodes), while nodes in $\mathcal{M}$ will be processed as MILPs (MILP-nodes). $\mathcal{L}$ and $\mathcal{M}$ are always disjoint. The approach we propose, that is a tree-search procedure, can be roughly described by Algorithm 3.1.

As in the case of Algorithm 1.1, there are several degrees of freedom in Algorithm 3.1, and we are going to explain the single steps in more detail now.

3.6.2.1 Initialization

In order to initialize the upper bound UB, we run CPLEX's 12.6.2 root node heuristics on a big-M reformulation of (3.33) - (3.38). The upper bound UB might not only be important for pruning, but also for propagation. This is particularly true in the case of JSS, as we will see further down.

3.6.2.2 Node selection

As for choosing a node in line 3 of Algotirhm 3.1, in theory, various strategies to explore the search tree could be invented, possibly changing multiple times between the lists $\mathcal{L}$ and $\mathcal{M}$. However, we concentrate on one particular way that allows us to describe Algorithm 3.1 by two phases. As long as there are open LP-nodes in the

Algorithm 3.1: Tree of trees for indicator constraints

```
1  initialize UB and the root node ρ, set L = {ρ} and M = ∅;
2  while T ≠ ∅ do
3  |  choose η ∈ T, set T = T \ {η};
4  |  propagate fixings and bounds on η;
5  |  solve η and denote by (x̂, ẑ) and ŵ its optimal solution and objective
   |  value;
6  |  if ŵ < UB then
7  |  |  if ẑ integral then
8  |  |  |  update UB = ŵ;
9  |  |  else
10 |  |  |  choose a binary variable z_i with fractional ẑ_i;
11 |  |  |  create nodes η⁻ and η⁺ by adding z_i to B⁻_η and B⁺_η, respectively;
12 |  |  |  set T = T ∪ {η⁻, η⁺};
13 |  |  end
14 |  end
15 end
```

tree, we restrict to those. Only when there are no LP-nodes left, we start selecting MILP-nodes from the tree. Note that whenever an MILP-node is processed, the condition in Line 7 of Algorithm 3.1 is true and thus no new nodes, in particular no new LP-nodes, are added to $\mathcal{T}$. Thus, we have a first, the LP-phase, and a second, the MILP-phase of the algorithm. Formally, line 3 of Algorithm 3.1 can be written as:

---

if $\mathcal{L} \neq \emptyset$ **then** choose node η from $\mathcal{L}$;
else choose node η from $\mathcal{M}$;

Roughly speaking, the list of MILP-nodes is slowly filled during the LP-phase (see Section 3.6.2.6 below), and no MILP-node is processed before all LP-nodes are. In any case, when choosing from either list, we always select the node with minimum ℓ, thus pursuing a so-called best-bound-first strategy.

3.6.2.3 Node solve

In line 5, we build the MILP with indicator constraints associated to the subproblem of node η. By that, we clearly mean that for any fixed binary variable in $\mathcal{B}^\eta_+ \cup \mathcal{B}^\eta_-$, the respective constrained is either added to the global constraints, or simply discarded, depending on whether it is activated or deactivate by the particular value of the binary variable. We then build an MILP reformulation of the node's subproblem, containing the remaining indicator constraints, and solve it. More precisely, we

solve it as an MILP, if it was chosen from $\mathcal{M}$, but relax integrality and solve it as an LP, if it was chosen from $\mathcal{L}$. Every time we build a reformulation, we can further decide whether to use a single big-M constraint or all constraints coming from the disjunctive reformulation of each indicator constraint. In any case, we will always use the local bounds $\underline{x}^\eta$ and $\overline{x}^\eta$ to build either reformulation. We compare two strategies for now:

- `persp0`: never use the disjunctive reformulation, but only big-M constraints
- `persp1`: use the disjunctive reformulation at MILP-nodes only

Note that when using either of the strategies `persp0` or `persp1`, the LP-phase of Algorithm 3.1 is identical. Therefore, the list of MILP-nodes on the tree when entering the MILP-phase is the same. This makes the two strategies particularly attractive for measuring the impact of the additional inequalities coming from the disjunctive reformulation with local bounds: the same list of sub-MILPs is solved either with (`persp1`) or without (`persp0`) perspective inequalities[8]. As LP- and MILP-solver, we use CPLEX 12.6.2. Whenever an MILP is solved, we can set UB as a cutoff value, that is, we can add a constraint imposing that the objective value be upper bounded by UB[9].

3.6.2.4 Propagation and specialization for JSS

The motivation behind Algorithm 3.1 lies, as has been mentioned before, in the use of local bounds on the variables. Thus, an effective propagation procedure in Line 4 in order to tighten the bounds $\underline{x}^\eta$ and $\overline{x}^\eta$ as much as possible (in conjunction with deducing more binary fixings), is the probably most important part of Algorithm 3.1. In its general version, we allow the propagation procedure to be generic or problem-specific. As in the previous section, we do not study general bound tightening procedures here, but present the implementation of a propagation procedure for JSS. Note that the MILP formulation for JSS with indicator constraints (3.20) - (3.26) is in the form (3.33) - (3.38).

The propagation procedure for JSS consists of several subroutines that we describe in the following. We have seen how to compute bounds on the start variables of any subproblem of JSS via heads and tails in the disjunctive graphs in (1.57) and (1.58). The computation of the upper bound is based on local information (the tail) as well as global information (the upper bound UB). Therefore, in the case of JSS, we equip the nodes in the tree with the local heads and tails r^η_ν and q^η_ν, instead of

[8] A possible third strategy would be to use the disjunctive reformulation also in LP-nodes, say `persp2`. However, we did not test this strategy in our computational experiments.

[9] CPLEX for example provides a user interface for this.

Algorithm 3.2: Propagation for JSS

```
1  set k = 1;
2  do
3  |  call longest path;
4  |  call cpo;
5  |  do
6  |  |  call disj;
7  |  |  call adjust;
8  |  while progress is made;
9  |  if k = 1 then call global;
10 |  k = k + 1;
11 while progress is made;
```

the local bounds $\underline{x}^\eta$ and $\overline{x}^\eta$, directly (or, more precisely, with quantities that are as least the local heads and tails, see Observation 3.19). Also, many subroutines in the propagation procedure are based explicitly on these heads and tails. When solving any node (cf. Section 3.6.2.3), in order to be able to apply a reformulation, we always recover bounds on the variables via (1.57) and (1.58), using the currently known upper bound UB (that can differ from the trivial one in (1.59)). Via the sets $\mathcal{B}_+^\eta$ and $\mathcal{B}_-^\eta$, we can also build the disjunctive graph of the node's subproblem as described in Section 1.5.2.1. Therefore, for brevity, we will occasionally just refer to the node's disjunctive graph.

We omit the superscripts η from now on, assuming that all heads r_ν, tails q_ν, and sets of fixed binary variables $\mathcal{B}_+$ and $\mathcal{B}_-$ refer to the node in question. Tightening the bounds on the variables can be achieved by increasing r_ν and q_ν, and this is the aim of all of the following subroutines. Some of the procedures are also able to detect an infeasibility. In such cases, the node is fathomed immediately. The subroutines are the following.

- `longest path`: For each operation ν, we calculate its head in the node's disjunctive graph. If it is greater than r_ν, we update the latter. The same is done with the tail and q_ν.

Observation 3.19. *Some of the propagation procedures are able to modify r_ν and q_ν without adding binary fixings, which is why calculating for example the head in a node's disjunctive graph might result in a quantity strictly lower than r_ν. Therefore, in any node, we keep track of r_ν and q_ν as quantities that are at least the heads and tails in the disjunctive graph, but not necessarily equal to them.*

- `cpo`: In this subroutine we use CPO 12.6.2 as a black-box. In particular, we compute local bounds on the start variables via (1.57) - (1.58) with the

currently known UB, and use them to build a CP model for (3.20) - (3.26). Then, we add all the precedence relations coming from fixed binary variables in $\mathcal{B}_+$ or $\mathcal{B}_-$ and call the initial propagation of CPO 12.6.2. This can result in detecting infeasibility or in a tightening of the bounds on the start variables. In the latter case, we recover the increased r_ν and q_ν via (1.57) - (1.58).

- `disj`: This procedure has been proposed in [CP94]. Roughly speaking, it represents an efficient way of detecting all binary fixings based on the following observation. When for two operations that are to be executed on the same machine, say $\nu = (i, k)$ and $\varrho = (j, k)$, we have that

$$r_\varrho + p_\varrho + p_\nu + q_\nu > UB,$$

then certainly operation ν has to precede operation ϱ in any feasible solution with an objective value that is at least as good as UB. Then if $i < j$, we can add x_{ij}^k to $\mathcal{B}_+$, or otherwise add x_{ji}^k to $\mathcal{B}_-$. In any case, we can update $r_\varrho = \max(r_\varrho, r_\nu + p_\nu)$ and $q_\nu = \max(q_\nu, q_\varrho + p_\nu)$.

- `adjust`: Also this procedure has been proposed in [CP94]. For any subset of operations that are to be executed on the same machine, say $\Psi = \{(i, k) \mid i \in I\}$ for some $I \subseteq [n]$, one can show that if for some $t_0 \in \Psi$ we have

$$\min_{\nu \in \Psi} r_\nu + \sum_{\nu \in \Psi} p_\nu + \max_{\nu \in \Psi \setminus \{\nu_0\}} q_\nu > UB,$$

all $\nu \in \Psi \setminus \{\nu_0\}$ have to precede ν_0 in any feasible solution with an objective value that is at least as good as UB. One can then fix all the corresponding binary variables and set $r_{\nu_0} = \max\left(r_{\nu_0}, \max_{\nu \in \Psi \setminus \{\nu_0\}} r_\nu + p_\nu\right)$. The proposed procedure represents an efficient way of detecting all binary fixings based on this observation. An additional and analogue procedure by interchanging the roles of heads and tails is used as well.

- `global`: This procedure, again proposed in [CP94], can be seen as a kind of probing. Heads and tails are dichotomically increased inside their domain, and each time other propagation procedures are called as subroutines. If an infeasibility is detected, the respective counterpart can be updated. E.g., if setting r_t to a certain value f leads to an infeasibility, operation t cannot start later than $f-1$, and $\underline{s}_t$ can be set to $f-1$, so that q_t can be updated via (1.58). The procedure can be described as in Algorithm 3.3. Again, an additional and analogue procedure by interchanging the roles of heads and tails is used as well.

Algorithm 3.3: Subroutine `global`

```
1  make copies r̃_ν and q̃_ν of all heads and tails;
2  foreach operation ν do
3  |  do
4  |  |  r_ν = ⌈(UB−(q_ν+p_ν)+r_ν)/2⌉;
5  |  |  do
6  |  |  |  call longest path, cpo, disj and adjust;
7  |  |  while progress is made;
8  |  |  if infeasibility is detected then
9  |  |  |  q̃_ν = UB − (r_ν + p_ν) + 1;
10 |  |  end
11 |  while no infeasibility is detected and r_ν < UB − (q_ν + p_ν);
12 |  reinstall r_ν = r̃_ν and q_ν = q̃_ν for all ν and undo all binary fixings that
   |  were made in line 6;
13 end
```

The whole propagation procedure can be described as in Algorithm 3.2. The procedure `global` is rather time-consuming and we use it only in the first iteration of Algorithm 3.2. In MIP, we usually do not want to find the whole set of equivalent optimal solutions, but at least one. In this spirit, in case a feasible solution has already been found, we are only interested in solutions with an objective value that is strictly better than UB. If in addition the data is integral (which is true for all JSS instances in this chapter), we can artificially decrease UB by 1 inside the propagation procedure.

3.6.2.5 Branching strategy

As a branching strategy in Line 10 of Algorithm 3.1, we choose one inspired by so-called strong branching [AKM05]. For each fractional binary variable z_i, we actually create the two possible child nodes η_i^+ and η_i^- obtained by adding z_i to $\mathcal{B}_+^\eta$ and $\mathcal{B}_-^\eta$, respectively, and call the propagation procedure in Line 4 on these two nodes. This can result in three cases:

- Both η_i^+ and η_i^- are detected infeasible by propagation. If this happens, the parent node η is fathomed immediately.
- Exactly one of the nodes η_i^+ and η_i^- is detected infeasible, say η_i^+. In that case, we add z_i to $\mathcal{B}_-^\eta$ and go back to line 4 of Algorithm 3.1.
- None of the two is detected infeasible. In that case, we compute their relative tightness with respect to η, and then compute

$$\sigma_i := (\tau(\eta_i^+, \eta) + \varepsilon) \cdot (\tau(\eta_i^-, \eta) + \varepsilon)$$

Provided that no infeasibility was detected in any of the two nodes for any fractional z_i, we choose the one with maximal σ_i.

We make a technical remark. The above branching strategy is relatively time-consuming, because the propagation procedure is called twice for every fractional binary variable in each iteration. One could therefore think to apply a lighter version of Algorithm 3.2 inside the branching step in Line 10 of Algorithm 3.1. In the case of JSS for example, we omit the relatively time consuming subroutine `global` when calling the propagation procedure inside the branching step.

3.6.2.6 Adding nodes to the tree

Fractional solutions that lead to the creation of new nodes can only occur when the node η was chosen from $\mathcal{L}$. We have the following strategy for deciding whether a new node as in Line 11 of Algorithm 3.1 will be added to $\mathcal{L}$, and thus will be processed as a LP-node, or to $\mathcal{M}$, and thus will eventually be solved as an MILP, based on its absolute tightness. We call the propagation procedure on the node that is to be added to $\mathcal{T}$ and then compute its tightness $\bar{\tau}$. When the node is "tight enough", we add it to $\mathcal{M}$, or otherwise, we add it to $\mathcal{L}$. Formally, after having chosen some threshold γ, Line 11 of Algorithm 3.1 can be written as:

---

propagate fixings and bounds on η;
if $\bar{\tau}(\eta) \geq \gamma$ **then** $\mathcal{M} = \mathcal{M} \cup \{\eta\}$;
else $\mathcal{L} = \mathcal{L} \cup \{\eta\}$;

In case the propagation is ineffective and the tightness of a node is not augmented substantially during the algorithm, this strategy would continue branching for a long time and create too many new nodes. That is why we also add a newly created node to $\mathcal{M}$ when its depth has reached $\bar{d} = 10$.

3.6.2.7 Computation for JSS

We now report on computational experiments with Algorithm 3.1 and the subroutines described above on a set of small sized JSS instances, partly used in Section 3.5.2. In particular, we used all instances with up to 15 jobs and 10 machines therein, but as well included instances with 15 jobs and 15 machines found at [JSS], leading to a total number of 29 instances, in order to compare the two strategies `persp0` and `persp1`. Algorithm 3.1 has been coded in C/C++ and CPLEX's C API. All experiments were conducted on a single core of a 3.1 GHz quad-core machine with 1.96 GB RAM, and we imposed a time limit of 3 hours (indicated by ∞, when reached). Table 3.5 summarizes several statistics regarding the application of Algorithm 3.1

with parameters $\alpha = 0.5$ and $\gamma = 0.15$ to those instances of the testbed where at most one of the two strategies hit the time limit. Shown are the duration of the LP-phase of the algorithm in seconds (column t_{LP}), the number of MILP-nodes in which an MILP was solved to optimality (column n_{MILP}), the computing time of the whole algorithm in seconds (column time) and the average number of branch-and-bound nodes needed to solve the MILPs of the MILP-nodes (column #nodes). Note that the first two quantities are the same for both strategies[10]. The mean values are computed over all instances for which none of the two strategies reached the time limit.

In order to interpret the results in the table correctly, it is important to shed some light on the role of the current UB inside the propagation procedure for JSS. In fact, as was anticipated in Section 3.6.2.1, the propagation is highly dependent on this upper bound. This can be seen, for example, in the subroutines `disj` and `adjust`, but also in the calculation of the upper bound on the start variables, cf. (1.58), that is in turn used to build a constraint programming model in the subroutine `cpo`. For example, we observe that the number of MILPs that are solved during the algorithm is relatively low in all cases, but note that this number is not necessarily equal to the number of MILP-nodes that are created. In fact, it often happens that at the end of the first MILP-solve, an UB is found that is relatively close to the optimal solution. With that UB, a lot of the open MILP-nodes on the tree can be declared infeasible by the propagation procedure and are thus not counted in n_{MILP}. However, for the same reason, before the first MILP is solved during the algorithm, UB is relatively far away from the optimal solution and the propagation procedure on the so far created nodes, including the MILP-node that is processed first, has not been as effective as it could have been with a "good" UB. Thus, the computing time needed to solve this first MILP is usually relatively high as compared to the remaining MILPs. This might also be one reason for the fact that so far, we are not able to beat default CPLEX 12.6.2 in solving most JSS instances with Algorithm 3.1. Nevertheless, we see an advantage of strategy `persp1` that makes use of the disjunctive inequalities, in terms of computing times but as well in terms of the branch-and-bound nodes that are explored, especially in the more difficult instances in Table 3.5. This seems like a promising result.

[10] When executing the algorithm with the two different strategies, variability in the duration of the LP-phase is negligible.

3.7 Outlook

We believe that the material in this chapter can stimulate future research in two ways, one of more theoretical nature, the other more practical. We have seen some kind of evolution in the various theorems presented. The high-dimensional space description of Theorem 1.11 was first projected onto the original space of variables in the special case of Theorem 3.2, that was then extended in several steps and possibly varying directions throughout Theorem 3.4 to Proposition 3.17. We believe that this line of research is open for further extensions, involving, for example, either more complicated or simply more functions.

On the practical side, we still work on improving the results obtained with Algorithm 3.1 on JSS, that can be classified as only preliminary so far. There is much more room to do propagation on these problems, for example by the concept of decision diagrams [Ake78]. Also, CP is still performing significantly better than MILP on JSS. CP regularly has, however, difficulties in proving the optimality of feasible solution of hard JSS instances. We believe that MILP can play an important complementary role in this respect, possibly by integrating not only propagation, but also CP-based search techniques into our MILP-based algorithm.

4 MILPs with non-contiguous split domains

In this chapter, we study an approach to certain MILPs that is inspired by simple observations regarding some of the modeling techniques provided by the MILP paradigm. We start by illustrating these observations through an examplatory sudoku game as in Figure 4.1. In a sudoku of dimension n, the task is to place a number between 1 and n^2 into each cell of a $n^2 \times n^2$-grid, given that some cells are initially filled. The difficulty arises from the fact that in each row, each column, and each of the designated smaller $n \times n$-squares, each number is allowed to appear only once. If we were to model this combinatorial problem by any established programming paradigm, we would probably start by assigning a variable to each cell, so that its value is to describe the number we are going to place there. For example, the variable $y_{5,5}$ in Figure 4.1 is to describe the number we are going to place in the 5th row of the 5th column. Let us neglect for a moment most of the constraints that describe the entire problem and just focus on the single variable $y_{5,5}$. If the programming paradigm we choose is CP, we are allowed to work with

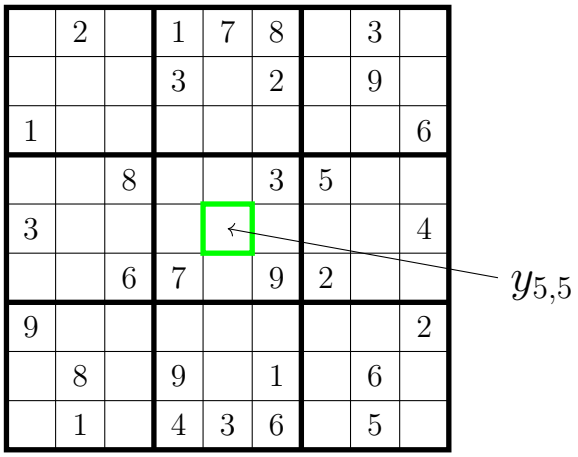


Figure 4.1: A 3×3 sudoku

finite domains. That is, we could just write down that the domain of $y_{5,5}$ is

$$D(y_{5,5}) = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

In Mixed Integer Programming instead, we would write

$$1 \leq y_{5,5} \leq 9, \quad y_{5,5} \in \mathbb{Z},$$

i.e., $y_{5,5}$ is an integer variable with lower and upper bounds equal to 1 and 9, respectively. What triggers the attention in this illustrative example is what happens when we take into account the initial problem data in Figure 4.1. We can see that $y_{5,5}$ cannot take the values 3, 4, 7 and 9. Back in CP, the problem could be modeled with `alldifferent` constraints, cf. Definition 1.8 and Example 1.9, and any effective propagation procedure would assure these straightforwardly implementable changes with respect to our initial description of the domain. We can just cancel these values from therein and get

$$D(y_{5,5}) = \{1, 2, 5, 6, 8\}.$$

In Mixed Integer Programming, this process is less straightforward. Of course, we can reduce the upper bound and write

$$1 \leq y_{5,5} \leq 8, \quad y_{5,5} \in \mathbb{Z},$$

but we have not yet accounted for the constraint $y_{5,5} \notin \{3, 4, 7\}$, which in this direct form is not foreseen in the modeling tools provided by MIP.

Of course all integer programmers would object and (correctly) assert that they know how to model the above situation. One can simply take a series of additional binary variables and write

$$\begin{aligned} y_{5,5} = \; & x_1 + 2x_2 + 5x_5 + 6x_6 + 8x_8, && (4.1) \\ & x_1 + x_2 + x_5 + x_8 = 1, && (4.2) \\ & x_1, x_2, x_5, x_6, x_8 \in \{0, 1\}. && (4.3) \end{aligned}$$

This is what has been done in integer programming for 50 years now and is thus well-proven practice. Yet, it highlights exactly the point we want to make with our admittedly informal example. In order to express the fact that a variable be different from a value in the "interior" of its domain[1], we have to use auxiliary variables. We

[1] In case of an integer variable, we actually mean the integer values inside the interior of the convex hull of its domain.

always have to take some kind of detour and introduce additional variables, but there are no means to model this phenomenon explicitly. As foreshadowed above, in CP such constraints are routinely expressed and, much more important, also exploited algorithmically, e.g., through filtering and propagation. In this chapter, our aim is to analyze whether and how we can exploit such explicit representations in MILP.

Any integer-constrained variable has, strictly speaking, a domain that is a non-contiguous set. But in the above example, it is in addition true that the domain of variable $y_{5,5}$ is given by the integer values inside an already non-contiguous set,

$$y_{5,5} \in \big([1,2] \cup [5,6] \cup \{8\}\big) \cap \mathbb{Z}. \tag{4.4}$$

In the above case, one could also say $y_{5,5}$ has holes in its domains. For example, it is allowed to take the values 2 and 5, but nothing in between. That is, there is a hole between 2 and 5. More formally, this can be expressed by disjunctions. In particular, in this situation, the disjunction

$$y_{5,5} \leq 2 \vee y_{5,5} \geq 5 \tag{4.5}$$

is valid: every feasible solution has to satisfy at least one of the (in this case two) disjunctive terms. Actually, the above is a special case of real-valued split disjunctions, that we will introduce more formally in Section 4.1, and that are not restricted to integer-constrained variables as in the above example. Therein, we will also introduce an extension of the class of MILPs in such a way that non-contiguous domains like in (4.4), or even more general ones involving so-called split vectors, can be written as explicit constraints. In particular, disjunctions of the form (4.5) are then valid disjunctions. We will analyze how such constraints can play a role in the context of certain MILPs in Sections 4.1.1 and 4.1.2. Then, as anticipated above, we are interested in ways of exploiting this explicit information algorithmically, of course based on MILP techniques. In particular, we will spend much time on the derivation of MILP cutting planes in Section 4.2. Later on, in Section 4.3, we will analyze the exploitation of these constraints in branching, leading to an experimental exact algorithm for problems belonging to the aforementioned extension of MILP problems. The findings of this chapter are based on joint work with Pierre Bonami, Andrea Lodi and Andrea Tramontani.

4.1 Real-valued split disjunctions

For the moment, we assume the existence of an underlying MILP

$$\min \quad c^T x \tag{4.6}$$

$$\text{s.t.} \quad Ax = b \tag{4.7}$$

$$x \in \mathbb{R}^{n-p}_+ \times \mathbb{Z}^p_+. \tag{4.8}$$

We will then consider disjunctions of the form $\pi^T x \leq \pi_l \vee \pi^T x \geq \pi_u$, where the triple (π^T, π_l, π_u) belongs to the set

$$\Pi := \{(\pi^T, \pi_l, \pi_u) \in \mathbb{R}^{n+2} \mid \pi_l < \pi_u\}.$$

We call such disjunctions *real-valued split disjunctions*, or just *real-valued splits*. π alone is also called the split vector. The case of a real-valued split such that the split vector has its support on the integer constrained variables and only integral entries, i.e.,

$$\pi_i = 0 \ \ \forall\ i = 1, \ldots, n-p, \quad \pi_i \in \mathbb{Z} \ \ \forall\ i = n-p+1, \ldots, n, \quad (\pi_l, \pi_u) \in \mathbb{Z}^2$$

has already been studied in [ACL05b] under the name *general split disjunction*. The extensively studied notion of *ordinary split disjunctions* [CKS90], usually just called split disjunctions, can be seen as the special case of general splits with $\pi_l + 1 = \pi_u$. In a general split, the data is assumed to be integer because historically, split disjunctions were designed to act on the integer variables of (4.6) - (4.8) only. We introduce the notion of real-valued splits in order to be able to consider split disjunctions that act on the continuous variables as well. However, in Sections 4.2.4 and 4.3 we will mostly pay attention to integer variables, i.e., to general splits. We are particularly interested in the computational advantages of considering general splits (π^T, π_l, π_u) with $\pi_u - \pi_l > 1$ over corresponding ordinary splits, that is, splits of the form $(\pi^T, \pi_0, \pi_0 + 1)$ with $\pi_l \leq \pi_0 \leq \pi_u - 1$. Therefore, in the remainder, we will use the denomination *wide split disjunctions* for general splits, due to the fact that their "width" can be greater than one, i.e., $\pi_u - \pi_l \geq 1$, whereas $\pi_u - \pi_l = 1$ for ordinary splits.

A natural question that arises is the one about the validity of a real-valued split disjunction for an MILP. While it is easy to see that any solution to (4.6) - (4.8) satisfies any ordinary split disjunction, this is not true for wide splits, not to mention real-valued splits. Inspired by (4.4), for the remainder of this chapter, we therefore introduce the notion of a Mixed Integer Linear Program with non-contiguous split

domains (MILPncsd),

$$\min \quad c^T x \tag{4.9}$$

$$\text{s.t.} \quad Ax = b \tag{4.10}$$

$$\pi^{k^T} x \in \bigcup_{j=1}^{L_k} [l_k^j, u_k^j] \qquad \forall\, k \in [K] \tag{4.11}$$

$$x \in \mathbb{R}_+^{n-p} \times \mathbb{Z}_+^p, \tag{4.12}$$

assuming that K real-valued split vectors in $\mathbb{R}^n$ and associated unions of sets are given. We note that the feasible region of a MILPncsd can be transformed to the union of a finite number of polyhedra. Thus, a MILPncsd is a special case of a Disjunctive Program [Bal79, Bal98]. For each $j = 2, \ldots, L_k$ and $k \in [K]$, without loss of generality, we will assume $u_k^{j-1} < l_k^j$, and thus, by construction, the real-valued split disjunction

$$\pi^T x \leq \pi_l \vee \pi^T x \geq \pi_u \tag{4.13}$$

with $\pi = \pi^k$, $\pi_l = u_k^{j-1}$ and $\pi_u = l_k^j$ is valid for (4.9) - (4.12). Equivalently, we say that π has a hole $[\pi_l, \pi_u]$ in its domain. We will see in Sections 4.1.1 and 4.1.2 how the generic notion of an MILPncsd can actually play a role in the context of certain (ordinary) MILPs.

A special role in the context of split disjunctions often play so-called simple splits, that is $\pi = \mathbf{e}_i$ for some i, the i-th unit vector. In that case the split disjunction translates to just $x_i \leq \pi_l \ \vee \ x_i \geq \pi_u$. If one of the given split vectors in (4.9) - (4.12) is a unit vector, that constraint just translates to the fact that there is a variable with a non-contiguous domain. Therefore, equivalent to $(\mathbf{e}_i, \pi_l, \pi_u)$ being valid, we can also say that variable x_i has a hole in its domain. Most of the examples we will present in Sections 4.1.1 - 4.1.2 are actually based on variables with non-contiguous domains and thus lead MILPncsd's in which all split vectors are simple split vectors. In theory, by introducing a new variable $x_{n+1} = \pi^T x$, every real-valued split disjunction can be reduced to a simple one. However, for analyzing whether non-contiguous split domains are valid for a given MILP, it might be disadvantageous to restrict to simple splits only, which is why in the following, we establish our theoretical results for not necessarily simple splits.

We will present some natural extensions of the theory of ordinary (and general) split disjunctions, with particular focus on the separation of cutting planes in Section 4.2. We will make use of the fact that often, throughout the literature on split cuts [CKS90], integrality of the split vector and adjacency of the right-hand sides

are only required for arguing the validity of the split, but not for deriving valid inequalities. However, this observation is not true anymore when one goes over to integer integer strengthening principles, that we will analyze those in Section 4.2.3. In the remainder of this chapter, we call the MILP that is obtained by replacing (4.11) in (4.9) - (4.12) by

$$l_k^1 \leq {\pi^k}^T x \leq u_k^{L_k}$$

the *MILP-relaxation* of (4.9) - (4.12). Also, the *LP-relaxation* of (4.9) - (4.12) is just the LP-relaxation of its MILP-relaxation.

Observation 4.1. *The notions introduced so far could trivially be extended by considering an MINLP instead of just an MILP* (4.6) - (4.8)*. Also, what is said in Sections 4.1.1 - 4.1.2 and parts of Section 4.3 remain true when allowing for underlying MINLPs. However, the derivation of cutting planes in Section 4.2 is restricted to the context of MILPs, and the extension of MILP cutting planes to the case in which nonlinear functions are involved is not trivial in general, see, e.g., [Bon11].*

In the next two sections, we provide some examples of MILPs in which non-contiguous domains of certain real-valued split vectors, and thus the validity of real-valued split disjunctions, that are no ordinary splits, can be certified. In most cases, we end up with simple wide splits, that is, holes in the domains of (integer) variables. A conceptual difference will appear to be the fact whether these holes are deduced from primal or dual information.

4.1.1 Certifying split validity by primal information

We first give examples in which non-contiguous domains in the form of holes in the domains of variables are certified explicitly by the constraints present in an underlying MILP. This happens when the modeling tricks, that we called a detour in the introduction, are applied. In fact, such representations are always based on the introduction of auxiliary variables. We present two modalities of this phenomenon, the first one involving big-M constraints. The second one involves so-called GUB-link constraints, that are precisely the constraint structure that we have seen in (4.1) - (4.3).

4.1.1.1 Big-M constraints

The condition that a variable y has to lie inside a non-contiguous domain,

$$y \in \bigcup_{j=1}^{L} [l^j, u^j] \tag{4.14}$$

with $u^{j-1} < l^j$ can be modeled in a mixed integer setting by introducing a binary variable x_j for each of the intervals and imposing big-M constraints, cf. Section 3.1,

$$\begin{aligned} l^j - y &\leq (1 - x_j) \cdot (l^j - l^1) \qquad \forall\, j \in [L] \\ y - u^j &\leq (1 - x_j) \cdot (u^L - u^j) \qquad \forall\, j \in [L] \\ \sum_{j=1}^{L} x_j &= 1. \end{aligned}$$

In any feasible solution, exactly one of the binary variables is equal to one. This will impose the bounds of exactly one of the intervals on the variable y, while all other constraints become redundant. In any case, any feasible solution has to satisfy the simple real-valued split disjunctions

$$y \leq u^{j-1} \vee y \geq l^j, \quad j = 2, \ldots, L. \tag{4.15}$$

Observation 4.2. *If y is integer constrained, we may assume that all l^j and u^j are integers. If in addition $u^{j-1} + 1 < l^j$ for some j,* (4.15) *contains at least one valid wide split disjunction. Even if $u^{j-1} + 1 = l_j$ for all j, they remain valid disjunctions, in particular they are ordinary simple split disjunctions. However, this case is of less interest for us, because* (4.14) *could be equivalently written as $y \in \mathbb{Z}$, $l^1 \leq y \leq u^L$, which can be readily expressed in an explicit way in any MILP solver. In particular, any such MILP solver is already equipped to perform the algorithmic exploitation of* (4.15) *that we seek in this chapter.*

An example of an MILP where such a constraint structure can be found is the TSPMTW. We have seen an MILP model in for this problem Section 1.5.1, namely (1.32) - (1.37) augmented by (1.39) - (1.42)[2]. This example is perfect for showing how an MILPncsd can arise in the context of MILPs. First of all, as explained above, the big-M constraints (1.39) - (1.42) translate the non-contiguous domains

$$a_i \in \bigcup_{d=1}^{L_i} [l_i^d, u_i^d] \quad \forall\, i \in \mathcal{N} \setminus \{p, q\}. \tag{4.16}$$

Thus we could write an MILPncsd formulation for the TSPMTW by augmenting the system (1.32) - (1.37), (1.39) - (1.42) by (4.16). In such an MILPncsd, clearly, either (1.39) - (1.42) or (4.16) would be redundant. This leads to the following interesting observation. As a matter of fact, the only purpose of the constraints

[2]We have seen in Chapter 3 how the related TSPTW can also be modeled by means of indicator constraints, leading to different MILP reformulations. However, we use here the more traditional approach of directly using big-M constraints in order to not mix several effects.

(1.39) - (1.42) and hence the variables w_{id} is imposing the validity of the non-contiguous domains. Therefore, if we account for them explicitly by (4.16), we can drop the big-M constraints (1.39) - (1.42), including the variables w_{id}, and as a side effect reduce the number of variables in the model significantly. All in all, we get a valid MILPncsd formulation for the TSPMTW:

$$\min \quad \sum_{\substack{(i,j)\in\mathcal{A}\\ i\neq q, j\neq p}} t_{ij}x_{ij} \tag{4.17}$$

$$\text{s.t.} \quad \sum_{j\in\delta_i^+} x_{ij} = 1 \qquad \forall\, i \in \mathcal{N}\setminus\{q\} \tag{4.18}$$

$$\sum_{i\in\delta_j^+} x_{ij} = 1 \qquad \forall\, j \in \mathcal{N}\setminus\{p\} \tag{4.19}$$

$$a_i + p_i + t_{ij} \leq a_j + M_{ij}\cdot(1-x_{ij}) \qquad \forall\,(i,j)\in\mathcal{A} : i \neq q \tag{4.20}$$

$$a_i \in \bigcup_{d=1}^{L_i} [l_i^d, u_i^d] \qquad \forall\, i \in \mathcal{N}\setminus\{p,q\} \tag{4.21}$$

$$x_{ij} \in \{0,1\} \qquad \forall\,(i,j)\in\mathcal{A} : i\neq q, j\neq p. \tag{4.22}$$

4.1.1.2 GUB-links

A GUB-link is characterized by the following pair of mixed integer equations involving a variable y and a series of binary variables x_j and numbers $\varphi_j \in \mathbb{R}$, that we assume to be ordered increasingly,

$$y = \sum_{j=1}^{L} \varphi_j x_j$$
$$\sum_{j=1}^{L} x_j = 1.$$

Here, in any feasible solution, exactly one of the binary variables is equal to one, meaning that y is equal to exactly one of the φ_j. In other words, y has a non-contiguous domain,

$$y \in \bigcup_{j=1}^{L} \{\varphi_j\}. \tag{4.23}$$

Again, this implies that the real-valued split disjunctions

$$y \leq \varphi_{j-1} \vee y \geq \varphi_j, \quad \forall\, j = 2,\ldots,L \tag{4.24}$$

are valid.

Observation 4.3. *Similar to what has been said in Observation 4.2, if y is integer constrained, we may assume that all φ_j are integers. If the φ_j are non-consecutive integers,* (4.24) *are valid wide split disjunctions. In case the φ_j are in fact consecutive, they remain valid, but this case is of less interest for us, because* (4.23) *could also be written as $y \in \mathbb{Z}$, $\varphi_1 \leq y \leq \varphi_L$.*

We now recall an example of an MILP where the above structure can be found, namely the Multi-mode Resource Leveling problem introduced in Section 1.5.2. A MILP formulation has been given in (1.50) - (1.56). One instance of that model has been included in MIPLIB 2010 (namely, instance 30n20b8). It turns out that the preprocessor of CPLEX 12.6.1 is able to fix a series of binary variables in (1.56) to zero. In particular, for each job i we are able to identify a set

$$\Upsilon_i = \bigcup_{j=1}^{L_i} [l_i^j, u_i^j] \subseteq [0, T]$$

with $u_i^{j-1}+1 < l_i^j$, such that $x_{i\ell t} = 0 \ \forall \ \ell \in \mathcal{M}_i$, $t \notin \Upsilon_j$. In other words, job i cannot start at such t and we are able to identify some holes in the domain of the implicitly integer-constrained variables s_i and e_i. Instead of imposing these holes through big-M constraints as done in the previous section, one can just strengthen the GUB-links in the model. Thus, a stronger MILP formulation is

$$\min \quad \sum_{k \in \mathcal{R}} c_k R_k \tag{4.25}$$

$$\text{s.t.} \quad \sum_{t \in \Upsilon_i} \sum_{\ell \in \mathcal{M}_i} x_{i\ell t} = 1 \qquad \forall\, i \in [n] \tag{4.26}$$

$$\sum_{t \in \Upsilon_i} \sum_{\ell \in \mathcal{M}_i} t \cdot x_{i\ell t} = s_i \qquad \forall\, i \in [n] \tag{4.27}$$

$$\sum_{t \in \Upsilon_i} \sum_{\ell \in \mathcal{M}_i} (t + p_{i\ell}) \cdot x_{i\ell t} = e_i \qquad \forall\, i \in [n] \tag{4.28}$$

$$e_i \leq s_j \qquad \forall\, (i,j) \in E \tag{4.29}$$

$$\sum_{i \in [n]} \sum_{\ell \in \mathcal{M}_i} r_{i\ell k} \sum_{\substack{\tau = t - p_{i\ell} + 1 \\ \tau \in \Upsilon_i}}^{t} x_{i\ell\tau} \leq R_k \qquad \forall\, k \in \mathcal{R},\ t \in [0, T] \tag{4.30}$$

$$x_{i\ell t} \in \{0,1\} \qquad \forall\, t \in [0,T],\ \ell \in \mathcal{M}_i,\ \forall\, i \in [n]. \tag{4.31}$$

In any case, we can augment either of the two MILPs (1.50) - (1.56) or (4.25) -

(4.31) by the constraints

$$s_i \in \bigcup_{j=1}^{L_i} [l_i^j, u_i^j] \qquad \forall\ i \in [n] \tag{4.32}$$

$$e_i \in \bigcup_{j=1}^{L_i} [l_i^j + \min_{\ell \in \mathcal{M}_i}\{p_{i\ell}\}, u_i^j + \max_{\ell \in \mathcal{M}_i}\{p_{i\ell}\}] \qquad \forall\ i \in [n], \tag{4.33}$$

in order to get an MILPncsd formulation. A difference to the TSPMTW is that the binary variables involved in the GUB-links appear as well in other constraints, namely (4.30), and we cannot just spare them here in order to reduce the model size. Thus the constraints (4.32) - (4.33) remain redundant constraints.

4.1.2 Certifying split validity by dual information

In the previous section, we deduced the validity of non-contiguous split domains by primal information. More precisely, every (primal) feasible solution was assured to lie in these domains. Now instead, we will allow that non-contiguous domains are imposed, that not every primal feasible solution satisfies. In the following MILP, however, we will see that for every primal feasible solution that is excluded from the feasible region in the resulting MILPncsd, there is at least one other feasible solution with identical objective value. The MILP and the MILPncsd are thus equivalent in the sense that they have the same optimal objective value. Hence, we can characterize these domains to be derived from dual information.

The MILP we are going to analyze here is a formulation for the so-called Lazy Bureaucrat Problem (LBP), and strictly speaking, it is an ILP. The Lazy Bureaucrat Problem can be seen as a lazy counterpart of the classical Knapsack Problem introduced in Section 1.5.3. Similar to therein, we are given a set of items $i \in [n]$ with non-negative profits p_i and non-negative weights w_i, both of which we assume to be integral, and a knapsack with capacity C. The objective is slightly different though. Namely, we want to pack a subset of items into the knapsack such that:

1. the profit of all packed items is minimized,

2. their weight does not exceed the capacity,

3. but adding any non-packed item would exceed it.

The task is thus to find a so-called maximal packing with minimum profit. In [FLS15] are proposed several ILP formulations for the LBP. As therein, we assume that the items are sorted non-decreasingly first according to their weights, then according to

their profits, and define the critical item,

$$i_c = \arg\min\Big\{ i \in [n] \;\Big|\; \sum_{j \le i} w_j > C \Big\}, \tag{4.34}$$

as the first item that exceeds the capacity, assuming that all precedent items are packed as well. The critical weight is $w_c = w_{i_c}$. In [FLS15] is shown that it is sufficient to translate condition 3 from above into a constraint only up to the critical item, and one of the most competing ILP formulations is

$$\min \quad \sum_{i=1}^{n} p_i x_i \tag{4.35}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \le C \tag{4.36}$$

$$\sum_{i=1}^{n} w_i x_i + z \ge C + 1 \tag{4.37}$$

$$z \le w_c - (w_c - w_i)(1 - x_i) \qquad \forall\, i \in [i_c] \tag{4.38}$$

$$x_i \in \{0, 1\} \qquad \forall\, i \in [n] \tag{4.39}$$

$$z \in \mathbb{Z}_+. \tag{4.40}$$

The variable z models the weight of the smallest item left out of the packing, although in the above model, it is not forced to be exactly equal to it. Nevertheless, in some sense, it does satisfy the purpose it has been introduced for, and we state this in the following simple result.

Lemma 4.4. *For every feasible solution of* (4.35) - (4.40)*, there is another feasible solution with the same cost and*

$$z \in \{w_i \mid i \le i_c\}. \tag{4.41}$$

Proof. Let $(\hat{x}, \hat{z})$ be feasible to (4.35) - (4.40) and denote the corresponding packing by $S := \{i \in [n] \mid \hat{x}_i = 1\}$. Further, let $\tilde{i} := \min\{i \in [n] \mid i \notin S\}$. Clearly, $\tilde{i} \le i_c$, and from (4.38) we get $\hat{z} \le w_{\tilde{i}}$. Because increasing the value of $\hat{z}$ does not violate (4.37), $(\hat{x}, w_{\tilde{i}})$ is feasible, satisfies (4.41), and since the objective function (4.35) depends on x only, has the same cost as $(\hat{x}, \hat{z})$. □

Thus, if the weights are non-consecutive integers, imposing the domain of the (integer) variable z to be as in (4.41) is non-redundant, cf. also Observation 4.3. All in all, we can introduce an MILPncsd for the LBP, whose feasible region is contained in the one of (4.35) - (4.40), but whose optimal objective value remains unchanged,

by imposing the holes in the domain of z deriving from (4.41) explicitly,

$$\min \quad \sum_{i=1}^{n} p_i x_i \tag{4.42}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq C \tag{4.43}$$

$$\sum_{i=1}^{n} w_i x_i + z \geq C + 1 \tag{4.44}$$

$$z \leq w_c - (w_c - w_i)(1 - x_i) \qquad \forall\, i \in [i_c] \tag{4.45}$$

$$z \in \bigcup_{i=1}^{i_c} \{w_i\} \tag{4.46}$$

$$x_i \in \{0, 1\} \qquad \forall\, i \in [n] \tag{4.47}$$

$$z \in \mathbb{Z}_+. \tag{4.48}$$

We further examine the extension of the LBP to an analogue problem with several bureaucrats, that is, several knapsacks. We call this problem the Multiple Lazy Bureaucrat Problem (MLBP). In contrast to the setting of the LBP, we are given a set of knapsacks $j \in [K]$, each with a capacity C_j. We want to pack the items into the knapsacks such that

1. the profit of all packed items is minimized,

2. the weight of the items packed into a single knapsack does not exceed its capacity,

3. but adding any non-packed item into any knapsack would exceed it.

The model (4.35) - (4.40) is extendable to the MLBP by introducing binary variables x_{ij} that indicate whether item i is packed into knapsack j, and by defining the critical item i_c and the critical weight w_c as in (4.34), but via the cumulated capacity $\sum_{j \in [K]} C_j$. Then, an MILP formulation for the MLBP is

$$\min \quad \sum_{i=1}^{n} p_i \Big(\sum_{j=1}^{K} x_{ij} \Big) \tag{4.49}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_{ij} \leq C_j \qquad \forall\, j \in [K] \tag{4.50}$$

$$\sum_{i=1}^{n} w_i x_{ij} + z \geq C_j + 1 \qquad \forall\, j \in [K] \tag{4.51}$$

$$\sum_{j=1}^{K} x_{ij} \leq 1 \qquad \forall\, i \in [n] \tag{4.52}$$

$$z \leq w_c - (w_c - w_i)\left(1 - \sum_{j=1}^{K} x_{ij}\right) \qquad \forall\, i \in [i_c] \tag{4.53}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i \in [n],\ j \in [K] \tag{4.54}$$

$$z \in \mathbb{Z}_+. \tag{4.55}$$

Again, one can show that for every feasible solution of (4.49) - (4.55), there exists an equivalent one in which

$$z \in \{w_i \mid i \leq i_c\},$$

and we can build an MILPncsd formulation for the MLBP by augmenting (4.49) - (4.55) with (4.41). Our interest in the MLBP will become clear in Section 4.2.4 and is based on the following observation.

Observation 4.5. *A difference between the models for LBP and MLBP is that in the latter, the variable z, that is the only one involved in the constraints imposing a non-contiguous split domain, appears in K so-called knapsack cover constraints,* (4.51)*, instead of just one,* (4.44)*.*

4.2 Real-valued-split cuts

We now assume the existence of an underlying MILPncsd as in (4.9) - (4.12), and assume that

1. either its MILP-relaxation,

2. or its LP-relaxation

has been solved to the point $\hat{x}$. Then, we know that the real-valued split disjunctions (4.13) are valid, and provided that one of them is violated, i.e. $\pi_l < {\pi^{k_0}}^T \hat{x} < \pi_u$ with $\pi_l = u_{k_0}^{j_0 - 1}$ and $\pi_u = l_{k_0}^{j_0}$ for some k_0, j_0, our aim is to derive a cutting plane that cuts off $\hat{x}$. That is, we want to find a valid inequality $\alpha^T x \geq \beta$, such that $\alpha^T \hat{x} < \beta$. We consider one disjunction at a time, i.e., we can drop the indeces i_0 and j_0 in the remainder of this section.

We denote the feasible region of (4.9) - (4.12) by $\mathcal{F}$, and the feasible region of its LP-relaxation by P, i.e., $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$. Thus, we set

$$P_l^{(\pi,\pi_l)} := \{x \in P \mid \pi^T x \leq \pi_l\}, \quad P_u^{(\pi,\pi_u)} := \{x \in P \mid \pi^T x \geq \pi_u\},$$

and $P^{(\pi,\pi_l,\pi_u)} := \text{conv}(P_l^{(\pi,\pi_l)} \cup P_u^{(\pi,\pi_u)})$. We call any linear inequality that is valid for $P^{(\pi,\pi_l,\pi_u)}$ a *real-valued-split cut*. Since $\mathcal{F} \subseteq P^{(\pi,\pi_l,\pi_u)}$, any real-valued-split cut is valid for (4.9) - (4.12). If the considered split is a wide split, we call the valid inequality *wide split cut*.

In the next two sections, we will show two ways of separating real-valued split cuts, namely, as intersection cuts [Bal71] (possible in case 1 from above), and as lift-and-project cuts [Bon12] (possible also in case 2). Without loss of generality, we will assume that the matrix A has full row rank.

4.2.1 Intersection cuts from real-valued split disjunctions

In this section, we assume that we are given a simplex tableau of P as in (1.7) - (1.8),

$$x_i = \hat{x}_i \ + \sum_{j \in N} r_j^i x_j, \qquad i \in B \tag{4.56}$$

$$x_i \geq 0, \qquad i \in [n], \tag{4.57}$$

where the index set of variables is partitioned into basic and non-basic variables, $[n] = B \ \cup \ N$. In particular, such a tableau is given when the LP-relaxation of (4.9) - (4.12) has been solved to the point $\hat{x}$ by means of the simplex method. In that case, $\forall\ i \in N$ we have $\hat{x}_i = 0$, and thus

$$\pi^T \hat{x} = \sum_{i \in B} \pi_i \hat{x}_i. \tag{4.58}$$

In [ACL05b] an explicit formula for the intersection cut derived from general splits is given. This formula can easily be extended to real-valued splits, and we formally state it here. There are several ways to derive the same valid inequality from split sets (see further down), and the one we use to prove the following result relies on the so-called maximum principle, used for example during the derivation of disjunctive cuts [Bal98]. Therein, explicit formulas for disjunctive cuts, from which the valid inequality in the next result can be recovered as well, are given. We will use similar arguments for proving Proposition 4.12 later on.

Proposition 4.6. *Assume that $\hat{x}$ violates the real-valued split disjunction (π^T, π_l, π_u), i.e.,*

$$\pi_l < \pi^T \hat{x} < \pi_u,$$

and $\forall\ j \in N$, define $f_j := \pi_j + \sum_{i \in B} \pi_i r_j^i$. A valid inequality for $P^{(\pi,\pi_l,\pi_u)}$ is then

given by

$$\sum_{j \in N} \max \left\{ \frac{-f_j}{\pi^T \hat{x} - \pi_l}, \frac{f_j}{\pi_u - \pi^T \hat{x}} \right\} x_j \geq 1. \tag{4.59}$$

Proof. By taking into account the simplex tableau, we can rewrite

$$\begin{aligned} \pi^T x &= \sum_{i \in B} \pi_i x_i + \sum_{j \in N} \pi_j x_j \\ &= \sum_{i \in B} \pi_i \left(\hat{x}_i + \sum_{j \in N} r^i_j x_j \right) + \sum_{j \in N} \pi_j x_j \\ &= \sum_{i \in B} \pi_i \hat{x}_i + \sum_{j \in N} \left(\pi_j + \sum_{i \in B} \pi_i r^i_j \right) x_j. \end{aligned}$$

With the definition of the f_j and (4.58), this gives

$$\pi^T x = \pi^T \hat{x} + \sum_{j \in N} f_j x_j.$$

By using this last equation and rearranging the disjunction $\pi^T x \leq \pi_l \vee \pi^T x \geq \pi_u$, we see that every point in $P^{(\pi, \pi_l, \pi_u)}$ has to satisfy

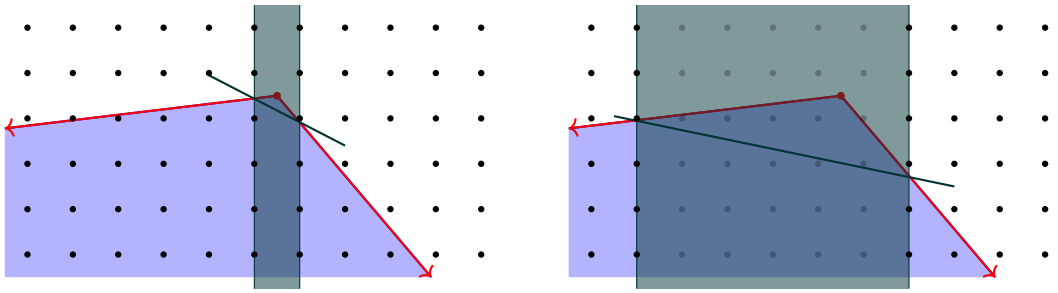
$$\begin{aligned} & \sum_{j \in N} f_j x_j \leq \pi_l - \pi^T \hat{x} \qquad \vee \qquad \sum_{j \in N} f_j x_j \geq \pi_u - \pi^T \hat{x} \\ \iff & \sum_{j \in N} \frac{-f_j}{\pi^T \hat{x} - \pi_l} x_j \geq 1 \qquad \vee \qquad \sum_{j \in N} \frac{f_j}{\pi_u - \pi^T \hat{x}} x_j \geq 1. \end{aligned}$$

The claim follows by applying the maximum principle. □

The valid inequality of Proposition 4.6 is clearly a real-valued-split cut, and it is easy to check that it is violated by $\hat{x}$. We call the inequality the *intersection cut from the real-valued split* (π^T, π_l, π_u), because it can also be derived like an intersection cut from ordinary split sets [Bal71]. That is, if in (1.7) - (1.8) one relaxes the non-negativity on the basic variables, then the resulting set can be shown to be a translated polyhedral cone, often denoted by $P(B)$. Then, (4.59) can as well be obtained by computing the intersection points of the extreme rays of $P(B)$ with the boundary of the real-valued split set

$$S := \{x \in \mathbb{R}^n \mid \pi_l \leq \pi^T x \leq \pi_u\}.$$

Figure 4.2 (a) depicts an example of $P(B)$, a split set and the corresponding ordinary intersection cut in a two-dimensional basic space. Again, if (π^T, π_l, π_u) is a wide split,



(a) Ordinary intersection cut (b) Wide Intersection cut

Figure 4.2: Intersection cuts in the plane

we call (4.59) a *wide intersection cut*.

As anticipated, we are particularly interested in the advantage of using wide split cuts over ordinary split cuts. In the context of ordinary split cuts, whenever the dot product of a given split vector π and $\hat{x}$ is fractional, one can attempt to generate a split cut from the violated ordinary split $(\pi^T, \lfloor \pi^T \hat{x} \rfloor, \lceil \pi^T \hat{x} \rceil)$. If in this situation, there actually is a valid wide split (π^T, π_l, π_u) with $\pi_l \leq \lfloor \pi^T \hat{x} \rfloor$ and $\lceil \pi^T \hat{x} \rceil \leq \pi_u$, that then is automatically violated by $\hat{x}$, a wide split cut will intuitively be at least as good as the ordinary split cut. This is illustrated in Figures 4.2 (a) and (b). In the case of intersection cuts, it is easy to formalize this intuition by means of Definition 1.5. We have the following result.

Proposition 4.7. *Assume that $\hat{x}$ violates the wide split disjunction (π^T, π_l, π_u) (i.e., $\pi_u - \pi_l \geq 1$) and that $\pi^T \hat{x}$ is fractional. Then the corresponding wide intersection cut dominates the intersection cut from the ordinary split $(\pi^T, \lfloor \pi^T \hat{x} \rfloor, \lceil \pi^T \hat{x} \rceil)$.*

Proof. Obviously, (4.59) can also be used to compute the ordinary intersection cut. We further note that the quantities f_j in (4.59) do not depend the right-hand sides of the split disjunction, but only on the simplex tableau and the split vector π. Thus, the two intersection cuts in question only differ in the denominators of their coefficients. Since $\pi_u - \pi_l \geq 1$, $\pi_l \leq \lfloor \pi^T \hat{x} \rfloor$ and $\pi_u \geq \lceil \pi^T \hat{x} \rceil$, which implies that each coefficient of the wide intersection cut is bounded from above by the corresponding coefficient of the ordinary intersection cut. Thus follows the dominance. □

4.2.2 Lift-and-project cuts from real-valued split disjunctions

We now show how to derive valid inequalities from violated real-valued split disjunctions in the fashion of lift-and-project cuts [Bon12]. Lift-and-project cuts are intimately related to the theory of Disjunctive Programming that we treated in Section 1.4. This relation becomes intuitively clear when having a close look at the definition of the set $P^{(\pi,\pi_l,\pi_u)}$ for which we want the cutting planes to be valid. It is,

in fact, the convex hull of the union of two polyhedral sets, which is why Disjunctive Programming techniques are useful here. We state a special case of Theorem 1.12, that we will use further down, in the following corollary.

Corollary 4.8. *The linear inequality $\alpha^T x \geq \beta$ is valid for $P^{(\pi,\pi_l,\pi_u)}$ if and only if there exist $\theta_1, \theta_2 \in \mathbb{R}^m$ and $\tau_1, \tau_2 \in \mathbb{R}_+$ such that*

$$\begin{aligned}
\alpha &\geq A^T\theta_1 - \tau_1\pi \\
\alpha &\geq A^T\theta_2 + \tau_2\pi \\
\beta &\leq b^T\theta_1 - \tau_1\pi_l \\
\beta &\leq b^T\theta_2 + \tau_2\pi_u.
\end{aligned}$$

The corollary can be applied in order to formulate a so-called cut-generating Linear Program, that attempts to find one valid inequality that is violated by $\hat{x}$. Different versions of cut-generating Linear Programs, involving for example different normalization constraints [BP02], have been proposed. In this section, we choose to follow the lines of [Bon12], and we will present the theoretical foundations - mainly extensions of results therein - that serve the derivation of real-valued-split cuts. In the case of ordinary splits, much more detail, especially regarding the relations to other cut-generating Linear Programs and other families of MILP cutting planes, can be found in [Bon12]. Also, the rank-1 lift-and-project cuts that were at the basis of the closures we computed in Section 3.5 were precisely the ones derived [Bon12].

We start by giving a characterization of when an $\hat{x} \in P$, that violates the real-valued split disjunction, is also in $P^{(\pi,\pi_l,\pi_u)}$ or not. For brevity, we denote the inverse of the width of the split by σ_{π_l,π_u}, that is

$$\sigma_{\pi_l,\pi_u} := \frac{1}{(\pi_u - \pi_l)}.$$

Proposition 4.9. *Let $\hat{x} \in P$ and $\pi_l < \pi^T\hat{x} < \pi_u$. Then $\hat{x} \in P^{(\pi,\pi_l,\pi_u)}$ if and only if there is an $y \in \mathbb{R}^n$ such that*

$$\begin{aligned}
\pi^T y - \pi_u \cdot \sigma_{\pi_l,\pi_u}(\pi^T\hat{x} - \pi_l) &\geq 0 \\
Ay = b \cdot \sigma_{\pi_l,\pi_u}(\pi^T\hat{x} - \pi_l) \\
0 \leq y \leq \hat{x}.
\end{aligned}$$

Proof. Let $\tilde{\pi} := \sigma_{\pi_l,\pi_u}\pi$, $\tilde{\pi}_0 := \sigma_{\pi_l,\pi_u}\pi_l$ and $\tilde{\pi}_u := \sigma_{\pi_l,\pi_u}\pi_u$. Note that $\tilde{\pi}_u = \tilde{\pi}_0 + 1$ and that $\hat{x} \in P^{(\pi,\pi_l,\pi_u)}$ is equivalent to $\hat{x} \in P^{(\pi,\tilde{\pi}_0,\tilde{\pi}_0+1)}$. If $(\pi, \tilde{\pi}_0, \tilde{\pi}_0 + 1)$ is an ordinary split, i.e., if all the data is integral, Proposition 1 of [Bon12] states that the latter

is true if and only if there is an $y \in \mathbb{R}^n$ such that

$$\tilde{\pi}^T y - \tilde{\pi}_u \cdot (\tilde{\pi}^T \hat{x} - \tilde{\pi}_0) \geq 0 \tag{4.60}$$

$$Ay = b \cdot (\tilde{\pi}^T \hat{x} - \tilde{\pi}_0) \tag{4.61}$$

$$0 \leq y \leq \hat{x}. \tag{4.62}$$

It is easy to check that Proposition 1 of [Bon12] remains true if the data of the underlying split is real-valued. After dividing (4.60) by σ_{π_l,π_u} and rearranging, the claim follows. □

Proposition 4.9 leads to a Linear Program that can answer the question of whether $\hat{x} \in P$ does or does not lie in $P^{(\pi,\pi_l,\pi_u)}$. In [Bon12], this Linear Program is called the membership LP:

$$\max \quad \pi^T y - \pi_u \cdot \sigma_{\pi_l,\pi_u}(\pi^T \hat{x} - \pi_l) \tag{4.63}$$

$$\text{s.t.} \quad Ay = b \cdot \sigma_{\pi_l,\pi_u}(\pi^T \hat{x} - \pi_l) \tag{4.64}$$

$$0 \leq y \leq \hat{x}. \tag{4.65}$$

The membership LP can be used to answer the aforementioned question in a straightforward way. When the optimal solution value of the membership LP is negative, $\hat{x} \notin P^{(\pi,\pi_l,\pi_u)}$, and we can attempt to find a valid cutting plane that cuts off $\hat{x}$. How to compute this cutting plane is of course a much more interesting question, but it turns out that again the membership LP is of help. In particular, an explicit valid linear inequality that is violated by $\hat{x}$ can be found by going over to the dual of the membership LP, cf. Section 1.1.1. After transforming (4.63) - (4.65) into the form (1.9) - (1.12), its dual can be deduced from (1.13) - (1.16) and is given by

$$\min \quad \lambda^T b \cdot \sigma_{\pi_l,\pi_u}(\pi^T \hat{x} - \pi_l) + \mu^T \hat{x} - \pi_u \cdot \sigma_{\pi_l,\pi_u}(\pi^T \hat{x} - \pi_l) \tag{4.66}$$

$$\text{s.t.} \quad A^T \lambda + \mu \geq \pi \tag{4.67}$$

$$\lambda \in \mathbb{R}^m \tag{4.68}$$

$$\mu \in \mathbb{R}^n_+. \tag{4.69}$$

If the optimal objective value of the membership LP is negative, we know that there exists a dual feasible solution (λ, μ) (e.g., the optimal one) with negative dual objective value. Then, for any such solution, by rearranging the objective function (4.66), we know that the linear inequality

$$\left(\sigma_{\pi_l,\pi_u}(\lambda^T b - \pi_u)\pi^T + \mu^T\right) x \geq \sigma_{\pi_l,\pi_u}\pi_l(\lambda^T b - \pi_u) \tag{4.70}$$

is violated by $\hat{x}$. We call this inequality *lift-and-project cut from the real-valued split* (π, π_l, π_u), and in case (π^T, π_l, π_u) is a wide split, *wide lift-and-project cut*. So far, we have not shown that it is actually valid for $P^{(\pi,\pi_l,\pi_u)}$. In [Bon12], this is done by showing that the dual of the membership LP, (4.66) - (4.69), is equivalent to a certain cut generating Linear Program, provided that its optimal solution value is non-positive. We will now prove the validity of the linear inequality (4.70) by going the somewhat opposite direction. That is, we will start with the explicit formula (4.70) and show its validity by application of Corollary 4.8. The interest of the proof is of course that we conduct it for real-valued splits, but it can also be seen as an alternative proof of the result in [Bon12].

Proposition 4.10. *Assume that* (λ, μ) *is a solution of* (4.66) - (4.69) *with non-positive objective value. Then* (4.70) *is valid for* $P^{(\pi,\pi_l,\pi_u)}$.

Proof. We start by defining $\tau_1 := -\sigma_{\pi_l,\pi_u}(\lambda^T b - \pi_u)$ and $\tau_2 := 1 - \tau_1 = \sigma_{\pi_l,\pi_u}(\lambda^T b - \pi_l)$. Also, let $\theta_1 \in \mathbb{R}^m$, and set $\theta_2 := \theta_1 - \lambda$. With that, we define

$$\alpha := A^T\theta_1 + \mu - \tau_1\pi \quad \text{and} \quad \beta := b^T\theta_1 - \tau_1\pi_l.$$

Next, we show that for any $x \in P$,

$$\left(\sigma_{\pi_l,\pi_u}(\lambda^T b - \pi_u)\pi^T + \mu^T\right) x - \sigma_{\pi_l,\pi_u}\pi_l(\lambda^T b - \pi_u) = \alpha^T x - \beta. \tag{4.71}$$

This can be done by rearranging and noting that for any such x, $Ax - b = 0$:

$$\begin{aligned}
&\left(\sigma_{\pi_l,\pi_u}(\lambda^T b - \pi_u)\pi^T + \mu^T\right) x - \sigma_{\pi_l,\pi_u}\pi_l(\lambda^T b - \pi_u) \\
&\quad = \theta_1^T(Ax - b) + \mu^T x + \sigma_{\pi_l,\pi_u}(\lambda^T b - \pi_u)\pi^T x - \sigma_{\pi_l,\pi_u}\pi_l(\lambda^T b - \pi_u) \\
&\quad = (\theta_1^T A + \mu^T - \tau_1\pi^T)x - b^T\theta_1 + \tau_1\pi_l = \alpha^T x - \beta.
\end{aligned}$$

Thus, if we show that $\alpha^T x \geq \beta$ is valid for $P^{(\pi,\pi_l,\pi_u)}$, so is (4.70). We will do that by Corollary 4.8. Therefore, note that,

$$\alpha = A^T\theta_1 + \mu - \tau_1\pi = A^T\theta_2 + A^T\lambda + \mu - \pi + \tau_2\pi \geq A^T\theta_2 + \tau_2\pi, \tag{4.72}$$

where the last inequality follows from (4.67). Also,

$$\begin{aligned}
\beta = b^T\theta_1 - \tau_1\pi_l &= b^T\theta_2 + b^T\lambda + \sigma_{\pi_l,\pi_u}(\lambda^T b - \pi_u)\pi_l \\
&= b^T\theta_2 + b^T\lambda(1 + \sigma_{\pi_l,\pi_u}\pi_l) - \sigma_{\pi_l,\pi_u}\pi_u\pi_l \\
&= b^T\theta_2 + b^T\lambda\sigma_{\pi_l,\pi_u}\pi_u - \sigma_{\pi_l,\pi_u}\pi_u\pi_l \\
&= b^T\theta_2 + \sigma_{\pi_l,\pi_u}(b^T\lambda - \pi_l)\pi_u = b^T\theta_2 + \tau_2\pi_u.
\end{aligned}$$

Altogether, since μ is non-negative, α and β satisfy the condition of Corollary 4.8, provided we show that also τ_1 and τ_2 are non-negative. Because $\hat{x} \in P$, we know from (4.71) and by hypothesis, that $\alpha^T \hat{x} - \beta \leq 0$. That is,

$$0 \geq (\theta_1^T A + \mu^T - \tau_1 \pi^T)\hat{x} - b^T \theta_1 + \tau_1 \pi_l = \mu^T \hat{x} - \tau_1(\pi^T \hat{x} - \pi_l).$$

Since $\mu^T \hat{x} \geq 0$ and $\pi^T \hat{x} - \pi_l > 0$, this implies $\tau_1 \geq 0$. By using (4.72) and the other representation of β, the same can be deduced for τ_2 and we are done. □

4.2.3 Strengthening intersection cuts

Ordinary intersection cuts from split disjunctions, contained as a special case in the theory presented in Section 4.2.1, are derived by taking into account the integrality requirements of the basic variables. In fact, the split violation is determined by the basic variables only, cf. (4.58). Intuitively, deriving a cutting plane by taking into account the integrality requirements of the non-basic variables as well should result in stronger cutting planes. This idea leads to the integer strengthening principle of intersection cuts from split disjunctions, outlined for example in [ACL05a]. It can also be recovered from the concept of monoidal strengthening introduced in [BJ80]. As well in the case of ordinary lift-and-project cuts there exists theory that allows to strengthen the cutting planes [BCC96, BJ80]. The mentioned strengthening of ordinary intersection cuts can be derived by modifying the underlying split disjunctions in a certain way, and in the following we exploit an analogous approach in order to strengthen intersection cuts from real-valued split disjunctions. Starting from (4.13), we want to find (minimal) $\gamma_l, \gamma_u \in \mathbb{Z}_+$ such that $\forall\, \xi \in \mathbb{Z}$, the disjunction

$$\pi^T x + \gamma_l \xi \leq \pi_l \vee \pi^T x + \gamma_u \xi \geq \pi_u \tag{4.73}$$

is still valid. We will see further down that the introduction of the degree of freedom represented by ξ allows to strengthen the intersection cut. Roughly speaking, we can choose the best ξ in order to reduce the coefficients of the intersection cut for each integer-constrained non-basic variable separately. However, we have to assure that the modified disjunction (4.73) remains valid.

For an ordinary split, $\gamma_l = \gamma_u = 1$ does the job and gives rise to the aforementioned well-studied integer strengthening principle. In our more general case, this is not necessarily true, i.e., such disjunctions are not necessarily valid. Intuitively, we cannot shift the dot product of the underlying split vector with $\hat{x}$ by an arbitrary integer without leaving a valid hole. What we can do is shift it by integer multiples of the distance from π_l and π_u to the upper and lower bounds of the expression $\pi^T x$, respectively. In other words, we have to shift at least by a quantity that will

push everything out of the feasible region. This is intuitively weaker than integer strengthening for ordinary splits, where we can shift unitarily, especially if π_l and π_u are far away from these bounds. Nevertheless, we assume in the following that bounds of $\pi^T x$ are known,

$$\underline{\pi} \leq \pi^T x \leq \overline{\pi}. \tag{4.74}$$

In (4.9) - (4.12) for example, such bounds for the k-th split vector π^k are given by l_k^1 and $u_k^{L_k}$. We then have the following Lemma.

Lemma 4.11. *Assume that* (4.74) *holds. Then, with* $\gamma_l = \overline{\pi} - \pi_l$ *and* $\gamma_u = \pi_u - \underline{\pi}$, *the disjunction*

$$\pi^T x + \gamma_l \xi \leq \pi_l \vee \pi^T x + \gamma_u \xi \geq \pi_u \tag{4.75}$$

is valid $\forall\ \xi \in \mathbb{Z}$.

Proof. Without loss of generality, we may assume $\gamma_l, \gamma_u \geq 0$. Otherwise, π_l and π_u may always be modified such that this holds. For $\xi = 0$, there is nothing to show. If $\xi \geq 1$, for the left-hand-side of the right term of disjunction (4.75), since $\pi^T x \geq \underline{\pi}$ and $(\pi_u - \underline{\pi}) \geq 0$, we have

$$\pi^T x + \gamma_u \xi = \pi^T x + (\pi_u - \underline{\pi})\xi \geq \underline{\pi} + (\pi_u - \underline{\pi}) = \pi_u.$$

Therefore, the right term of (4.75) is always fulfilled. In an analogue fashion, one can show that if $\xi \leq -1$, the first term of (4.75) is alway fulfilled. The claim follows. □

With the above Lemma, we can prove the validity of a *strengthened intersection cut from the real-valued split* (π^T, π_l, π_u).

Proposition 4.12. *Consider the setting of Proposition 4.6, and denote the set of integer-constrained non-basic variables by* $J := N \cap \{n-p+1, \ldots, n\}$. *If we define*

$$\hat{\pi}_j := \frac{-f_j(\pi_u - \pi_l)}{(\pi_u - \underline{\pi})(\pi^T \hat{x} - \pi_l) + (\overline{\pi} - \pi_l)(\pi_u - \pi^T \hat{x})},$$

then the inequality

$$\sum_{j \in J} \min\left\{ \frac{-f_j - (\overline{\pi} - \pi_l)\lfloor \hat{\pi}_j \rfloor}{\pi^T \hat{x} - \pi_l}, \frac{f_j + (\pi_u - \underline{\pi})\lceil \hat{\pi}_j \rceil}{\pi_u - \pi^T \hat{x}} \right\} x_j + \sum_{j \in N \setminus J} \max\left\{ \frac{-f_j}{\pi^t \hat{x} - \pi_l}, \frac{f_j}{\pi_u - \pi^T \hat{x}} \right\} x_j \geq 1$$

is valid.

Proof. Consider an arbitrary split vector $\tilde{\pi}$ with $\tilde{\pi}_j = 0 \; \forall \; j \notin J$. Clearly, $\tilde{\pi}^T x \in \mathbb{Z}$. Thus, by Lemma 4.11,

$$\pi^T x + (\overline{\pi} - \pi_l)\tilde{\pi}^T x \leq \pi_l \;\vee\; \pi^T x + (\pi_u - \underline{\pi})\tilde{\pi}^T x \geq \pi_u \tag{4.76}$$

is a valid disjunction. Proceeding as in the proof of Proposition 4.6, (4.76) can be written as

$$\sum_{j \in N} \frac{-f_j}{\pi^T \hat{x} - \pi_l} x_j - \frac{\overline{\pi} - \pi_l}{\pi^T \hat{x} - \pi_l} \tilde{\pi}^T x \geq 1 \;\vee\; \sum_{j \in N} \frac{f_j}{\pi_u - \pi^T \hat{x}} x_j + \frac{\pi_u - \underline{\pi}}{\pi_u - \pi^T \hat{x}} \tilde{\pi}^T x \geq 1.$$

Writing $\tilde{\pi}^T x = \sum_{j \in J} \tilde{\pi}_j x_j$, the latter can be rearranged to

$$\sum_{j \in J} \frac{-f_j - (\overline{\pi} - \pi_l)\tilde{\pi}_j}{\pi^T \hat{x} - \pi_l} x_j + \sum_{j \in N \setminus J} \frac{-f_j}{\pi^T \hat{x} - \pi_l} x_j \geq 1 \quad \vee$$
$$\sum_{j \in J} \frac{f_j + (\pi_u - \underline{\pi})\tilde{\pi}_j}{\pi_u - \pi^T \hat{x}} x_j + \sum_{j \in N \setminus J} \frac{f_j}{\pi_u - \pi^T \hat{x}} x_j \geq 1.$$

Applying the maximum principle gives the valid inequality

$$\sum_{j \in J} \max \left\{ \frac{-f_j - (\overline{\pi} - \pi_l)\tilde{\pi}_j}{\pi^T \hat{x} - \pi_l}, \frac{f_j + (\pi_u - \underline{\pi})\tilde{\pi}_j}{\pi_u - \pi^T \hat{x}} \right\} x_j$$
$$+ \sum_{j \in N \setminus J} \max \left\{ \frac{-f_j}{\pi^T \hat{x} - \pi_l}, \frac{f_j}{\pi_u - \pi^T \hat{x}} \right\} x_j \geq 1.$$

For each $j \in J$, we minimize the coefficient of the above inequality over $\tilde{\pi}_j \in \mathbb{Z}$ independently. Therefore, let the two functions $u_j(\cdot)$ and $v_j(\cdot)$ be defined as the two terms appearing in the coefficient for $j \in J$, i.e.

$$u_j(\tilde{\pi}_j) := \frac{-f_j - (\overline{\pi} - \pi_l)\tilde{\pi}_j}{\pi^T \hat{x} - \pi_l}$$
$$v_j(\tilde{\pi}_j) := \frac{f_j + (\pi_u - \underline{\pi})\tilde{\pi}_j}{\pi_u - \pi^T \hat{x}}.$$

One can check that $u_j(\cdot)$ and $v_j(\cdot)$ are monotonically decreasing and increasing, respectively, and that $u_j(\hat{\pi}_j) = v_j(\hat{\pi}_j)$. Thus, $\hat{\pi}_j$ minimizes $\max\{u_j(\tilde{\pi}_j), v_j(\tilde{\pi}_j)\}$ over $\mathbb{R}$. The claim follows by going to the integer arguments adjacent to $\hat{\pi}_j$. □

As anticipated, the integer strengthening principle underlying Proposition 4.12 is intuitively weaker than the one for ordinary intersection cuts. While we could prove for example a dominance relation between a wide intersection cut and corresponding ordinary intersection cuts in Proposition 4.7, this is not true for the integer-

strengthened versions. For example, in the case of an intersection cut from a simple ordinary split, the strengthened cut becomes precisely the GMI cut introduced in Definition 1.4. One can show that all coefficients of a GMI cut lie in $[0, 1]$, see, e.g., [ACL05a], but there seems to be no analogue property for strengthened wide intersection cuts. This can be seen as some kind of dilemma. Whenever we have a violated wide split disjunction, it is not clear whether the best strategy is to separate a wide intersection cut that can then be strengthened weakly, or to weaken the disjunction to an ordinary split and separate an ordinary intersection cut, that can then be strengthened strongly to a GMI cut. We will analyze this dilemma, that is somewhat related to the work in [BQ10], computationally in Section 4.2.4.1. Also, we make some considerations on how to improve on the tool provided by Lemma 4.11 in the following section.

4.2.3.1 Thoughts on regularly distributed non-contiguities

We return to the question of how to shift the real-valued split in order to get disjunctions that are still valid. Are there situations in which we can shift it by a quantity less than the one that pushes everything outside of the feasible region? It seems that we can answer this question positively if we assume regularly distributed non-contiguities in the domain. We therefore consider a split vector π, for simplicity assumed to be an integral vector giving rise to wide splits, whose domain is a subset of

$$\bigcup_{\xi \in \mathbb{Z}} [l + s\xi, u + s\xi], \tag{4.77}$$

where l, u and $s > u - l$ are fixed parameters. Basically, two adjacent intervals have the same width and are just shifted with respect to each other by the quantity s. In this situation Lemma 4.11 holds with $\pi_l = u$, $\pi_u = l + s$ and $\gamma_l = \gamma_u = s$, and Proposition 4.12 can be extended accordingly. The smaller s, the stronger the integer strengthening that leads to the cut in Proposition 4.12 should be. Instead of shifting the disjunction out of the feasible region, we can just shift it by one interval.

Example 4.13. *Taking the special case of a simple split, we consider a variable y with domain $[0, 1] \cup [3, 4] \cup [12, 13]$, i.e., having holes $[1, 3]$ and $[4, 12]$. Implied by these holes (and the bounds $0 \leq y \leq 13$) are the regularly distributed ones $[1, 3]$, $[4, 6]$, $[7, 9]$ and $[10, 12]$. The domain of y is a subset of $\cup_{\xi \in \mathbb{Z}} [3\xi, 1 + 3\xi]$.*

Continuing with the special case of simple splits, we examine how a variable y with a domain like (4.77) can be modeled in a mixed integer setting:

$$y = l + s\xi + z \tag{4.78}$$

$$\xi \in \mathbb{Z} \tag{4.79}$$

$$z \in [0, u - l]. \tag{4.80}$$

Further specializing, we go over to the case in which the intervals are actually single points ($l = u$, or equivalently $z = 0$). The variable y is then just a translated multiple of an integer variable[3], and (4.78) becomes:

$$y = l + s\xi. \tag{4.81}$$

Basically, y is allowed to take only isolated values that are distributed periodically. In this case, i.e., when we have (4.81), there is some doubt on the practical relevance of a (strengthened or non-strengthened) wide intersection cut derived from y. Whenever y violates a valid disjunction arising from the holes in its domain, ξ violates the integrality requirement. In addition, it is likely that whenever y is basic, so is ξ and vice versa. Therefore, the two intersection cuts that could be separated might be somewhat equivalent. However, we did not test the separation of simple real-valued split cuts from variables that are multiples of others computationally. In any case, it is not clear whether the above doubts on the practical relevance of wide intersection cuts also come up when we have either a non-simple split, or $l < u$, i.e., a third variable z is involved in (4.78), or even both. Even more, these doubts are potentially justified only if the regularly distributed non-contiguities are modeled explicitly via (4.78) - (4.80) or (4.79)/(4.81). If this is not the case and the hole information comes from dual considerations as in Section 4.1.2, an integer strengthening procedure that is based on regularities as in (4.77) is certainly interesting.

4.2.4 Computation

We now present the results of some computational experiments with real-valued-split cuts applied to several MILPncsd's. In particular, we use instances of the problems presented in Sections 4.1.1 and 4.1.2 as test instances. All problem instances here are build from integer data, and thus all continuous variables therein are implicitly integer constrained variables and we include their indices in $\{n - p + 1, \ldots, n\}$. Also, we tested on MIPLIB 2010 instances with randomly created non-contiguities in the domains of integer-constrained variables. Hence, all splits in the considered MILPncsd's are simple wide splits. That is, the fact of a split disjunction being violated is equivalent to saying that an integer variable lies in a hole. As has been said right before Proposition 4.7, we are interested in testing the computational

[3]For example, for $s = 2$ we get a variable that takes either only even or only odd values.

Algorithm 4.1: r rounds of intersection cuts

```
solve the LP-relaxation of (4.9) - (4.12), and set k = 1;
do
    let the optimal simplex tableau be given as in (4.56) - (4.57);
    for i ∈ B ∩ {n − p + 1, . . . , n} do
        if b_1 AND x̂_i lies in a hole then
            compute the corresponding wide intersection cut, if possible, and
            add it to the LP-relaxation;
        else if b_2 AND x̂_i is fractional then
            compute the intersection cut corresponding to the ordinary simple
            split, if possible, and add it to the LP-relaxation;
        end
    end
    resolve the LP-relaxation;
    k = k + 1;
while (k < r AND cuts added);
```

advantage of wide split cuts over corresponding ordinary split cuts. Therefore, we include the latter in our experiments. However, we do not only include such cuts corresponding to split disjunctions with the same split vectors π[4] that appear in the underlying MILPncsd, but also simple split cuts corresponding to all integer and binary variables in the model. This is motivated by an observation that can be made in the context of Section 4.1.1. In particular, when non-contiguous split domains are modeled explicitly, part of the wide split disjunction is translated into ordinary split disjunctions involving the auxiliary binary variables. In such a case, we want to analyze the advantage of separating cutting planes from wide split disjunctions in addition to ordinary split cuts related to these binary variables.

All in all, we can say that we only generate simple wide split or ordinary simple split cuts in the following. The different cut generation procedures that we use can be described as in Algorithms 4.1 and 4.2, that are more elaborate versions of an approach as in Algorithm 5. In the former, we generate intersection cuts, while the latter is designed for lift-and-project cuts. We never mix the separation of intersection cuts and lift-and-project cuts into a single procedure. For ease of exposition, they are formulated explicitly for simple wide split disjunctions, i.e., holes in the domains of integer variables. It should be clear though that both can be extended easily to the separation of split cuts from arbitrary real-valued splits. Algorithm 4.2 is a direct adaption of the procedure proposed in [Bon12]. It is slightly more elaborate than Algorithm 4.1 in the sense that from one iteration to another, only those variables, that led to a cut in the previous iteration, are considered,

[4] more precisely, $\mathbf{e}_i$ in the case of simple splits

Algorithm 4.2: iterated lift-and-project cuts

```
1  initialize the membership LP with the data A and b;
2  solve the LP-relaxation of (4.9) - (4.12), and set k = 1, abort = false;
3  do
4  |  set K = {n − p + 1, ..., n}, reinit = true;
5  |  do
6  |  |  set I_1 = I_2 = ∅;
7  |  |  for i ∈ K do
8  |  |  |  if b_1 AND x̂_i lies in a hole then
9  |  |  |  |  I_1 = I_1 ∪ {i};
10 |  |  |  else if b_2 AND x̂_i is fractional then
11 |  |  |  |  I_2 = I_2 ∪ {i};
12 |  |  |  end
13 |  |  end
14 |  |  set K = ∅;
15 |  |  if k ≤ r then
16 |  |  |  for i ∈ I_1 do
17 |  |  |  |  compute the corresponding wide lift-and-project cut by solving
   |  |  |  |  the membership LP, if possible, add it to the LP-relaxation
   |  |  |  |  and set K = K ∪ {i};
18 |  |  |  end
19 |  |  |  for i ∈ I_2 do
20 |  |  |  |  compute the corresponding ordinary lift-and-project cut by
   |  |  |  |  solving the membership LP, if possible, add it to the
   |  |  |  |  LP-relaxation and set K = K ∪ {i};
21 |  |  |  end
22 |  |  |  k = k + 1;
23 |  |  else
24 |  |  |  abort = true;
25 |  |  end
26 |  |  resolve the LP-relaxation;
27 |  while K ≠ ∅;
28 |  if reinit then abort = true;
29 while !abort;
```

possibly reinitializing the whole list when no cut was separated[5]. Both algorithms contain a limit r on the number of iterations of cut separation and two boolean variables b_1 and b_2 as input parameters. Different combinations of the latter help to describe the different cut generation strategies we apply. b_1 indicates that we do separate simple wide split cuts, while b_2 indicates whether we do separate ordinary simple split cuts. We use the following denomination for the three different strategies

[5]Restricting to the separation of (strengthened) ordinary lift-and-project cuts, Algorithm 4.2 is precisely the one that has been used for computing the closures P_e and P_e^* in Section 3.5.

we consider.

- `w/o` (without wide splits): For each fractional binary or integer variable, we compute a split cut from an ordinary split, if possible.
- `w` (with wide splits): For each integer variable, we compute a wide split cut if its value lies in a hole, if possible, or otherwise a split cut from an ordinary split, if possible. In addition, for each fractional binary variable, we compute a split cut from an ordinary split, if possible.
- `o` (only wide splits): For each integer variable, we compute a wide split cut if its value lies in a hole, if possible.

In the case of intersection cuts, a separation is only possible, if the corresponding variable is basic, while for lift-and-project cuts, the negativity of the optimal objective value of the membership LP is required[6]. The correspondence of the three strategies to the parameters b_1 and b_2 is subsumed in the following table.

b_2 \ b_1	true	false
true	w	w/o
false	o	-

It turned out that the separation of wide split cuts with either of the two algorithms above in the context of the TSPMTW does not augment the dual bound, neither on the MILPncsd formulation (4.17) - (4.22), nor the one where the big-M constraints (1.39) - (1.42) are included as redundant constraints. A reason might be that the TSPMTW can be seen as a combination of a TSP component and a scheduling component. The TSP part alone makes the problem already quite hard, and the wide split cuts applied to our MILPncsd model act on its scheduling part of the model only, thus not being able to augment the dual gap. We exclude the TSPMTW from the computational results in the following sections, but will get back to it in Section 4.3.1. Algorithms 4.1 and 4.2 have been coded in C/C++, and as LP solver we use CPLEX 12.6.1 through its C API. Throughout the following sections we mostly report the percentage of the initial dual gap that is closed by the separation of cutting planes through either of Algorithms 4.1 and 4.2 when imposing some iteration limt r. That is, if δ_{bef} denotes the dual gap of the initial LP-relaxation, and δ_{aft} the one after termination of the algorithm, the column % gap closed in the following tables shows $(\delta_{bef} - \delta_{aft})/\delta_{bef}$ (as a percentage). Also, the total number of cuts that have been generated is sometimes shown.

[6]In addition, a cut might not be separated due to numerical issues in either case.

4.2.4.1 The Lazy Bureaucrat Problem

In order to create instances of the MILPncsd model (4.42) - (4.48) for the LBP we used the knapsack instance generator presented in [MPT99] and available at [PIS]. This leads to instances with n items with profits and weights in the range $[0, R]$, and whose correlation is determined by the membership to one of nine different classes. For each class $q \in [9]$, we generated one instance for each combination of values

$$n \in \{10, 20, 30, 40, 50, 100, 500, 1000, 2000\}$$
$$R \in \{1000, 10000\}$$
$$C \in \{\lfloor 0.25W \rfloor, \lfloor 0.5W \rfloor, \lfloor 0.75W \rfloor\},$$

where W is the cumulated weight $\sum_{i \in [n]} w_i$, resulting in 54 instances for each class[7]. Each time we apply Algorithms 4.1 or 4.2 with all three strategies `w/o`, `w` and `o` to a set of instances, this set can be divided into four disjunct subsets, or groups:

- group 1: instances that result in a significant difference between strategies `w/o` and `w` in terms of the initial dual gap closure (say, at least one percent)
- group 2: those of the remaining instances that result in a significant relative difference between strategies `w/o` and `w` in terms of the number of cuts that are separated (say, at least ten percent)
- group 3: those of the remaining instances that result in a positive dual gap closure when applying strategy `o`
- group 4: the remaining instances, i.e., those that do not result in any significant difference between strategies `w/o` and `w`, and wide split cuts alone are not able to close any dual gap (including instances in which neither ordinary split cuts nor wide split cuts have any effect)

For each of the nine classes, Table 4.1 shows the number of the instances out of the 54[8] generated ones, that did not result in numerical difficulties (column "total") and their division into the different groups after the separation of (wide) intersection cuts through Algorithm 4.1 with an iteration limit of $r = 10$. Not all classes seem to be interesting from our point of view. In fact, it often happens that strategies `w` and `w/o` perform essentially identical. However, this is not true for the classes 4, 6 and 7. Therefore, in what follows we restrict to class 4.

[7] When $q = 9$, instances with $R = 1000$ are almost always infeasible, and therefore only the 27 instances with $R = 10000$ are considered.

[8] 27 for class 9

Tables 4.2 - 4.4 show the results of the above application of Algorithm 4.1 on class 4, divided by groups, in more detail. Table 4.2 shows the instances in which we can close significantly more gap with strategy `w` than with `w/o`, i.e., group 1. That means that the separation of wide intersection cuts on top of ordinary intersection cuts is highly advantageous. Interestingly, on these instances the separation of only wide intersection cuts (strategy `o`) already leads to a gap closure that is significantly greater than the one achieved with strategy `w/o`, and almost reaches the one of strategy `w`. Another side effect of the separation of wide intersection cuts becomes eminent in Table 4.2, namely the reduction of the number of cuts that are separated in total. Apart from closing more gap, strategy `w` also decreases this number significantly with respect to strategy `w/o`. This is a desirable effect since the number of constraints in an LP, to which separated cutting planes have to be added in order to benefit from the additional dual gap closure, influence the computational effort when solving the latter. Remarkably, the significant gap closure achieved by strategy `o` requires a very limited number of separated cuts. The side effect concerning the number of cuts also persists throughout Table 4.3, where those instances are shown that do not benefit significantly from the separation of wide intersection cuts in terms of the gap closure, i.e. group 2. Again, strategy `o` is highly competitive and requires a very limited number of cuts in total to achieve almost the same gap closure as strategies `w/o` and `w`. In Table 4.4 we see those instances that do not result in any significant difference between strategies `w/o` and `w`, but still wide intersection cuts alone are able to close a positive amount of the initial dual gap, that is, group 3.

Tables 4.5 - 4.7 show analogue results for Algorithm 4.2 with $r = 10$. The tendencies regarding the differences between the three strategies observed in the case of wide intersection cuts mostly persist: wide lift-and-project cuts can significantly increase the gap closure in various instances, when separated on top of ordinary lift-and-project cuts as well as when separated alone. A slight difference with respect to wide intersection cuts consists in the number of cuts that are separated. Although this number is decreased on average by wide lift-and-project cuts throughout Tables 4.5 - 4.7, there is a non-negligible number of instances in which strategy `w` increases the total number of cuts, especially in Table 4.6. In the remainder of this and in the next two sections, we restrict to showing the results of the separation of wide intersection cuts, underlining that for wide lift-and-project cuts we obtained results that exhibited the same characteristics that we just outlined.

We also conducted experiments with the MLBP, for which an MILPncsd model is given by (4.49) - (4.55) augmented by (4.46). An MLBP instance can be created from an LPB instance by, for example, dividing the knapsack capacity C into K smaller, equal knapsacks. In particular, for even C, we set $C_j = 2 \cdot \lfloor \frac{C}{2} \rfloor$ for all $j \in [K]$

in the following, and for odd C we set $C_j = 2 \cdot \lfloor \frac{C+1}{2} \rfloor - 1$. The weights and profits remain unchanged. In this way, we create instances based the 25 LBP instances assigned to group 1 through the separation of wide intersection cuts above, i.e., those in Table 4.2, for $K \in \{2, 5, 10\}$. Table 4.8 shows the results of the separation of $r = 10$ rounds of intersection cuts (Algorithm 4.1) through strategies `w/o` and `w` on all of these instances. Also, the same results in case of the LBP, i.e., columns two and three of Table 4.2, are shown. In addition, in each case the column Δ shows the relative difference of the gap closures achieved with the two strategies. I.e., if $\phi_{\texttt{w/o}}$ denotes the gap closure achieved by strategy `w/o`, and $\phi_{\texttt{w}}$ the one achieved by strategy `w`,

$$\Delta = (\phi_{\texttt{w}} - \phi_{\texttt{w/o}})/\phi_{\texttt{w/o}}.$$

The first observation we can make is that the problem gets more difficult with increasing K. In fact, the gap closure of either strategy deteriorates. The second observation is that the relative advantage of the strategy that exploits wide intersection cuts increases. Although there are single instances with a negative Δ, i.e., `w/o` is the winning strategy, on average Δ increases significantly with increasing K. Together with Observation 4.5, this can be seen as computational evidence of the strength of wide intersection cuts. In fact, with increasing K the variable z plays a more important role in the model, which is why the separation of wide intersection cuts, that are separated based on valid disjunctions violated by z, helps relatively more to strengthen the LP-relaxation.

Finally, we turn to some experiments concerning the integer strengthening of wide intersection cuts discussed in Section 4.2.3. We want to analyze the dilemma discussed therein: whenever a variable lies in a hole and is fractional, should we separate the wide intersection cut and strengthen it by Proposition 4.12, knowing that this strengthening is relatively weak, or should we just separate an ordinary intersection cut, knowing that we can perform stronger strengthening and obtain a GMI cut? In order to analyze this dilemma, we introduce two new strategies:

- `w-s` (with strengthened wide splits): Equal to strategy `w`, except that every separated wide intersection cut is strengthened (in particular, for variables without holes, we still separate ordinary intersection cuts, if possible).
- `w/o-g` (with GMI cuts instead of wide splits): equal to `w`, except that for every basic variable that violates a wide split disjunction (in which case with strategy `w` we separate a wide intersection cut), we separate a GMI cut.

Table 4.9 compares the gap closures achieved through $r = 10$ rounds of intersection cuts (Algorithm 4.1) with strategies `w`, `w-s` and `w/o-g` on the instances of group 1 above, i.e., those in Table 4.2. We see that the strengthening of every wide

intersection cut leads to a significant improvement in rare cases, thus improving the average slightly. In any case, strategies `w` and `w-s` both beat `w/o-g`. This means that GMI cuts are not able to reach the gap closure that we can achieve by exploiting the wide split disjunction.

4.2.4.2 Multi-mode Resource Leveling

We further tested our procedures on 9 instances of the Multi-mode Resource Leveling problem, including the aforementioned one, `30n20b8`, that has been included in MIPLIB 2010. The remaining instances are pairs of variants of 4 underlying ones. We use the model (4.25) - (4.31), augmented by (4.32) - (4.33). Table 4.10 shows the results of applying Algorithm 4.1 with $r = 10$, that is, the separation of wide intersection cuts. There is a slight improvement of strategy `w` with respect to `w/o`, mostly due to one instance in which wide intersection cuts can close almost four percent more of the initial dual gap. One might argue that in these instances, the exploitation of the holes is already performed by the ordinary intersection cuts based on the binary variables that translate the holes into explicit constraints, see also the introductory part of Section 4.2.4. However, when looking at strategy `o`, we might also motivate the only marginal improvement of `w` over `w/o` by the fact that the holes do not seem to play a significant role at all in closing the dual gap in these instances. We also note that in terms of the total number of cuts that are separated, we again observe an only slight difference in case wide intersection cuts are separated. The separation of wide lift-and-project cuts leads to similar results on these instances.

To further analyze the issue of explicitly modeled holes, we report on the results obtained with further instances in which the non-contiguous domains are certified by primal information in the next section.

4.2.4.3 MIPLIB 2010 - random holes

Finally, we conducted experiments with randomly created instances. In particular, we took already existing instances from MIPLIB 2010 and randomly generated holes that were distributed in the domains of the involved integer variables. We took all MIPLIB 2010 instances of problem type MIP or IP and problem status *easy*[9]. We then further discarded instances whose variable ranges are either to large ($> 10^9$) or too small (< 2), infeasible instances, feasibility problems, or instances with a single integer variable, ending up with a set of 14 instances that did not result in numerical difficulties. We randomly distributed holes of three different sizes inside the domain of each involved integer variable, where the size of a hole is always relative to the range of the variable. Increasing the maximum number of holes of

[9] as per January 2016

each size, and increasing the probability that a hole is allocated or not, we created non-contiguities with three different intensities $\mu \in \{1, 2, 3\}$. As anticipated, we are interested in demonstrating the computational advantage of wide split cuts even when the wide split disjunctions are translated into ordinary split disjunctions via auxiliary binary variables, that is, when the non-contiguous domains are certified by primal information as in Section 4.1.1. In all of the considered instances, we therefore explicitly added either big-M constraints or GUB-links in order to impose generated the holes, which results in MILPncsd models with redundant constraints. Table 4.11 shows the results of the separation of $r = 10$ rounds of intersection cuts with strategies `w`, `w/o` and `o`[10]. Each cell in Table 4.11 shows the average dual gap closure of either strategy over a number of instances that are derived from the same original MIPLIB instance with the same μ. The number of instances underlying each cell is shown in the columns "#"[11]. On average, we observe a significant advantage of strategy `w` over `w/o`, and also wide split cuts alone are able to close a significant portion of the initial dual gap. Furthermore, the gap closures of strategies `w` and `o` increase with increasing μ on average, confirming the intuition that we can benefit algorithmically by exploiting wide split disjunctions directly, although this is not true in all single cases. We observe various scenarios. For example, it happens that strategies `w` and `w/o` are more or less equivalent, but strategy `o` does not reach the same level of gap closure (instance n4-3), similar to Multi-mode Resource Leveling instances from the previous section. But it also happens that strategy `w` leads to a significant advantage over `w/o`, while `o` can be less (instance lectsched-4-obj) or equally (instance neos-555424) performing. Also, we observe that all three strategies are comparable (instance sp98ir). All in all, we see that the exploitation of wide split disjunctions can be advantageous even when their validity is certified by primal information.

4.3 Exactly solving MILPs with non-contiguous split domains

So far we focused on how to separate wide split cuts for MILPncsd problems and reported on computational results concerning the dual gap closure achieved by these cuts. In other words, we analyzed the algorithmic exploitation of wide split disjunctions in MILP through cutting planes. In this section we are concerned with an experimental algorithm that solves such problems to optimality, exploiting wide

[10] In order to maintain a tractable model size, GUB-links were used to impose the holes in only 5 out of the 14 original MIPLIB instances.

[11] In all cases, we attempted to generate 5 random instances, but in some cases the procedure to generate holes with a given intensity μ fails.

Algorithm 4.3: MILPncsd branch & bound

1 let the root node ρ be the LP-relaxation of (4.9) - (4.12);
2 set $\mathcal{T} = \{\rho\}$ and $UB = \infty$;
3 **while** $\mathcal{T} \neq \emptyset$ **do**
4 choose $\eta \in \mathcal{T}$, set $\mathcal{T} = \mathcal{T} \setminus \{\eta\}$;
5 solve η and denote by $\hat{x}$ and $\hat{z}$ its optimal solution and objective value;
6 **if** $\hat{z} < UB$ **then**
7 **if** $\hat{x}$ *integral* **then**
8 **if** $\hat{x}$ *satisfies the constraints* (4.11) **then**
9 update $UB = \hat{z}$;
10 **else**
11 choose $k \in [K]$ and $2 \leq j \leq L_k$ with $u_k^{j-1} < {\pi^k}^T \hat{x} < l_k^j$;
12 create nodes η^- and η^+ by adding the constraints ${\pi^k}^T x \leq u_k^{j-1}$ and ${\pi^k}^T x \geq l_k^j$, respectively, to η;
13 set $\mathcal{T} = \mathcal{T} \cup \{\eta^-, \eta^+\}$;
14 **end**
15 **else**
16 choose an integer-constrained x_i with fractional value $\hat{x}_i$;
17 create nodes η^- and η^+ by adding the constraints $x_i \leq \lfloor \hat{x}_i \rfloor$ and $x_i \geq \lceil \hat{x}_i \rceil$, respectively, to η;
18 set $\mathcal{T} = \mathcal{T} \cup \{\eta^-, \eta^+\}$;
19 **end**
20 **end**
21 **end**

split disjunctions also in branching. We therefore assume to have an underlying MILPncsd as in (4.9) - (4.12). Our proposal of such an algorithm builds on Algorithm 1.1, the branch & bound for MILP, and is outlined in Algorithm 4.3.

Roughly speaking, Algorithm 4.3 works by initially applying a branch-and-bound algorithm to the MILP-relaxation of an MILPncsd. The difference to Algorithm 1.1 then lies in Lines 8 - 13. Whenever an integer feasible solution is found, we check whether the non-contiguous split domains are satisfied, and if not create two child nodes by imposing the bounds of a violated hole explicitly. One motivation behind the fact that Algorithm 4.3 is built on top of a branch-and-bound algorithm for MILP is that we can exploit MILP as a sophisticated technology, in the sense that we can possibly use an MILP code as a black-box inside an implementation of Algorithm 4.3. A special case in which we can do so is the following.

Example 4.14. *If all split vectors in* (4.9) - (4.12) *are simple wide splits (i.e., there are integer variables with holes in their domains), Algorithm 4.3 can be implemented*

on top of CPLEX using the incumbent[12]- and branch-callback[13]. A direct integration of the usual integer branching of Lines 16 - 17 with the branching on disjunctive constraints as in Lines 11 - 12, also involving continuous variables, would be possible, for example, in SCIP. However, also in that case primal heuristics would have to adapted to satisfying the constraints (4.11).

In any case, we know that the disjunctions (4.13) are valid for the underlying MILPncsd, and we can thus attempt to separate real-valued split cuts whenever an LP has been solved in Line 5 of Algorithm 4.3. In that regard, we make some technical remarks about the two ways of separating wide split cuts that we presented in Sections 4.2.1 and 4.2.2.

Remark 4.15. *If we separate real-valued split cuts at a non-root node η in Algorithm 4.3, they can be locally valid or globally valid, cf. Section 1.1.3. In order to separate intersection cuts, we need a simplex tableau, and in most MILP codes, we obtain that of node η, including local bounds on the variables. This means that the corresponding intersection cuts are local cuts. In order to obtain globally valid cuts, one would have to recover a globally valid tableau, which might be more involved. Implementation-wise, lift-and-project cuts are less demanding in this respect. If we use Algorithm 4.2 to separate split cuts, we can control whether they are locally or globally valid by initializing the membership LP, cf. Line 1 of Algorithm 4.2, with data of the LP associated to η or data associated to the root node ρ.*

4.3.1 TSP with multiple time windows

In what follows, we test Algorithm 4.3 in the context of the TSPMTW, cf. Section 1.5.1. We used instances that are based on the VRPMTW instances in [BHL14]. Therein, graphs with 100 nodes are used, and we thus created a TSPMTW instance from a VRPMTW one on a subgraph by randomly selecting $n \in \{25, 40\}$ of them. The node characteristics such as the process time and the multiple time windows remain unchanged. For the two values of n, we created at most 5 TSPMTW instances from an original VRPMTW one, ending up with 142 feasible instances in total.

As was noted in Section 4.1.1.1, for the TSPMTW we have a pair of MILPncsd and MILP models, (4.17) - (4.22) and (1.32) - (1.37) augmented by (1.39) - (1.42),

[12]Unless primal heuristics are disabled, the condition in Line 8 of Algorithm 4.3 will also be checked whenever CPLEX finds an integer feasible solution that is not necessarily the solution of an LP solved at some node η.

[13]The branch-callback of CPLEX allows to branch only by tightening the integer variable bounds on the two child nodes, thus mimicking a simple split. A possible workaround in order to be able to branch on more general disjunctive constraints as in Line 12 of Algorithm 4.3 would be to introduce an additional variable $x_{n+1} = {\pi^k}^T x$ for every appearing split vector. We did not do this in our experiments though.

that are equivalent, with the difference that the non-contiguous domains in the former are reformulated with big-M constraints in the latter. The resulting auxiliary binary variables in the MILP are not present in the MILPncsd, which makes the total number of variables significantly smaller. In the MILPncsd model, all non-contiguous domains are given by simple wide splits. Therefore, we can proceed as in Example 4.14, implementing Algorithm 4.3 in C/C++ on top of CPLEX 12.6.1. In Table 4.12 we compare such an implementation applied to the MILPncsd (column CPXR) with the application of default[14] CPLEX 12.6.1 to the MILP model (column CPXF). We also include the results of Algorithm 4.3 implemented as before, but with the separation of $r = 20$ rounds of wide intersection cuts at the root node, that is, Algorithm 4.1 with $b_1 =$ `true` and $b_2 =$ `false` (column CPXR+cuts). Shown are the CPU times in seconds and the number of nodes that were explored in each case. All experiments were conducted on a single core of a 3.1 GHz quad-core machine with 1.96 GB RAM, and we imposed a time limit of one hour (indicated by ∞, when reached).

Table 4.12 shows those instances in which at least one of the three algorithms required at least 10 minutes of CPU time. The mean values instead are computed over all instances out of the 142 for which none of the three algorithms hit the time limit. We see that on average, Algorithm 4.3 reduces the computing times for solving an instance by roughly 20 % with respect to default CPLEX applied to an MILP model with auxiliary binary variables[15]. There is also a reduction in the number of branch-and-bound nodes. This nice result confirms that the direct exploitation of wide split disjunctions, in this case through branching, can be advantageous. Going over to the implementation of Algorithm 4.3 with the separation of wide split cuts at the root node, we observe yet another improvement on average, in terms of computing times as well as in terms of nodes. However, this improvement is only marginal, and in many cases, the number of nodes remains unchanged, meaning that the two algorithms probably explore the exact same search tree. This is of course consistent with what was said in the introductory part of Section 4.2.4; the TSPMTW model is far too weak in order to benefit significantly from wide split cuts that are separated at the root node.

[14] In order to maintain the results comparable, default means with empty incumbent-, branch- and cut-callbacks. Also, in the light of Remark 4.15, CPLEX's presolve was disabled in all its variants in this section, in order to be able to retrieve the simplex tableaus of the original instance for the generation of wide intersection cuts.

[15] Although the time limit is hit 3 times in the first case, and only 2 times in the latter

4.4 Outlook

The results presented in this chapter can be seen as a paradigm change in the way we model and solve MILPs. We have seen significant advantages of exploiting non-contiguous domains directly in some special cases of MILPs. So far, the appearance of holes in the domains of integer variables in general MILPs seems to be marginal. In MIPLIB for example, the majority of the variables in some instance is typically given by binary variables, in which case the search for non-contiguous domains is obsolete. Nevertheless, an admittedly provocative question that can be read between the lines of this chapter is whether the relative rarity of integer variables in integer programming problems arises naturally, or has been reinforced in some sense by the evolution of the methodologies of integer programming, that are sometimes more focused on binary variables. In that respect, we believe that the fresh view on some modeling and algorithmic aspects of MILP that has been given in this chapter is valuable.

Appendix

List of Figures

List of Algorithms

List of Tables

instance	#nodes	of which sources	#pipes	#diameters
shamir	7	1	8	14
hanoi	32	1	34	6
new york	20	1	21	12
blacksburg	31	1	35	11
foss_poly_0	37	1	58	7
foss_iron	37	1	58	13
foss_poly_1	37	1	58	22
pescara	71	3	99	13
modena	272	4	317	13

Table 2.1: Characteristics of water network design instances

	BONMIN-based [BDL+11] time limit 7,200 sec. single core 2.4 GHz 1.94 GB RAM	LP/CVXNLP [Rag13] time limit 14,400 sec. mem. limit 165,000 nodes dual core 2.63 GHz 3 GB RAM		SCIP [Vig13b, Vig12] time limit 10,000 sec. dual core 3.2 GHz 48 GB RAM	
instance	ub	lb	ub	lb	ub
shamir	419,000.00	419,000.00	419,000.00	419,000.00	419,000.00
hanoi	6,109,620.90	6,109,620.90	6,109,620.90	6,109,620.90	6,109,620.90
new york	39,307,799.72	39,307,799.72	39,307,799.72	23,363,470.00	39,307,800.00
blacksburg	118,251.09	118,251.09	118,251.09	116,945.00	116,945.00
foss_poly_0	70,680,507.90	70,062,509.72	71,741,922.90	67,559,218.00	67,559.218
foss_iron	178,494.14	177,515.24	178,494.14	175,992.00	175,992.00
foss_poly_1	29,117.04	26,236.17	31,352.87	28,043.86	28,043.86
pescara	1,820,263.72	1,700,108.17	1,814,271.91	1,639,746.00	1,938,885.00
modena	2,576,589.00	2,206,137,43	4,191,445.38	2,115,898.00	-

Table 2.2: Reported computational results for water network design instances

instance	convex combination method						incremental method					
	piecew. lin. appr.			piecew. lin. rel.			piecew. lin. appr.			piecew. lin. rel.		
	ub		time	ub		time	ub		time	ub		time
$K = 2$												
shamir	*508,000.00		40.16	385,000.00		7.97	*508,000.00		37.53	385,000.00		4.85
hanoi	-		∞	5,932,490.90		125.74	*7,523,518.00		∞	5,932,490.90		67.23
blacksburg	*137,857.71		4517.34	115,592.42		21.38	*137,857.71		1883.29	115,592.42		31.38
foss_poly_0	-		∞	-		∞	-		∞	69,994,333.10		5396.17
foss_iron	-		∞	*176,840.77		3861.78	-		∞	*176,840.77		1279.91
foss_poly_1	-		∞	-		∞	-		∞	*29,139.18		∞
pescara	-		∞	-		∞	-		∞	-		∞
modena	-		∞	-		∞	-		∞	-		∞
$K = 4$												
shamir	*419,000.00		34.96	383,000.00		27.24	*419,000.00		9.33	383,00.00		8.97
hanoi	*6,278,434.90		2168.98	5,898,285.40		227.71	*6,278,434.90		614.77	5,898,285.40		157.85
blacksburg	*121,595.08		1752.56	116,945.44		242.13	*121,595.08		324.16	116,945.44		105.48
foss_poly_0	-		∞	-		∞	-		∞	-		∞
foss_iron	-		∞	-		∞	-		∞	*180,691.01		∞
foss_poly_1	-		∞	-		∞	-		∞	*30,490.68		∞
pescara	-		∞	-		∞	-		∞	-		∞
modena	-		∞	-		∞	-		∞	-		∞
$K = 10$												
shamir	*419,000.00		1931.31	*419,000.00		779.64	*419,000.00		52.70	*419,000.00		80.19
hanoi	*6,124,518.80		4531.01	*6,105,381.40		6047.37	*6,124,518.80		779.11	*6,105,381.40		1487.99
blacksburg	*118,461.50		778.31	116,945.44		6411.51	*118,461.50		451.16	116,945.44		395.00
foss_poly_0	-		∞	-		∞	-		∞	-		∞
foss_iron	-		∞	-		∞	-		∞	-		∞
foss_poly_1	-		∞	-		∞	-		∞	-		∞
pescara	-		∞	-		∞	-		∞	-		∞
modena	-		∞	-		∞	-		∞	-		∞

Table 2.3: Comparison of piecewise linearizations for water network design instances

instance	big-M lp	CH lp	P_e big-M	P_e CH	P_e^* big-M	P_e^* CH
1nl_1	99.67	99.67	97.93	97.90	14.88	18.99
1nl_2	99.59	99.59	97.93	97.93	7.76	13.29
1nl_3	99.90	99.90	99.70	99.70	20.22	14.46
1nl_4	98.23	98.23	89.14	88.98	6.54	15.11
1nl_5	98.24	98.24	91.40	91.38	6.39	8.99
2nl_10	99.57	99.57	96.09	95.98	14.55	14.23
2nl_6	99.88	99.88	98.19	98.18	5.56	8.33
2nl_8	99.21	99.21	85.08	84.76	0.00	3.95
3nl_11	99.77	99.77	99.14	99.13	10.55	10.15
3nl_12	99.82	99.82	99.49	99.49	11.49	8.62
3nl_13	99.83	99.83	99.38	99.38	19.24	19.30
3nl_14	99.87	99.87	99.58	99.58	49.84	24.46
3nl_15	99.83	99.83	99.50	99.50	27.22	24.23
4nl_16	99.76	99.76	98.10	98.05	9.50	7.57
4nl_17	1.32	0.00	0.00	0.00	0.00	0.00
5nl_10	99.76	99.76	98.77	98.76	11.49	12.00
5nl_11	99.78	99.78	98.56	98.55	18.24	12.63
5nl_12	99.83	99.83	99.09	99.08	28.57	14.84
5nl_1	99.78	99.78	98.92	98.92	8.47	9.31
5nl_2	99.78	99.78	98.52	98.51	17.23	16.90
5nl_3	99.84	99.84	99.13	99.12	16.04	18.17
5nl_4	99.77	99.77	98.86	98.86	18.72	19.21
5nl_5	99.79	99.79	98.49	98.48	15.57	21.53
5nl_6	99.85	99.85	99.17	99.17	13.96	10.98
5nl_7	99.77	99.77	98.85	98.85	5.88	7.85
5nl_8	99.78	99.78	98.47	98.46	17.55	30.94
5nl_9	99.84	99.84	99.12	99.12	14.84	12.49
mean	96.58	96.54	94.65	94.62	14.39	14.10

Table 3.1: Comparison of dual gaps of big-M and disjunctive formulations for supervised classification instances with *weak* preprocessing

instance	big-M lp	CH lp	P_e big-M	P_e CH	P_e^* big-M	P_e^* CH
2nl_10	9.12	1.33	0.00	0.00	0.00	0.00
2nl_6	0.01	0.00	0.00	0.00	0.00	0.00
3nl_13	93.28	82.92	63.03	30.66	1.59	0.09
3nl_14	0.01	0.00	0.00	0.00	0.00	0.00
3nl_15	93.95	93.93	67.18	64.10	0.85	1.93
4nl_16	0.24	0.00	0.00	0.00	0.00	0.00
4nl_17	1.06	0.00	0.00	0.00	0.00	0.00
5nl_14	0.01	0.00	0.00	0.00	0.00	0.00
mean	24.71	22.27	16.28	11.85	0.31	0.25

Table 3.2: Comparison of dual gaps of big-M and disjunctive formulations for supervised classification instances with *strong* preprocessing

instance	big-M lp	CH lp	P_e big-M	P_e CH	P_e^* big-M	P_e^* CH
abz5-10x10	30.38	28.46	25.27	24.48	23.23	23.40
abz6-10x10	21.31	20.65	17.28	16.06	14.08	15.47
abz7-20x15	37.50	35.30	33.34	*32.42	*32.19	*32.44
abz8-20x15	33.78	31.75	30.53	*29.68	*30.35	*29.68
abz9-20x15	31.22	29.28	27.45	*27.03	*27.32	*26.96
ft06-6x6	14.54	14.54	14.25	13.47	11.86	12.59
ft10-10x10	29.56	28.12	25.68	24.12	22.04	23.93
ft20-20x5	66.78	64.52	63.24	61.56	60.86	60.75
la01-10x5	37.98	34.17	29.34	26.75	29.02	26.75
la02-10x5	39.84	39.70	37.40	34.95	35.45	33.01
la03-10x5	41.54	38.49	32.58	31.67	31.92	31.64
la04-10x5	37.45	34.05	27.95	26.17	27.16	26.16
la05-10x5	35.91	35.91	35.59	34.20	34.66	34.16
la06-15x5	55.39	52.65	49.78	49.01	49.31	47.91
la07-15x5	57.75	56.88	54.97	53.42	53.78	52.86
la08-15x5	57.24	53.56	49.06	47.70	47.03	47.47
la09-15x5	59.83	57.18	53.98	52.75	53.66	52.66
la10-15x5	53.75	52.98	51.30	49.94	50.30	49.32
la11-20x5	66.20	64.12	61.11	*60.47	60.92	60.42
la12-20x5	60.73	60.56	59.07	58.29	56.51	57.23
la13-20x5	66.78	63.97	60.69	58.99	60.08	*57.66
la14-20x5	65.71	64.97	62.04	61.17	60.62	60.80
la15-20x5	68.68	64.90	61.39	60.57	60.59	60.57
la16-10x10	24.12	22.49	20.09	19.80	20.03	19.51
la17-10x10	17.60	16.65	15.30	15.27	15.28	15.12
la18-10x10	21.81	20.89	18.39	17.95	17.81	17.65
la19-10x10	26.72	24.57	21.87	20.59	20.28	19.67
la20-10x10	16.18	16.18	15.58	15.04	13.83	14.72
la21-15x10	31.45	31.43	29.92	28.69	29.01	28.63
la22-15x10	33.22	30.72	27.26	25.63	27.26	25.59
la23-15x10	37.98	35.34	32.14	31.87	32.12	31.38
la24-15x10	24.70	24.65	23.49	22.11	21.78	21.87
la25-15x10	25.99	25.09	23.94	22.80	22.91	22.58
la26-20x10	41.13	39.18	36.59	*35.84	*36.27	*36.11
mean	40.31	38.64	36.11	35.01	34.98	34.60

Table 3.3: Comparison of dual gaps of big-M and disjunctive formulations for JSS instances

instance	big-Mz lp	CHz lp	big-Mx lp	CHx lp	P_e big-Mx	P_e CHx	P_e^* big-Mx	P_e^* CHx
rbg035a.2	0.54	0.51	0.53	0.50	0.45	0.39	0.33	0.34
rbg035a	2.33	2.27	2.32	2.25	1.87	1.15	1.21	0.83
rbg038a	0.52	0.52	0.52	0.52	0.37	0.25	0.34	0.23
rbg040a	4.50	4.40	4.48	4.37	3.92	2.97	2.92	2.55
rbg041a	3.87	3.85	3.87	3.85	3.50	3.06	2.76	2.62
rbg042a	2.33	2.24	2.32	2.23	1.86	1.45	1.41	1.22
rbg048a	1.35	1.35	1.35	1.35	1.34	1.31	1.33	1.29
rbg049a	1.22	1.22	1.22	1.22	1.19	1.18	1.16	1.15
rbg050a	0.84	0.82	0.84	0.82	0.76	0.57	0.72	0.49
rbg050b	1.06	1.06	1.06	1.06	1.05	0.98	1.03	0.95
rbg050c	0.69	0.68	0.69	0.68	0.67	0.63	0.62	0.61
rbg055a	0.98	0.98	0.98	0.98	0.95	0.87	0.87	0.74
rbg067a	0.80	0.80	0.80	0.80	0.77	0.70	0.69	0.55
rbg086a	0.94	0.93	0.94	0.92	0.81	0.66	0.72	0.64
rbg092a	0.88	0.86	0.88	0.86	0.81	0.70	0.72	0.66
rbg125a	1.37	1.32	1.37	1.31	1.13	0.96	0.94	0.76
rbg132.2	1.54	1.52	1.54	1.51	1.42	1.21	1.24	1.12
rbg132	2.22	2.13	2.21	2.13	1.87	1.34	1.46	1.09
rbg152.3	0.63	0.62	0.63	0.61	*0.58	*0.52	*0.53	*0.50
rbg152	1.30	1.24	1.28	1.22	*1.05	*0.70	*0.76	*0.50
rbg172a	1.65	1.57	1.64	1.56	1.36	*1.14	1.15	*0.91
rbg193.2	1.68	1.61	1.68	1.60	*1.50	*1.22	*1.23	*1.05
rbg193	1.67	1.55	1.67	1.54	*1.56	*1.20	*1.18	*1.01
rbg201a	2.03	1.93	2.02	1.92	1.88	*1.45	1.48	*1.27
rbg233.2	1.80	1.74	1.80	1.73	*1.63	*1.42	*1.39	*1.37
rbg233	2.04	1.91	2.03	1.88	*1.77	*1.55	*1.51	*1.38
mean	2.81	2.77	2.80	2.76	2.62	2.45	2.42	2.32

Table 3.4: Comparison of dual gaps of big-M and disjunctive formulations for TSPTW instances

instance	t_{LP}	n_{MILP}	time persp0	time persp1	#nodes persp0	#nodes persp1
abz5-10x10	14.84	8	58.01	121.54	2647.25	3828.50
abz6-10x10	13.75	8	18.03	20.25	381.12	478.38
ft06-6x6	1.95	4	2.01	2.01	35.75	21.75
ft10-10x10	140.69	6	349.12	590.04	16078.33	18016.17
la01-10x5	33.96	1	42.21	124.52	11089.00	11951.00
la02-10x5	60.15	1	66.00	137.06	6327.00	5065.00
la03-10x5	46.55	2	50.63	51.07	3081.00	2184.50
la04-10x5	23.40	4	26.00	26.36	1237.75	1349.50
la05-10x5	14.36	5	15.54	16.56	793.20	802.20
la16-10x10	15.81	5	26.58	35.25	1504.40	1749.20
la17-10x10	17.58	6	31.44	44.52	1828.67	1717.33
la18-10x10	63.47	2	68.00	68.05	1576.50	1109.50
la19-10x10	32.48	4	39.85	43.68	1000.75	1040.00
la20-10x10	14.98	6	19.53	21.78	743.00	731.00
la22-15x10	228.36	1	10379.87	∞	1537622.00	-
la24-15x10	244.00	1	9514.04	6311.43	1397201.00	675751.00
la25-15x10	141.29	4	3372.31	∞	140029.25	-
ta01-15x15	85.34	5	∞	3859.66	-	40137.80
ta02-15x15	232.72	3	∞	6211.82	-	111951.67
ta03-15x15	140.31	6	7608.38	6967.78	109298.67	69376.33
ta04-15x15	385.74	3	8080.79	6130.63	227370.33	103396.33
ta08-15x15	274.44	3	∞	9969.92	-	192434.67
mean	74.34	4.24	1530.36	1218.39	104834.92	52856.92

Table 3.5: Comparison of strategies `persp0` and `persp1` in the tree of trees for JSS instances

q	total	group 1	group 2	group 3	group 4
1	53	3	3	2	45
2	54	4	1	7	42
3	54	6	0	2	46
4	53	25	14	8	6
5	54	5	0	2	47
6	50	21	20	2	7
7	51	22	19	3	7
8	53	4	0	2	47
9	26	0	0	1	25

Table 4.1: Division of LBP instances after the separation of wide intersection cuts by instance class

	% gap closed			#cuts		
instance	w/o	w	o	w/o	w	o
10000-4-100-75	9.21	12.53	11.73	139	62	5
10000-4-10-25	32.97	34.28	28.40	46	27	2
10000-4-10-50	79.12	100.00	100.00	1	1	1
10000-4-10-75	34.31	42.21	35.69	55	32	2
10000-4-20-50	16.58	22.69	1.50	38	43	1
10000-4-20-75	8.37	11.45	4.41	48	61	2
10000-4-30-25	22.37	29.34	21.16	51	42	3
10000-4-30-75	30.70	32.50	29.19	71	43	2
10000-4-40-75	12.35	24.40	22.84	120	64	5
10000-4-500-50	1.10	6.17	6.06	170	137	10
10000-4-500-75	0.38	1.41	1.34	123	81	6
10000-4-50-50	29.26	31.48	27.10	100	48	3
1000-4-1000-50	21.36	30.46	30.17	179	59	4
1000-4-100-25	3.30	5.16	0.00	39	36	0
1000-4-10-50	35.99	68.49	56.18	37	31	2
1000-4-10-75	18.75	21.15	14.57	50	33	1
1000-4-2000-75	5.21	7.11	6.83	158	76	4
1000-4-20-25	38.73	100.00	100.00	48	9	2
1000-4-20-75	28.94	100.00	100.00	117	21	3
1000-4-30-50	11.54	16.34	13.80	94	41	3
1000-4-40-25	17.78	23.10	19.18	70	46	3
1000-4-40-50	12.04	13.39	11.68	97	50	3
1000-4-40-75	8.92	42.90	41.38	150	53	5
1000-4-500-75	2.97	5.10	4.89	134	73	6
1000-4-50-50	13.01	17.12	15.26	113	60	4
mean	19.81	31.95	28.13	89.92	49.16	3.28

Table 4.2: Separation of wide intersection cuts on LBP instances, group 1

	% gap closed			#cuts		
instance	w/o	w	o	w/o	w	o
10000-4-1000-25	1.32	1.96	1.73	124	72	5
10000-4-1000-50	0.92	1.39	1.22	161	69	5
10000-4-100-25	2.36	2.64	0.98	41	56	1
10000-4-100-50	2.26	2.16	1.30	82	57	3
10000-4-2000-25	0.39	0.49	0.37	61	38	3
10000-4-2000-50	0.41	0.42	0.35	69	48	3
10000-4-2000-75	1.32	1.34	1.16	67	50	3
10000-4-30-50	30.04	30.06	29.90	30	18	2
10000-4-500-25	1.59	2.03	1.59	98	53	4
1000-4-10-25	99.49	100.00	100.00	27	3	1
1000-4-20-50	22.83	22.59	19.18	85	37	3
1000-4-30-25	16.56	16.67	13.26	63	30	2
1000-4-30-75	17.46	17.53	16.34	93	50	3
1000-4-50-75	2.14	2.31	0.00	48	41	0
mean	14.22	14.39	13.38	74.92	44.42	2.71

Table 4.3: Separation of wide intersection cuts on LBP instances, group 2

	% gap closed			#cuts		
instance	w/o	w	o	w/o	w	o
10000-4-1000-75	0.42	0.43	0.27	52	48	2
10000-4-20-25	32.03	31.62	25.15	52	48	2
10000-4-40-25	3.46	3.46	0.61	11	11	1
10000-4-40-50	5.70	6.07	2.30	58	64	4
10000-4-50-25	8.87	8.87	1.37	20	20	1
10000-4-50-75	3.09	3.09	0.56	64	64	2
1000-4-100-50	2.40	2.40	1.35	47	46	2
1000-4-50-25	8.28	8.64	5.24	61	66	3
mean	8.03	8.07	4.60	45.62	45.87	2.12

Table 4.4: Separation of wide intersection cuts on LBP instances, group 3

	% gap closed			#cuts		
instance	w/o	w	o	w/o	w	o
10000-4-100-75	2.38	6.81	7.27	31	17	10
10000-4-10-25	27.00	30.40	28.40	7	9	3
10000-4-10-75	12.79	35.79	35.69	18	13	8
10000-4-20-25	14.12	26.22	25.15	16	13	8
10000-4-30-25	6.19	20.42	20.42	13	15	10
10000-4-30-50	21.34	29.95	29.90	11	14	3
10000-4-30-75	11.49	26.38	27.86	26	15	10
10000-4-40-75	2.93	12.14	9.81	46	17	10
10000-4-50-50	7.53	22.97	22.97	36	19	10
1000-4-1000-50	4.89	17.77	17.76	93	22	10
1000-4-10-50	10.52	56.72	56.18	5	9	5
1000-4-10-75	7.25	15.90	14.57	15	7	1
1000-4-2000-75	2.47	4.39	4.56	50	24	10
1000-4-20-25	9.80	100.00	100.00	11	9	6
1000-4-20-50	12.71	21.01	19.18	19	15	10
1000-4-20-75	5.91	25.95	25.30	68	17	10
1000-4-30-25	4.49	10.16	10.16	23	15	10
1000-4-30-50	3.04	10.63	12.19	30	15	10
1000-4-30-75	7.67	14.54	14.41	42	24	10
1000-4-40-25	4.17	12.96	11.60	29	17	10
1000-4-40-50	3.29	9.19	8.27	37	23	10
1000-4-40-75	2.52	11.23	9.11	33	22	10
1000-4-500-75	0.86	2.41	2.38	46	18	10
1000-4-50-25	5.10	6.36	5.24	19	15	5
1000-4-50-50	3.20	10.48	10.43	58	19	10
mean	7.74	21.63	21.15	31.28	16.12	8.36

Table 4.5: Separation of wide lift-and-project cuts on LBP instances, group 1

instance	% gap closed			#cuts		
	w/o	w	o	w/o	w	o
10000-4-1000-25	0.46	1.20	1.27	12	17	10
10000-4-1000-50	0.27	0.72	0.64	52	20	10
10000-4-100-25	1.60	1.60	0.98	9	11	1
10000-4-100-50	1.54	1.69	1.30	11	16	4
10000-4-10-50	100.00	100.00	100.00	11	2	1
10000-4-2000-25	0.19	0.36	0.36	11	13	10
10000-4-2000-50	0.15	0.32	0.28	20	14	10
10000-4-20-50	6.44	6.44	1.50	11	13	1
10000-4-40-25	1.58	1.58	0.61	12	14	1
10000-4-40-50	3.92	3.92	2.30	14	17	4
10000-4-500-50	0.27	0.85	1.07	42	22	10
10000-4-500-75	0.10	0.54	0.50	47	17	10
10000-4-50-75	2.53	2.53	0.56	8	9	2
1000-4-100-50	1.76	1.76	1.35	10	12	3
1000-4-10-25	100.00	100.00	100.00	11	3	1
mean	14.72	14.90	14.18	18.73	13.33	5.20

Table 4.6: Separation of wide lift-and-project cuts on LBP instances, group 2

instance	% gap closed			#cuts		
	w/o	w	o	w/o	w	o
10000-4-1000-75	0.35	0.35	0.27	12	13	2
10000-4-2000-75	0.69	1.20	1.15	27	25	10
10000-4-20-75	5.02	5.02	4.41	11	12	3
10000-4-500-25	0.47	1.31	1.17	22	21	10
10000-4-50-25	4.20	4.20	1.37	11	12	1
mean	2.14	2.41	1.67	16.60	16.60	5.20

Table 4.7: Separation of wide lift-and-project cuts on LBP instances, group 3

Table 4.8: Separation of wide intersection cuts on LBP and MLBP instances

	LBP			MLBP, $K = 2$			MLBP, $K = 5$			MLBP, $K = 10$		
instance	w/o	w	Δ	w/o	w	Δ	w/o	w	Δ	w/o	w	Δ
10000-4-100-75	9.21	12.53	36.04	6.51	10.21	56.83	1.45	0.94	-35.17	0.04	22.78	56850.00
10000-4-10-25	32.97	34.28	3.97	19.55	20.35	4.09	6.94	45.28	552.44	0.20	6.88	3340.00
10000-4-10-50	79.12	100.00	26.39	10.47	10.55	0.76	2.67	28.58	970.41	2.87	25.54	789.89
10000-4-10-75	34.31	42.21	23.02	35.78	35.90	0.33	3.06	4.63	51.30	1.38	1.55	12.31
10000-4-20-50	16.58	22.69	36.85	30.30	35.67	17.72	2.25	8.58	281.33	0.02	18.15	90650.00
10000-4-20-75	8.37	11.45	36.79	11.29	26.33	133.21	11.44	42.80	274.12	0.03	1.78	5833.33
10000-4-30-25	22.37	29.34	31.15	16.68	20.95	25.59	2.56	1.94	-24.21	0.27	7.70	2751.85
10000-4-30-75	30.70	32.50	5.86	9.08	9.39	3.41	5.53	3.89	-29.65	0.07	2.04	2814.28
10000-4-40-75	12.35	24.40	97.57	16.58	27.79	67.61	4.15	4.18	0.72	0.00	0.00	∞
10000-4-500-50	1.10	6.17	460.90	3.37	7.40	119.58	1.95	2.47	26.66	0.00	0.13	∞
10000-4-500-75	0.38	1.41	271.05	1.17	1.41	20.51	0.74	4.79	547.29	0.01	0.37	3600.00
10000-4-50-50	29.26	31.48	7.58	15.71	21.10	34.30	5.80	8.80	51.72	0.09	0.00	-100.00
1000-4-1000-50	21.36	30.46	42.60	13.29	33.22	149.96	0.63	4.39	596.82	0.14	0.14	0.00
1000-4-100-25	3.30	5.16	56.36	3.41	5.07	48.68	2.95	36.64	1142.03	0.04	0.06	50.00
1000-4-10-50	35.99	68.49	90.30	33.20	33.74	1.62	2.77	79.33	2763.89	2.57	7.01	172.76
1000-4-10-75	18.75	21.15	12.80	14.36	13.38	-6.82	1.66	8.38	404.81	4.57	59.71	1206.56
1000-4-2000-75	5.21	7.11	36.46	0.18	0.30	66.66	0.11	0.36	227.27	0.01	0.01	0.00
1000-4-20-25	38.73	100.00	158.19	52.74	53.62	1.66	3.10	5.45	75.80	2.72	4.94	81.61
1000-4-20-75	28.94	100.00	245.54	20.44	53.80	163.20	6.45	7.59	17.67	0.00	2.49	∞
1000-4-30-50	11.54	16.34	41.59	7.18	8.30	15.59	1.85	1.67	-9.72	0.02	7.56	37700.00
1000-4-40-25	17.78	23.10	29.92	16.28	16.50	1.35	11.51	13.07	13.55	0.02	21.24	106100.00
1000-4-40-50	12.04	13.39	11.21	8.71	9.10	4.47	11.75	17.79	51.40	0.03	3.21	10600.00
1000-4-40-75	8.92	42.90	380.94	34.55	39.68	14.84	0.90	5.88	553.33	0.07	16.98	24157.14
1000-4-500-75	2.97	5.10	71.71	1.12	2.75	145.53	0.32	0.69	115.62	0.18	2.89	1505.55
1000-4-50-50	13.01	17.12	31.59	8.59	12.17	41.67	4.33	4.32	-0.23	0.02	3.45	17150.00
mean	19.81	31.95	89.85	15.62	20.34	45.29	3.87	13.69	344.76	0.61	8.66	16602.96

instance	% gap closed		
	w	w-s	w/o-g
10000-4-100-75	12.53	12.53	9.21
10000-4-10-25	34.28	34.28	32.96
10000-4-10-75	42.21	42.21	34.31
10000-4-20-50	22.69	30.20	7.89
10000-4-20-75	11.45	22.51	8.37
10000-4-30-25	29.34	29.34	22.37
10000-4-30-75	32.50	32.52	30.70
10000-4-40-75	24.40	24.37	12.35
10000-4-500-50	6.17	6.17	1.05
10000-4-500-75	1.41	1.41	0.34
10000-4-50-50	31.48	31.48	29.26
1000-4-1000-50	30.46	30.46	22.72
1000-4-100-25	5.16	5.16	3.48
1000-4-10-50	68.49	68.49	35.99
1000-4-10-75	21.15	22.54	18.78
1000-4-2000-75	7.11	7.11	6.49
1000-4-20-25	100.00	100.00	38.73
1000-4-20-75	100.00	100.00	28.94
1000-4-30-50	16.34	16.34	11.54
1000-4-40-25	23.10	23.10	17.78
1000-4-40-50	13.39	13.39	12.04
1000-4-40-75	42.90	42.81	8.55
1000-4-500-75	5.10	5.10	3.04
1000-4-50-50	17.12	17.12	13.01
mean	29.11	29.94	17.07

Table 4.9: Separation of strengthened wide intersection cuts on LBP instances

instance	% gap closed			#cuts		
	w/o	w	o	w/o	w	o
30n20b8	18.73	18.49	0.07	2218	2169	203
N30-E60-W5-1hX	100.00	100.00	0.00	1105	1037	0
N30-E60-W5-1h	21.44	21.37	0.12	2280	2173	92
N30-E60-W5-20m5bX	25.37	25.37	0.00	1330	1370	136
N30-E60-W5-20m5b	33.93	37.80	0.27	1759	1729	185
N50-E70-W6-1hX	100.00	100.00	0.00	2144	1924	0
N50-E70-W6-1h	63.25	64.13	0.07	2964	2745	89
N50-E70-W6-20m5bX	32.97	32.21	0.10	3137	3144	137
N50-E70-W6-20m5b	6.87	7.18	0.07	3178	3134	279
mean	44.72	45.17	0.07	2235.00	2158.33	124.55

Table 4.10: Separation of wide intersection cuts on Multi-mode Resource Leveling instances

instance	$\mu = 1$				$\mu = 2$				$\mu = 3$			
	w/o	w	o	(#)	w/o	w	o	(#)	w/o	w	o	(#)
big-M constraints												
lectsched-4-obj	65.01	76.81	0.00	(5)	65.01	88.56	0.00	(5)	64.59	95.00	0.00	(5)
mik-250-1-100-1	62.48	69.81	0.00	(3)	34.35	65.44	0.00	(4)	36.64	61.23	0.00	(5)
mzzv11	55.84	59.97	30.69	(2)	47.18	59.53	48.40	(2)	-	-	-	(0)
n4-3	37.81	37.89	3.39	(5)	32.68	32.93	2.91	(5)	32.87	34.85	5.53	(5)
n9-3	23.56	24.09	0.00	(1)	22.26	23.21	4.98	(4)	31.20	32.76	17.96	(3)
neos-1224597	0.00	0.00	0.00	(5)	0.00	0.00	0.00	(4)	0.00	0.00	0.00	(4)
neos16	27.06	28.23	16.47	(5)	17.65	17.65	0.00	(3)	35.96	60.78	13.72	(3)
neos-555424	37.73	38.55	13.16	(5)	42.45	73.38	42.02	(5)	41.65	76.30	72.97	(5)
neos-686190	4.64	7.83	4.63	(5)	3.23	4.61	2.77	(5)	1.35	1.35	0.00	(1)
noswot	0.00	0.00	0.00	(5)	0.00	0.00	0.00	(5)	0.00	0.00	0.00	(5)
rococoB10-011000	13.67	20.06	0.41	(5)	17.66	17.40	1.41	(5)	40.74	43.15	31.52	(5)
rococoC10-001000	28.81	28.47	2.25	(5)	43.72	43.85	39.58	(5)	22.86	22.80	22.39	(5)
sp98ir	8.41	8.18	5.12	(5)	7.38	7.84	7.41	(5)	7.11	7.35	7.24	(5)
timtab1	31.57	33.20	3.12	(5)	32.15	32.69	1.04	(5)	31.86	32.94	2.08	(5)
mean	28.33	30.93	5.66		26.12	33.36	10.75		26.68	36.04	13.34	
GUB-links												
lectsched-4-obj	55.01	86.04	0.00	(5)	65.05	90.00	0.00	(5)	60.20	99.92	0.00	(5)
neos-1224597	0.00	0.00	0.00	(5)	0.00	0.00	0.00	(4)	0.00	0.00	0.00	(4)
neos16	17.65	29.41	16.47	(5)	23.53	23.53	0.00	(3)	23.53	60.78	13.72	(3)
sp98ir	7.40	7.57	5.05	(5)	6.97	6.66	6.77	(5)	6.95	7.03	7.35	(5)
timtab1	33.31	33.31	3.12	(5)	32.73	32.73	1.04	(5)	33.02	33.02	2.08	(5)
mean	22.68	31.26	4.93		25.66	30.58	1.56		24.74	40.15	4.63	

Table 4.11: Separation of wide intersection cuts on MIPLIB 2010 instances with random holes

instance	CPXF		CPXR		CPXR+cuts	
	time	#nodes	time	#nodes	time	#nodes
pcm201-25-1	205.73	(321133)	480.73	(1081462)	1253.62	(2437962)
pcm203-25-2	3358.90	(5440357)	1434.02	(2715201)	1428.14	(2715201)
pcm203-25-5	553.25	(690338)	642.16	(1067973)	643.56	(1067973)
prcm201-25-5	207.77	(316152)	701.80	(1072706)	581.16	(870949)
prcm203-25-5	244.14	(343000)	597.33	(965776)	827.60	(1267787)
prcm204-25-1	1145.16	(1538553)	959.36	(1330937)	983.92	(1330937)
prcm204-25-4	2169.15	(2642272)	∞	-	∞	-
prm203-25-1	700.74	(928593)	1131.38	(1638184)	1127.50	(1638184)
prm204-25-1	967.65	(1720791)	649.95	(1316733)	632.09	(1316733)
prm204-25-4	1632.59	(2181347)	∞	-	1606.19	(2326136)
rcm201-25-4	2073.09	(2853813)	201.41	(375224)	423.19	(745732)
rcm202-25-4	1026.36	(1418926)	1564.71	(2413034)	849.43	(1390125)
rcm202-25-5	1666.60	(2020490)	992.94	(1654361)	168.36	(304369)
rcm203-25-4	571.71	(1113311)	707.42	(1223908)	350.03	(619849)
rcm203-25-5	1165.88	(1541838)	605.88	(971505)	488.71	(773791)
rcm204-25-5	721.98	(1043259)	610.87	(1010179)	698.71	(1179048)
rcm205-25-4	∞	-	∞	-	∞	-
rcm206-25-4	∞	-	3236.25	(4926167)	3244.05	(4926167)
rcm206-25-5	3366.48	(4545009)	2636.12	(3987679)	2655.27	(3987679)
rcm207-25-1	942.16	(1257681)	257.59	(452846)	487.36	(833113)
rcm208-25-4	1485.43	(2133549)	2143.08	(3161291)	2137.38	(3161291)
rm201-25-2	617.10	(1259941)	52.95	(136317)	34.55	(83781)
rm203-40-5	1012.06	(1340578)	206.49	(241001)	206.68	(241001)
rm207-40-5	1206.69	(1024237)	996.47	(954226)	1156.14	(1240618)
mean	278.98	(414106.47)	217.22	(360721.99)	211.04	(349965.13)

Table 4.12: Comparison of direct MILPncd and MILP approaches for TSPMTW instances

Bibliography

[AAG09] M. Selim Aktürk, Alper Atamtürk, and Sinan Gürel. A strong conic quadratic reformulation for machine-job assignment with controllable processing times. *Operations Research Letters*, 37(3):187–191, 2009.

[ABZ88] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

[Ach09] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technical University of Berlin, 2009.

[ACL05a] Kent Andersen, Gérard Cornuéjols, and Yanjun Li. Reduce-and-split cuts: Improving the performance of mixed-integer gomory cuts. *Management Science*, 51(11):1720–1732, 2005.

[ACL05b] Kent Andersen, Gérard Cornuéjols, and Yanjun Li. Split closure and intersection cuts. *Mathematical programming*, 102(3):457–493, 2005.

[Ake78] Sheldon B. Akers. Binary decision diagrams. *Computers, IEEE Transactions on*, 100(6):509–516, 1978.

[AKM05] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[ALL10] Kumar Abhishek, Sven Leyffer, and Jeffrey T. Linderoth. FILMINT: An Outer Approximation-Based Solver for Convex Mixed-Integer Nonlinear Programs. *INFORMS Journal on computing*, 22(4):555–567, 2010.

[Asc95] Norbert Ascheuer. *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems*. PhD thesis, 1995.

[Bal71] Egon Balas. Intersection cuts–a new type of cutting planes for integer programming. *Operations Research*, 19(1):19–39, 1971.

[Bal79] Egon Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.

[Bal98] Egon Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1):3–44, 1998.

[BAO06] Walid Ben-Ameur and Adam Ouorou. Mathematical models of the delay constrained routing problem. *Algorithmic Operations Research*, 1(2), 2006.

[BBC+08] Pierre Bonami, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, François Margot, Nicolas Sawaya, and Andreas Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186 – 204, 2008.

[BBF+14] Pietro Belotti, Pierre Bonami, Matteo Fischetti, Andrea Lodi, Michele Monaci, Amaya Nogales-Gómez, and Domenico Salvagnin. On handling indicator constraints in mixed-integer programming. Technical Report OR/13/1, revised OR/14/20, DEI, University of Bologna, 2014.

[BCC93] Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical programming*, 58(1-3):295–324, 1993.

[BCC96] Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996.

[BDL+11] Cristiana Bragalli, Claudia D'Ambrosio, Jon Lee, Andrea Lodi, and Paolo Toth. On the optimal design of water distribution networks: a practical MINLP approach. *Optimization and Engineering*, 13(2):219–246, 2011.

[BGS08] Jens Burgschweiger, Bernd Gnädig, and Marc C. Steinbach. Optimization models for operative planning in drinking water networks. *Optimization and Engineering*, 10(1):43–73, 2008.

[BGS09] Jens Burgschweiger, Bernd Gnädig, and Marc C. Steinbach. Nonlinear programming techniques for operative planning in large drinking water networks. *Open Applied Mathematics Journal*, 3:14–28, 2009.

[BHL14] Slim Belhaiza, Pierre Hansen, and Gilbert Laporte. A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows. *Computers & Operations Research*, 52:269–281, 2014.

[BHV12] Timo Berthold, Stefan Heinz, and Stefan Vigerske. Extending a CIP framework to solve MIQCPs. In Jon Lee and Sven Leyffer, editors,

Mixed Integer Nonlinear Programming, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 427 – 444. 2012.

[Bix12] Robert E. Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2012.

[BJ80] Egon Balas and Robert G. Jeroslow. Strengthening cuts for mixed integer programs. *European Journal of Operational Research*, 4(4):224–234, 1980.

[BJS11] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[BKL+13] Pietro Belotti, Christian Kirches, Sven Leyffer, Jeffrey T. Linderoth, James Luedtke, and Ashutosh Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

[BKSW05] Artem Babayan, Zoran Kapelan, Dragan Savic, and Godfrey Walters. Least-cost design of water distribution networks under demand uncertainty. *Journal of Water Resources Planning and Management*, 131(5):375–382, 2005.

[BLL+09] Pietro Belotti, Jon Lee, Leo Liberti, François Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4-5):597–634, 2009.

[BLL+11a] Pietro Belotti, Leo Liberti, Andrea Lodi, Giacomo Nannicini, and Andrea Tramontani. Disjunctive inequalities: applications and extensions. *Wiley Encyclopedia of Operations Research and Management Science*, 2011.

[BLL11b] Pierre Bonami, Jeffrey T. Linderoth, and Andrea Lodi. Disjunctive cuts for mixed integer nonlinear programming problems. *Progress in Combinatorial Optimization*, pages 521–544, 2011.

[BLTW15] Pierre Bonami, Andrea Lodi, Andrea Tramontani, and Sven Wiese. On mathematical programming with indicator constraints. *Mathematical Programming*, 151(1):191–223, 2015.

[Bon11] Pierre Bonami. Lift-and-project cuts for mixed integer convex programs. In *Integer Programming and Combinatoral Optimization*, pages 52–64. Springer, 2011.

[Bon12] Pierre Bonami. On optimizing over lift-and-project closures. *Mathematical Programming Computation*, 4(2):151–179, 2012.

[BP02] Egon Balas and Michael Perregaard. Lift-and-project for mixed 0–1 programming: recent progress. *Discrete Applied Mathematics*, 123(1):129–154, 2002.

[BPS00] Jacek Błażewicz, Erwin Pesch, and Malgorzata Sterna. The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, 127(2):317–331, 2000.

[BQ10] Egon Balas and Andrea Qualizza. Stronger cuts from weaker disjunctions. Working paper, Carnegie Mellon University, 2010.

[Bro11] J. Paul Brooks. Support vector machines with the ramp loss and the hard margin loss. *Operations research*, 59(2):467–479, 2011.

[BT70] Martin Beale and John A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In J. Lawrence, editor, *Proceedings of the Fifth International Conference on Operations Research*, pages 447–454. Tavistock Publications, 1970.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[CAC14] Bernardete Coelho and António Andrade-Campos. Efficiency achievement in water supply systems–a review. *Renewable and Sustainable Energy Reviews*, 30:59 – 84, 2014.

[CCH+78] M. Collins, L. Cooper, R. Helgason, J. Kennington, and L. LeBlanc. Solving the pipe network analysis problem using optimization techniques. *Management Science*, 24(7):747–760, 1978.

[CFN77] Gérard Cornuéjols, Marshall L. Fisher, and George L. Nemhauser. Location of BankAccounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms. *Management Science*, 23:789–810, 1977.

[CKS90] William Cook, Ravindran Kannan, and Alexander Schrijver. Chvátal closures for mixed integer programming problems. *Mathematical Programming*, 47(1-3):155–174, 1990.

[CLS10] Eamonn Coughlan, Marco Lübbecke, and Jens Schulz. A branch-and-price algorithm for multi-mode resource leveling. In Paola Festa, editor,

Experimental Algorithms, volume 6049 of *Lecture Notes in Computer Science*, pages 226–238. Springer Berlin / Heidelberg, 2010.

[COIN] Computational Infrastructure for Operations Research. http://www.coin-or.org. Accessed: 2016-01-15.

[Coo12] William Cook. Markowitz and Manne+ Eastman+ Land and Doig= branch and bound. *Optimization Stories*, pages 227–238, 2012.

[Cor08] Gérard Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112(1):3–44, 2008.

[CP94] Jacques Carlier and Eric Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78(2):146–161, 1994.

[CPX] IBM ILOG CPLEX Optimization Studio. http://www-03.ibm.com/software/products/it/ibmilogcpleoptistud. Accessed: 2016-01-15.

[CS99] Sebastián Ceria and João Soares. Convex programming for disjunctive convex optimization. *Mathematical Programming*, 86(3):595–614, 1999.

[Dan60] George B. Dantzig. On the significance of solving linear programming problems with some integer variables. *Econometrica*, 28(1):30–44, 1960.

[dCaCR04] Maria da Conceição Cunha and Luisa Ribeiro. Tabu search algorithms for water network optimization. *European Journal of Operational Research*, 157(3):746 – 758, 2004.

[dCS12] Annelies de Corte and Kenneth Sörensen. Optimisation of water distribution network design: a critical review. Working Paper 2012016, University of Antwerp, Faculty of Applied Economics, August 2012.

[DG86] Marco A. Duran and Ignacio E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36(3):307–339, 1986.

[DGLT12] Sanjeeb Dash, Oktay Günlük, Andrea Lodi, and Andrea Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012.

[DLWB15] Claudia D'Ambrosio, Andrea Lodi, Sven Wiese, and Cristiana Bragalli. Mathematical programming techniques in water network optimization. *European Journal of Operational Research*, 243(3):774–788, 2015.

[DT06a] George B. Dantzig and Mukund N. Thapa. *Linear programming 1: introduction*. Springer Science & Business Media, 2006.

[DT06b] George B. Dantzig and Mukund N. Thapa. *Linear programming 2: theory and extensions*. Springer Science & Business Media, 2006.

[Elh06] Samir Elhedhli. Service system design with immobile servers, stochastic demand, and congestion. *Manufacturing & Service Operations Management*, 8(1):92–97, 2006.

[EPA] EPANET. http://www.epa.gov/water-research/epanet. Accessed: 2016-01-15.

[FG06] Antonio Frangioni and Claudio Gentile. Perspective cuts for a class of convex 0–1 mixed integer programs. *Mathematical Programming*, 106(2):225–236, 2006.

[FL94] Roger Fletcher and Sven Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical programming*, 66(1-3):327–349, 1994.

[FLS15] Fabio Furini, Ivana Ljubić, and Markus Sinnl. ILP and CP Formulations for the Lazy Bureaucrat Problem. In *Integration of AI and OR Techniques in Constraint Programming*, pages 255–270. Springer, 2015.

[FLT11] Matteo Fischetti, Andrea Lodi, and Andrea Tramontani. On the separation of disjunctive cuts. *Mathematical Programming*, 128(1-2):205–230, 2011.

[FT63] Henry Fisher and Gerald L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling*, 3(2):225–251, 1963.

[Fur14] Kevin Furman. A useful algebraic representation of disjunctive convex sets using the perspective function, 2014. Presented to the MINLP Workshop held at Carnegie Mellon University, Pittsburgh.

[Geo71] Arthur M Geoffrion. Duality in nonlinear programming: a simplified applications-oriented development. *SIAM review*, 13(1):1–37, 1971.

[GHHV12] Ambros Gleixner, Harald Held, Wei Huang, and Stefan Vigerske. Towards globally optimal operation of water supply networks. *Numerical Algebra, Control and Optimization (to appear)*, 2012.

[GKL+11] B. Geißler, O. Kolb, J. Lang, G. Leugering, A. Martin, and A. Morsi. Mixed integer linear models for the optimization of dynamical transport networks. *Mathematical Methods of Operations Research*, 73(3):339–362, 2011.

[GL03] Ignacio E. Grossmann and Sangbum Lee. Generalized convex disjunctive programming: Nonlinear convex hull relaxation. *Computational optimization and applications*, 26(1):83–100, 2003.

[GL12] Oktay Günlük and Jeffrey T. Linderoth. Perspective reformulation and applications. In Jon Lee and Sven Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 61–89. Springer New York, 2012.

[Gle13] Ambros Gleixner, 2013. Private communication.

[GLW07] Oktay Günlük, Jon Lee, and Robert Weismantel. MINLP strengthening for separable convex quadratic transportation-cost UFL. Technical Report RC24213 (W0703-042), IBM Research Division, 2007.

[GMMS12] Björn Geißler, Alexander Martin, Antonio Morsi, and Lars Schewe. Using piecewise linear functions for solving minlps. In Jon Lee and Sven Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 287–314. Springer New York, 2012.

[GMS13] Björn Geißler, Antonio Morsi, and Lars Schewe. A New Algorithm for MINLP Applied to Gas Transport Energy Cost Minimization. In Michael Jünger and Gerhard Reinelt, editors, *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*, pages 321–353. Springer Berlin Heidelberg, 2013.

[GNSK+15] Bissan Ghaddar, Joe Naoum-Sawaya, Akihiro Kishimoto, Nicole Taheri, and Bradley Eck. A lagrangian decomposition approach for the pump scheduling problem in water networks. *European Journal of Operational Research*, 241(2):490 – 501, 2015.

[Gom63] Ralph E. Gomory. An algorithm for integer solutions to linear programs. In Robert L. Graves and Philip Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

[GT13] Ignacio E. Grossmann and Francisco Trespalacios. Systematic modeling of discrete-continuous optimization models through generalized disjunctive programming. *AIChE Journal*, 59(9):3276–3295, 2013.

[GUR] Gurobi Optimizer. http://www.gurobi.com. Accessed: 2016-01-15.

[GZMA05] Mohamed S. Ghidaoui, Ming Zhao, Duncan A. McInnis, and David H. Axworthy. A review of water hammer theory and practice. *Applied Mechanics Reviews*, 58(1):49–76, 2005.

[HBCO12] Hassan Hijazi, Pierre Bonami, Gérard Cornuéjols, and Adam Ouorou. Mixed-integer nonlinear programs featuring "on/off" constraints. *Computational Optimization and Applications*, 52(2):537–558, 2012.

[HCLC99] Ching-Huei Huang, Chuei-Tin Chang, Han-Chern Ling, and Cheng-Chang Chang. A mathematical programming model for water usage and treatment network design. *Industrial & Engineering Chemistry Research*, 38(7):2666–2679, 1999.

[HF15] Jesco Humpola and Armin Fügenschuh. Convex reformulations for solving a nonlinear network design problem. *Computational Optimization and Applications*, 62(3):717–759, 2015.

[HFK15] Jesco Humpola, Armin Fügenschuh, and Thorsten Koch. Valid inequalities for the topology optimization problem in gas network design. *OR Spectrum*, pages 1–35, 2015.

[Hij10] Hassan Hijazi. *Mixed-integer nonlinear optimization approaches for network design in telecommunications*. PhD thesis, Université d'Aix Marseille, 2010.

[HL14] Hassan Hijazi and Leo Liberti. Constraint qualification failure in second-order cone formulations of unbounded disjunctions. Technical report, NICTA, Canberra ACT Australia, 2014.

[HMM57] Alan S. Manne Harry M. Markowitz. On the solution of discrete programming problems. *Econometrica*, 25(1):84–110, 1957.

[Hum14] Jesco Humpola. *Gas network optimization by MINLP*. PhD thesis, Technical University of Berlin, 2014.

[JL84] Robert G. Jeroslow and James K. Lowe. Mathematical Programming at Oberwolfach II. chapter Modelling with integer variables, pages 167–184. Springer Berlin Heidelberg, 1984.

[JSS] Job shop scheduling instances. http://optimizizer.com/TA.php. Accessed: 2016-02-19.

[KAA+11] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M Gleixner, Stefan Heinz, et al. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[KH06] Rainer Kolisch and Soenke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, 174(1):23–37, 2006.

[KHPS15] Thorsten Koch, Benjamin Hiller, Marc E. Pfetsch, and Lars Schewe. *Evaluating gas network capacities*, volume 21 of *MOS-SIAM Series on Optimization*. SIAM, 2015.

[Kıl11] Mustafa Kılınç. *Disjunctive Cutting Planes and Algorithms for Convex Mixed Integer Nonlinear Programming*. PhD thesis, University of Wisconsin-Madison, 2011.

[KL12] Oliver Kolb and Jens Lang. Simulation and continuous optimization. In Alexander Martin, Kathrin Klamroth, Jens Lang, Günter Leugering, Antonio Morsi, Martin Oberlack, Manfred Ostrowski, and Roland Rosen, editors, *Mathematical Optimization of Water Networks*, volume 162 of *International Series of Numerical Mathematics*, pages 17–33. Springer Basel, 2012.

[KLL10] Mustafa Kılınç, Jeffrey T. Linderoth, and James Luedtke. Effective separation of disjunctive cuts for convex mixed integer nonlinear programs. *Optimization Online*, 2010.

[LabOR] DEIS - Operations Research Group Library of Instances. http://www.or.deis.unibo.it/research.html. Accessed: 2016-01-15.

[LBvBW06] Carl D. Laird, Lorenz T. Biegler, and Bart G. van Bloemen Waanders. Mixed-integer approach for obtaining unique solutions in source inversion of water networks. *Journal of Water Resources Planning and Management*, 132(4):242–251, 2006.

[LBvBW07] Carl D. Laird, Lorenz T. Biegler, and Bart G. van Bloemen Waanders. Real-time, large scale optimization of water network systems using a subdomain approach. In Lorenz T. Biegler, Omar Ghattas, Matthias Heinkenschloss, David Keyes, and Bart G. van Bloemen Waanders,

editors, *Real-time PDE-constrained optimization*, chapter 15. SIAM Philadelphia, 2007.

[LG00] Sangbum Lee and Ignacio E. Grossmann. New algorithms for nonlinear generalized disjunctive programming. *Computers & Chemical Engineering*, 24(9):2125–2141, 2000.

[LL11a] Jon Lee and Sven Leyffer. *Mixed integer nonlinear programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*. Springer Science & Business Media, 2011.

[LL11b] Jeffrey T. Linderoth and Andrea Lodi. MILP software. *Wiley encyclopedia of operations research and management science*, 2011.

[Lod10] Andrea Lodi. Mixed integer programming computation. In Michael Jünger, Thomas Liebling, Denis Naddef, Geroge L. Nemhauser, William Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer, 2010.

[LY84] David G. Luenberger and Yinyu Ye. *Linear and nonlinear programming*, volume 2. Springer, 1984.

[MF14] Ruth Misener and Christodoulos A. Floudas. Antigone: algorithms for continuous/integer global optimization of nonlinear equations. *Journal of Global Optimization*, 59(2-3):503–526, 2014.

[MGM12] Antonio Morsi, Björn Geißer, and Alexander Martin. Mixed integer optimization of water supply networks. In Alexander Martin, Kathrin Klamroth, Jens Lang, Günter Leugering, Antonio Morsi, Martin Oberlack, Manfred Ostrowski, and Roland Rosen, editors, *Mathematical Optimization of Water Networks*, volume 162 of *International Series of Numerical Mathematics*, pages 35–54. Springer Basel, 2012.

[MNLib] MINLP Library. http://www.gamsworld.org/minlp/minlplib.htm. Accessed: 2016-01-15.

[Min89] Michel Minoux. Networks synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, 19(3):313–360, 1989.

[MLib] MIPLIB - Mixed Integer Problem Library. http://miplib.zib.de. Accessed: 2016-01-15.

[Mor13] Antonio Morsi. *Solving MINLPs on Loosely-Coupled Networks with Applications in Water and Gas Network Optimization*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2013.

[MPT99] Silvano Martello, David Pisinger, and Paolo Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424, 1999.

[MT90] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, Chichester, UK, 1990.

[MT04] Michela Milano and Michael Trick. *Constraint and integer programming*. Springer, 2004.

[MW84] Thomas L. Magnanti and Richard T. Wong. Network design and transportation planning: Models and algorithms. *Transportation science*, 18(1):1–55, 1984.

[NSGAE15] Joe Naoum-Sawaya, Bissan Ghaddar, Ernesto Arandia, and Bradley Eck. Simulation-optimization approaches for water pump scheduling and pipe replacement problems. *European Journal of Operational Research*, 2015.

[Pfa11] Bernhard Pfahringer. Conjunctive Normal Form. In *Encyclopedia of Machine Learning*, pages 209–210. Springer, 2011.

[PIS] David Pisinger's optimization codes. http://www.diku.dk/ pisinger/codes.html. Accessed: 2016-02-19.

[PR91] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[PT] S. Pilati and E. Todini. La verifica delle reti idrauliche in pressione (the verification of hydraulic networks under pressure). Unpublished.

[QG92] Ignacio Quesada and Ignacio E. Grossmann. An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. *Computers & chemical engineering*, 16(10):937–947, 1992.

[Rag13] Arvind U. Raghunathan. Global optimization of nonlinear network design. *SIAM Journal on Optimization*, 23(1):268–295, 2013.

[RB96] Lewis A. Rossman and Paul F. Boulos. Numerical methods for modeling water quality in distribution systems: A comparison. *Journal of Water Resources planning and management*, 122(2):137–146, 1996.

[RDLM14] Riccardo Rovatti, Claudia D'Ambrosio, Andrea Lodi, and Silvano Martello. Optimistic MILP modeling of non-linear optimization problems. *European Journal of Operational Research*, 239(1):32 – 45, 2014.

[RH99] Wolfgang Rauch and Poul Harremoës. Genetic algorithms in real time control applied to minimize transient pollution from urban wastewater systems. *Water Research*, 33(5):1265 – 1277, 1999.

[SA13] Hanif D. Sherali and Warren P. Adams. *A reformulation-linearization technique for solving discrete and continuous nonconvex problems.* Springer Science & Business Media, 2013.

[Sah14] Nikolaos V. Sahinidis. *BARON 14.3.1: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2014.

[Sch98] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

[SM99] Robert A. Stubbs and Sanjay Mehrotra. A branch-and-cut method for 0-1 mixed convex programming. *Mathematical programming*, 86(3):515–532, 1999.

[STF] Steve Jobs' 2005 Stanford Commencement Address. https://www.youtube.com/watch?v=UF8uR6Z6KLc. Accessed: 2016-03-10.

[Tra13] Andrea Tramontani. Lift-and-project cuts in CPLEX 12.5.1, 2013. Presented to the INFORMS Annual Meeting, Minneapolis.

[TS02] Mohit Tawarmalani and Nikolaos V. Sahinidis. *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications.* Springer Science & Business Media, 2002.

[TS05] M. Tawarmalani and Nikolaos V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103:225–249, 2005.

[VA13] Derek Verleye and El-Houssaine Aghezzaf. Optimising production and distribution operations in large water supply networks: A piecewise

linear optimisation approach. *International Journal of Production Research*, 51(23-24):7170–7189, 2013.

[VG99] Aldo Vecchietti and Ignacio E. Grossmann. LOGMIP: a disjunctive 0–1 non-linear optimizer for process system models. *Computers & chemical engineering*, 23(4):555–565, 1999.

[vH01] Willem-Jan van Hoeve. The alldifferent constraint: A survey. *arXiv preprint cs/0105015*, 2001.

[Vie15a] Juan Pablo Vielma. Extended and Embedding Formulations for MINLP, 2015. Presented to the Oberwolfach Workshop "Mixed-integer Nonlinear Optimization: A Hatchery for Modern Mathematics", Oberwolfach, Germany.

[Vie15b] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57(1):3–57, 2015.

[Vig12] Stefan Vigerske. *Decomposition in multistage stochastic programming and a constraint integer programming approach to mixed-integer nonlinear programming*. PhD thesis, Humboldt-Universität zu Berlin, 2012.

[Vig13a] Stefan Vigerske, 2013. Private communication.

[Vig13b] Stefan Vigerske. Solving MINLPs with SCIP, 2013. Presented to the 21st International Symposium on Mathematical Programming, Berlin, Germany.

[VN09] Juan Pablo Vielma and George L. Nemhauser. Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. *Mathematical Programming*, 128(1):49–72, 2009.

[WB06] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[Wie14] Sven Wiese. Short scientific report of STSM *Nonlinear network design* at Zuse Institute Berlin (ZIB). 2014.

[XPR] FICO Xpress Optimization Suite. http://www.fico.com/en/products/fico-xpress-optimization-suite. Accessed: 2016-01-15.