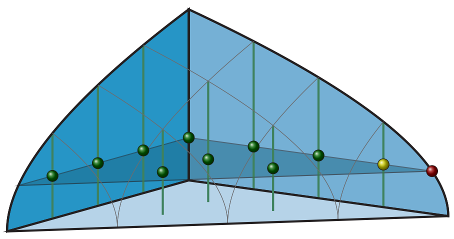
Heuristic algorithms in global MINLP solvers

vorgelegt von
Dipl.-Math. Timo Berthold
aus Berlin

Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation



Tag der wissenschaftlichen Aussprache: 06. November 2014

Berlin 2014
D 83

Abstract

In the literature for mixed integer programming, heuristic algorithms (particularly primal heuristics) are often considered as stand-alone procedures; in that context, heuristics are treated as an alternative to solving a problem to proven optimality. This conceals the fact that heuristic algorithms are a fundamental component of state-of-the-art global solvers for mixed integer linear programming (MIP) and mixed integer nonlinear programming (MINLP).

In the present thesis, we focus on this latter aspect; we study heuristic algorithms that are tightly integrated within global MINLP solvers and analyze their impact on the overall solution process. Our contributions comprise generalizations of primal heuristics for MIP towards MINLP as well as novel ideas for MINLP primal heuristics and for heuristic algorithms to take branching decisions and to collect global information in MIP. These are:

▷ *Shift-and-Propagate*, a novel propagation heuristic for MIP that does not require the solution to an LP relaxation,
▷ a generic way to generalize large neighborhood search (LNS) heuristics from MIP to MINLP,
▷ an *Objective Feasibility Pump* heuristic for nonconvex MINLP that uses second-order information and a dynamic selection of rounding procedures,
▷ *RENS*, an LNS start heuristic for MINLP that optimizes over the set of feasible roundings of an LP solution,
▷ *Undercover*, an LNS start heuristic for MINLP that solves a largest sub-MIP of a given MINLP,
▷ *Rapid Learning*, a heuristic algorithm to generate globally valid conflict constraints for MIPs,
▷ *Cloud Branching*, a heuristic algorithm that exploits dual degeneracy to reduce the number of candidates for branching variable selection.

Additionally, we propose a new performance measure, the *primal integral*, that captures the benefits of primal heuristics better than traditional methods. In our computational study, we compare the performance of the MIP and MINLP solver SCIP with and without primal heuristics on six test sets with altogether 983 instances from academic and industrial sources, including our project partners ForNe, SAP, and Siemens. We observe that heuristics improve the solver performance regarding all measures that we used – by different orders of magnitude. We further see that the harder a problem is to solve to global optimality, the more important the deployment of primal heuristics becomes.

The algorithms presented in this thesis are available in source code as part of the solver SCIP, of which the author has been a main developer for the last years. Methods described in this thesis have also been re-implemented within several commercial and noncommercial MIP and MINLP software packages, including Bonmin, cbc, Cplex, Gams, Sulum, and Xpress.

Zusammenfassung

Heuristische Methoden, insbesondere Primalheuristiken, werden in der Literatur oft als autarke Verfahren betrachtet und erscheinen somit als Gegenentwurf dazu, ein Problem nachweisbar optimal zu lösen. Diese Darstellung vernachlässigt jedoch, dass heuristische Algorithmen ein elementarer Bestandteil moderner MIP- und MINLP-Löser sind.

Die vorliegende Arbeit befasst sich mit ebenjenem Aspekt und behandelt heuristische Verfahren, welche unmittelbar in globale MINLP-Löser integriert sind und studiert deren Einfluss auf den Gesamtlösungsverlauf. Unsere Beiträge umfassen dabei sowohl Verallgemeinerungen von MIP-Heuristiken auf MINLP, als auch neuartige Primalheuristiken für MINLP, sowie allgemeinere heuristische Verfahren zum Branching und zur Gewinnung globaler Information beim Lösen von MIPs. Im einzelnen stellen wir vor:

▷ *Shift-and-Propagate*, eine neue Propagierungsheuristik für MIP, die nicht vom Lösen einer LP-Relaxierung abhängt,
▷ ein generisches Verfahren zur Verallgemeinerung von Nachbarschaftssuchmethoden von MIP auf MINLP,
▷ diverse Erweiterungen einer *Zielfunktions-Feasibility-Pump* für nichtkonvexes MINLP, unter anderem durch das Verwenden zweiter Ableitungen,
▷ *RENS*, eine nachbarschaftsbasierte Startheuristik für MINLP, die über den Raum aller zulässigen Rundungen einer Relaxierungslösung optimiert,
▷ *Undercover*, eine Primalheuristik für MINLP, die ein größtes Teil-MIP eines gegebenen nichtlinearen Problems löst,
▷ *Rapid Learning*, ein heuristisches Verfahren um global gültige Konflikte für ganzzahlige Programme zu generieren,
▷ *Cloud Branching*, ein heuristischer Algorithmus, der durch Ausnutzung dualer Degeneriertheit die Menge der Branching-Kandidaten reduziert.

Darüber hinaus führen wir ein neuartiges Maß ein, das *primale Integral*, um die Auswirkung primaler Heuristiken besser zu quantifizieren. In unserer Rechenstudie vergleichen wir den Lösungsverlauf des MIP- und MINLP-Lösers SCIP, mit und ohne Primalheuristiken, auf sechs Gruppen von Probleminstanzen. Diese umfassen sowohl akademische Benchmarks als auch Industrie-Instanzen unserer Projektpartner ForNe, SAP und Siemens. Wir stellen fest, dass Heuristiken das Verhalten in Bezug auf sämtliche Maße, die wir betrachten, verbessern. Außerdem zeigen wir, dass der Einsatz von Primalheuristiken umso wichtiger wird, je schwieriger eine Probleminstanz global optimal zu lösen ist.

Implementierungen der in dieser Arbeit präsentierten Algorithmen sind als Quellcode innerhalb des MINLP-Lösers SCIP verfügbar, von welchem der Autor in den vergangenen Jahren einer der Hauptentwickler war. Des Weiteren wurden viele der in dieser Arbeit vorgestellten Verfahren in diverse kommerzielle sowie akademische Softwarepakete integriert, unter anderem in Bonmin, cbc, Cplex, Gams, Sulum und Xpress.

Acknowledgments

Oh boy, seven years a PhD student? Well, this can be a very rewarding and exciting time, especially when you are working in such an amazing environment as ZIB – which will hopefully remain that way for a long time. The credit for this and my personal thanks go to Martin Grötschel for creating such an inspiring atmosphere; it was a pleasure to be part of this institute.

Thanks to Thorsten Koch for leaving me and the other members of his group the freedom to make our own decisions and design projects in the way that we wanted them to be. I have always understood ZIB, the "Department of Optimization" and the "IP group" as a cornucopia of opportunities and appreciate all the experience I gained during my time here: direct contact to R&D groups of blue chip companies, traveling to dozens of conferences, cooperation with universities all over the world, a four month stay in Australia, frequently having leading experts in our field as visitors, founding and running a spin-off company, et cetera. I've learned so much in so many different ways!

When I first came to ZIB, I met Tobias Achterberg who gave me a ten minute primer on "How a MIP solver really works" and told me that he was looking for a student assistant to improve the heuristic behavior of SCIP by implementing a Feasibility Pump and Local Branching. My first thought was: "That sounds like a job that gives rise to a great topic for a Diploma thesis." And it did. Even more fortunate, the first papers that Tobi gave to me – and thereby the first research papers I read in my life – were works by Matteo Fischetti and Andrea Lodi, hence fun to read and very accessible. Andrea, your work has always been an inspiration to me and I really appreciate that you are part of my PhD thesis committee.

Back to my timeline at ZIB: I spent two years with the best mentor I ever had, getting taught all the secrets of computational MIP. Tobi, thanks for everything, without you I would simply not be where I am today. Then suddenly, the time came when the SCIP mastermind moved on to a new challenge at ILOG, by coincidence just at the time when I finished my Diploma. Taking over the development of a big academic software package that was someone else's PhD project can be a burden. In our case, it was a blessing. Not only was the code in an excellent shape, it had such a deep design that we could easily extend it far beyond being "only" a MIP solver. "We" refers to the team of PhD students at ZIB who were responsible for the SCIP project, in particular Stefan Heinz who started at ZIB around the same time as I did and later Gerald Gamrath, Ambros Gleixner, and Stefan Vigerske. Joyful, joyful time to work with you. The SCIP Optimization Suite, how-

ever, is more than only SCIP; it also includes ZIMPL, SOPLEX, GCG, UG. Thanks to the people working on all those and other SCIP-related projects, in particular Marco Lübbecke and his group, Matthias Miltenberger, Marc E. Pfetsch and his group, Yuji Shinano, Dieter Weninger, and Kati Wolter.

I am indebted to our great Master's students Gregor Hendel and Michael Winkler. You guys have always been a big support. Right on! I further thank all colleagues from the department for the collegial and cordial working environment, foremost at the daily coffee breaks, lunches and several BBQs and excursions over the years.

Thanks to Marc E. Pfetsch who led our working group for the first two years: you have taught me a lot about the academic community and always kept me grounded. I thank Günter M. Ziegler for being my BMS mentor and always having good advice for me. Do you, Günter and Marc, actually know that your "Linear Algebra" course, that I took during my very first semester, contributed a lot to my decision to major in Discrete Mathematics?

Furthermore, I am very grateful to the numerous people proofreading this thesis and the papers that are incorporated into it. In particular, this includes Gerald Gamrath, Ralf Lenz, and Jonas Schweiger, all of whom read a couple of chapters. Special thanks to Christina Burt and Stephen Maher who had the unrewarding job of fighting my language deficiencies. Sorry for hard times having gived you by me much!

Thanks to all my friends at NICTA and the University of Melbourne for the great times that we spent together, in particular to Mark Wallace and Peter Stuckey for facilitating my down under trips. I am grateful to the DFG Research Center MATHEON *Mathematics for key technologies* which funded my research throughout my PhD time and the *Berlin Mathematical School* for their support.

I am indebted to the people maintaining the computational infrastructure at ZIB, especially Andreas Löbel. Further, I am deeply impressed by our library, in persona Regina Kossick. Whenever I ordered an article, the (perceived) response time was about ten minutes. I still have no clue to how that is even possible...

Alex and Svenja, thanks for helping me to free up some time for my PhD project by babysitting our daughter dozens of times, and thanks to you and all my other friends for helping me to free up my mind every now and then. I acknowledge Michaela Keinert for her terrific drawing on page 200. Special thanks to my buddy Annegret Dix for battling through our undergrad studies together – I owe you a lot.

Most of all, I have to thank my family: my amazing wife Frieda and my two wonderful children Annabell and Jonathan. You had my back whenever I needed you, you brought me up when I was down, you give a meaning to my life.

Danke!

Contents

1. Introduction

"This problem is $\mathcal{NP}$-complete, so it cannot be solved exactly; thus, we had to use a heuristic instead." We read and hear sentences like this way too often, and I do not know which part of that argument to correct first. First, and probably the least worst, is that a problem class being in $\mathcal{NP}$ actually implies that it *can be solved*, it just may take a long time. *It may!* Second, looking at all the industrial applications I have been faced with during my time at Zuse Institute Berlin and that the research community is discussing at numerous operations research conferences every year, the sound conclusion is that $\mathcal{NP}$-hard optimization problems *have to be solved* to optimality. Even more importantly, very often they can be solved within reasonable time! The good news is that today there is a variety of solver software available for $\mathcal{NP}$-hard problems. The third slip is more between the lines: sometimes, we tend to think of heuristics and complete procedures as a dichotomy rather than as a symbiosis. It is the goal of this thesis to quash this preconception. Not only might heuristics use complete methods as subroutines,[1] global solvers for optimization problems are full of heuristic algorithms.[2]

One of the major vendors for mixed integer linear programming (MIP) software actually promotes its product by emphasizing the importance of heuristics: "Our advanced MIP heuristics for quickly finding feasible solutions often produce good quality solutions where other solvers fall flat, leading to some of our biggest wins vs. the competition."[3] The past twenty years have witnessed a substantial progress in the development of MIP software packages. As a consequence, state-of-the-art MIP solvers, commercial and non-commercial, are nowadays capable of solving a variety of different types of MIP instances arising from real-world applications within reasonable time, and mixed integer linear programming has become a standard technique in planning and logistics.

For mixed integer nonlinear programming (MINLP), today's situation is comparable to that of MIP in the early nineties. Techniques to efficiently solve MINLPs are known in principle. Most of them have been developed

[1] The famous Christofides heuristic [Chr76] for the metric TSPs, for instance, is based on computing a minimum spanning tree and a minimum-weight perfect matching.

[2] We use the terminology of the global optimization community here: A global solver is a code that solves each instance of a certain class of optimization problems to proven global optimality (or proves that no such optimum exists). Synonymously, we will use the term *complete algorithm* to distinguish global solution procedures from heuristic approaches.

[3] `http://www.gurobi.com/products/gurobi-optimizer/features-and-benefits`, accessed 20.06.2013

in academic proof-of-concept implementations, but are not yet well-tested for a broad range of industrial problems, in particular not for large-scale applications. MINLP is not only empirically more challenging than MIP, but also theoretically harder, as can be seen from some complexity results that hold even when restricting the nonlinearity to quadratic functions:

- ▷ Mixed integer quadratic programming (MIQP) is undecidable [Mat70], whereas MIP is "only" $\mathcal{NP}$-complete [Kar72, BT76].
- ▷ Solving the continuous relaxation of an MIQP (i.e., a quadratic program) is $\mathcal{NP}$-hard [Sah74], whereas for MIP, solving the LP relaxation is possible in polynomial time [Kha79].
- ▷ Unconstrained 0-1 quadratic programming is $\mathcal{NP}$-hard [GJ79], whereas a linear objective function can be optimized over $\{0,1\}^n$ in $\mathcal{O}(n)$.

On the practical side, there are many real-world applications that are inherently nonlinear and need to by tackled by MINLP, see, e.g., [GS02]. This induces a growing need for MINLP algorithms that are at the same time innovative from a theoretical perspective and efficient in practice, properly integrated into a common, stable software implementation that targets solving instances of relevant size from a broad field of applications. There is reasonable hope that the success story of computational MIP can be repeated for the nonlinear case within the next ten to fifteen years.

We see this thesis as one step of this long journey. We present new ideas for heuristic algorithms that are conceived with a special focus on being employed within a global MINLP solver. This includes for instance a new filtering framework for branching heuristics and a heuristic to learn global bounds and conflict constraints. The main part of this thesis, however, deals with *primal heuristics*.

Primal heuristics are algorithms that try to find feasible solutions of good quality for a given optimization problem within a reasonably short amount of time.[4] There is typically no guarantee that they will find any solution, let alone an optimal one.

For mixed integer linear programs (MIPs) it is well known that general-purpose primal heuristics are able to find solutions with a small optimality gap for a wide range of problems; they have become a fundamental ingredient of state-of-the-art MIP solvers [Ber06, BFG$^+$00, LPT$^+$09]. Fischetti and Lodi state that they are "among the most crucial improvements [for MIP] over the last ten years" [FL10]. For mixed integer nonlinear programming, the last five years have shown a rising interest in the research community for general-purpose primal heuristics [BG10, BG14, BHPV11, BCLM09, BG12, DFLL10, DFLL12, LMN11, NB12, NBL08]. Within this thesis, we discuss new contributions to this area of research. The contributions of the present

[4]We will use the term *primal* heuristic exclusively for procedures that search (heuristically) for *feasible solutions*. In all other cases, we will speak of *heuristic algorithms*.

thesis comprise generalizations of MIP heuristics towards MINLP as well as genuinely new ideas for MINLP heuristics and for MIP heuristics.

Previous publications mostly considered primal heuristics as standalone algorithms. In contrast, we focus on heuristic procedures that are conceptualized to be tightly integrated within a global MINLP solver. This often leads to very different design decisions, as we will show at several points of this thesis.

Contributions and structure of this thesis.

In the following chapters, we will introduce general concepts and standard notation of (computational) MINLP, give an overview of available software and discuss the general embedding of heuristic algorithms within a global MINLP solver. Afterwards, we will first reason about how the performance of primal heuristics inside a global solver can be measured. In the following, we proceed from computationally cheap to more involved primal heuristics, starting with rounding and propagation heuristics for MIP, going via feasibility pump algorithms to large neighborhood search (LNS). After two chapters on branching heuristics and a heuristic algorithm for generating conflict constraints, we present our overall computational results. In detail, the content of this thesis is structured as follows.

In Chapter 3, we introduce a new performance measure to evaluate mathematical programming software, the *primal integral*. This measure takes the development of the incumbent solution over time into account, thereby favoring finding good solutions early. We show that for SCIP and CPLEX the primal integral changes by a factor of two when disabling primal heuristics.

In Chapter 4, we introduce *Shift-and-Propagate*, a new pre-root primal heuristic for MIP that does not require a feasible LP solution as a starting point. We show that combining five existing rounding and improvement heuristics with Shift-and-Propagate increases the number of instances for which a feasible solution is found by 60 %. Additionally, we find that the average primal gap is significantly reduced.

In Chapter 5, we introduce three novel contributions towards a *Feasibility Pump for nonconvex MINLP*: using a hierarchy of MIP solvers, employing a distance function that takes into account second-order information, and generating valid cutting planes also for the nonconvex part of the given MINLP. We show that the dynamic use of different MIP solving strategies and incorporating the Hessian of the Lagrangian into the auxiliary MIPs improves the number and the quality of feasible solutions found. The former strategy additionally decreases the average running time.

In Chapter 6, we introduce a generic way of generalizing *large neighborhood search heuristics from MIP to MINLP*. This includes the first presentation of nonlinear versions of Crossover and the DINS heuristic. We show that not only the generalized LNS heuristics increase the quality of the best feasible solution after root node computation, but improve the behavior of the solver for the global search.

In Chapter 7, we introduce *RENS*, a large neighborhood search algorithm that optimizes over the set of feasible roundings of a relaxation solution. We analyze how the roundability is affected by different relaxations, the usage of cutting planes and the fractionality of the solution. We show that RENS significantly improves the primal bound at the root node. Additionally, a version of SCIP that applies RENS frequently during search gives an improvement of 8 % in running time for MINLP.

In Chapter 8, we introduce *Undercover*, an LNS start heuristic for MINLP that explores linear subproblems induced by a minimum vertex cover. To this end, we define the notion of a minimum cover of an MINLP and demonstrate that MINLPs typically allow for very small covers. We show that for MIQCPs, applying Undercover at the root node significantly improved the overall performance of SCIP by 15 % and even by 32 % for hard instances.

In Chapter 9, we introduce *Rapid Learning*, a heuristic algorithm to learn valid conflicts and bound reductions via a partial CP search. We show that it is particularly useful for IPs and BPs that are infeasible or for problems where finding good solutions is much harder than proving optimality. For such problem instances, the mean running time decreases by 25 % when using Rapid Learning.

In Chapter 10, we introduce *Cloud Branching*, a framework for branching heuristics to exploit the knowledge of alternative relaxation solutions. We show that a version of Full Strong Branching that exploits the idea of Cloud Branching is about 30 % faster than default Full Strong Branching on a standard MIP test set with high dual degeneracy.

Chapter 11 constitutes our main computational study. We conduct experiments on three general, academic benchmark sets for MIP, MIQCP, and MINLP, namely, the MIPLIB, the GLOMIQO test set, and the MINLPLIB. Further, we present results for real-world applications from projects with industry partners who the author was associated with during his time at ZIB. These applications cover MIP (a project with SAP), MIQCP (SIEMENS), and MINLP (FORNE/Open Grid Europe). It turns out that primal heuristics consistently help to significantly improve performance on all six test sets. We observe that primal heuristics have only few impact on easy instances, but are crucial for solving hard optimization problems.

Figure 1.1 categorizes the content of Chapters 3–10 w.r.t. different criteria:

▷ The application: Are the presented methods designed for linear or nonlinear problems, even-handedly for both, or primarily for one type of problems with possible application to the other?

▷ The technique: Are the described methods primal heuristics or general heuristic algorithms, e.g., to choose branching variables, tighten bounds, generate conflicts. . . ?

▷ The innovation: Are the new contributions of each chapter extensions of existing methods or novel concepts? Further: Does the chapter contain an individual literature overview on the covered subject?

The implementations of all algorithms[5] that are presented in this thesis are publicly available as part of the constraint integer programming framework SCIP [Ach09, Sci] of which the author has been a main developer for the last seven years. Since SCIP is freely available in source for academic and non-commercial purposes, the C code of the implementations is readily available for any researcher who wants to analyze, modify, amend, or improve it. We aim to provide general descriptions of the procedures so that they can be replicated using other implementations of MIP and MINLP solvers.

Publications.

Significant parts of this thesis have been published in refereed conference proceedings and international journals, or have been submitted for publication. These include the following:

- ▷ Chapter 3 has been published in Operations Research Letters [Ber13].
- ▷ Chapter 4 is joint work with my master student Gregor Hendel and has been published in Journal of Heuristics [BH15].
- ▷ Chapter 5 is joint work with Pietro Belotti from Clemson University (now Fair Isaac Europe Ltd), a publication is in preparation.
- ▷ Chapter 6 is loosely based on joint work with Stefan Heinz from Zuse Institute Berlin (now Fair Isaac Europe Ltd), Marc E. Pfetsch from Technische Universität Carolo-Wilhelmina zu Braunschweig (now Technische Universität Darmstadt), and Stefan Vigerske from the Humboldt-Universität zu Berlin (now GAMS Software GmbH) [BHPV11].
- ▷ Chapter 7 has been published in Mathematical Programming Computation [Ber14].
- ▷ Chapter 8 is joint work with Ambros M. Gleixner from Zuse Institute Berlin and has been published in Mathematical Programming [BG14].
- ▷ Chapter 9 is based on joint work with Thibaut Feydy and Peter J. Stuckey from the University of Melbourne and has been published in the proceedings of CPAIOR2010, LNCS 6140 [BFS10].
- ▷ Chapter 10 is joint work with Domenico Salvagnin from the Università degli Studi di Padova and has been published in the proceedings of CPAIOR2013, LNCS 7874 [BS13].

[5]Except the nonlinear Feasibility Pump of Chapter 5 which has been implemented in COUENNE [BLL$^+$09]

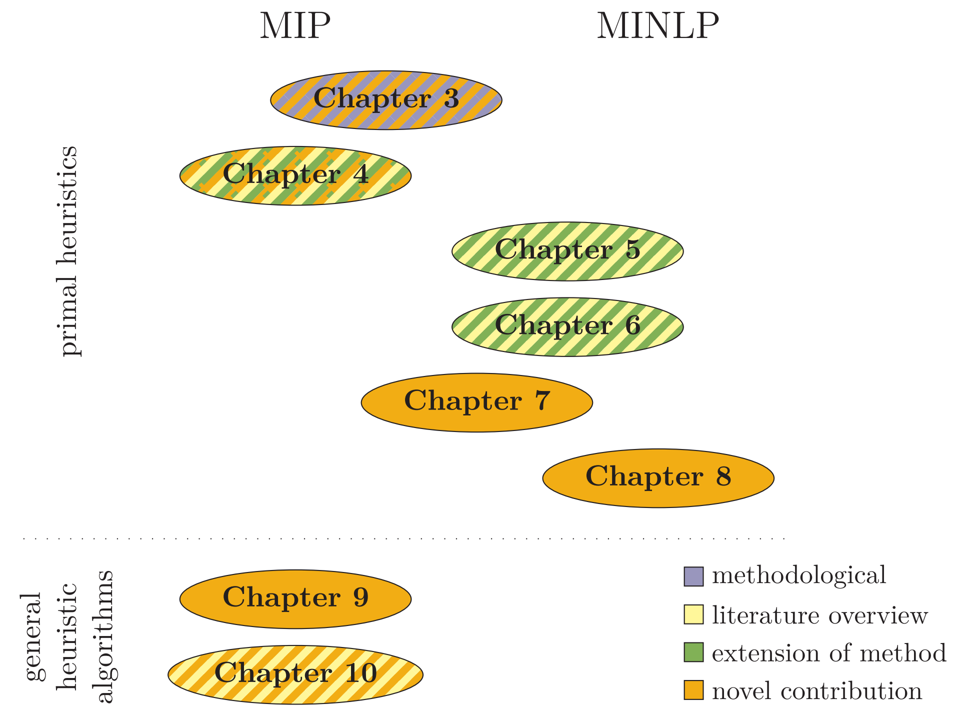


Figure 1.1.: Categorization of the chapters of this thesis by three different characteristics.

2. Concepts

In recent years, substantial progress has been made in the solvability of generic *mixed integer linear programs (MIPs)* [Ach07a, BFG$^+$00, LPT$^+$09, Lod10]. Furthermore, it has been shown that successful MIP solving techniques can often be extended to the more general case of *mixed integer nonlinear programs (MINLPs)* [ALL10, BLL$^+$09, BBC$^+$08]. Analogously, several authors have shown that an integrated approach of *constraint programming (CP)* and mixed integer programming (MIP) can help to solve optimization problems that were intractable with either of the two methods alone. For an overview see [Ach09, Hoo07, YAH10].

This chapter gives a brief introduction into the terminology and the solution strategies for the classes of mathematical optimization problems that we consider in this thesis. In Section 2.1, we provide basic definitions that we use throughout the remaining chapters. Section 2.2 introduces the key ideas efficiently solving MIPs and MINLPs to proven optimality. Further, it contains an overview of available software for solving mixed integer linear and nonlinear optimization problems. Finally, Section 2.3 discusses how these solvers employ heuristic algorithms.

2.1. Mixed integer nonlinear programming

Optimization problems that simultaneously feature nonlinear functions as constraints and integrality requirements for the variables are arguably among the most challenging problems in mathematical programming. This section introduces these *MINLPs* and their sub-classes.

Definition 2.1 (MINLP)**.** *A* mixed integer nonlinear program (MINLP) *is an optimization problem of the form*

$$\begin{aligned} \min \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & g_i(x) \leqslant 0 && \textit{for all } i \in \mathcal{M} \\ & l_j \leqslant x_j \leqslant u_j && \textit{for all } j \in \mathcal{N} \\ & x_j \in \mathbb{Z} && \textit{for all } j \in \mathcal{I}, \end{aligned} \tag{2.1}$$

where $\mathcal{I} \subseteq \mathcal{N} := \{1, \dots, n\}$ is the index set of the integer variables, $c \in \mathbb{R}^n$, $g_i : \mathbb{R}^n \to \mathbb{R}$ for $i \in \mathcal{M} := \{1, \dots, m\}$, and $l \in (\mathbb{R} \cup \{-\infty\})^n$, $u \in (\mathbb{R} \cup \{+\infty\})^n$ are lower and upper bounds on the variables, respectively.

We call $c^{\mathsf{T}} x$ the *objective function* and the $g_i(x)$ the *constraint functions* of (2.1). Functions which can be represented as $a^{\mathsf{T}} x + d$ with $a \in \mathbb{R}^n$, $d \in \mathbb{R}$ are called *linear functions*. Functions which can be represented as $x^\mathsf{T}Qx+a^\mathsf{T}x+d$ with $Q \in \mathbb{R}^{n\times n}, a \in \mathbb{R}^n, d \in \mathbb{R}$ are called *quadratic functions*. Without loss of generality, we assume that Q is symmetric.

Note that the format given in Definition 2.1 is very general. First, maximization problems can be transformed to minimization problems by multiplying all objective function coefficients by -1. Similarly, "$\geqslant$" constraints can be multiplied by -1 to obtain "$\leqslant$" constraints. Equations can be replaced by two opposite inequalities. A nonlinear objective function can always be reformulated by introducing one additional variable and constraint. We assume without loss of generality that $l_j \leqslant u_j$ for all $j \in \mathcal{N}$ and $l_j, u_j \in \mathbb{Z}$ for all $j \in \mathcal{I}$.

Let $\mathcal{B} := \{j \in \mathcal{I} \mid l_j = 0, u_j = 1\}$. We call $\{x_j \mid j \in \mathcal{B}\}$ the set of *binary variables*, $\{x_j \mid j \in \mathcal{I}\}$ the set of *integer variables*, $\{x_j \mid j \in \mathcal{I} \setminus \mathcal{B}\}$ the set of *general integer variables*, and $\{x_j \mid j \in \mathcal{N} \setminus \mathcal{I}\}$ the set of *continuous variables*. The interval $[l_j, u_j]$ or the set $\{l_j, \ldots, u_j\}$ is called the the *domain* of a continuous or integer variable x_j, respectively.

We denote by $[l, u] := \{x \in (\mathbb{R} \cup \pm\infty)^n \mid l_j \leqslant x_j \leqslant u_j \text{ for all } j \in \mathcal{N}\}$ the Cartesian product of the variable domains $[l_j, u_j]$, which is sometimes referred to as the *domain box*. For a given MINLP P, the set

$$\tilde{\mathcal{X}}(P) := \{x \in \mathbb{R}^n \mid g_i(x) \leqslant 0 \quad \text{for all } i \in \mathcal{M}, x \in [l,u], x_j \in \mathbb{Z} \text{ for all } j \in \mathcal{I}\}$$

is called the set of *feasible solutions* of the MINLP. Let $c^\star \in \mathbb{R} \cup \pm\infty$ with

$$c^\star := \inf\{c^\mathsf{T}x \mid x \in \tilde{\mathcal{X}}\}.$$

If $c^\star = -\infty$, we call P *unbounded*; if $c^\star = +\infty$, we call it *infeasible*. If $c^\star$ is finite, we call it the *optimal solution value* of P. A solution $x \in \tilde{\mathcal{X}}$ is called an *optimal solution* if and only if $c^\mathsf{T}x = c^\star$. If $c = 0$, we call P a *feasibility problem*.

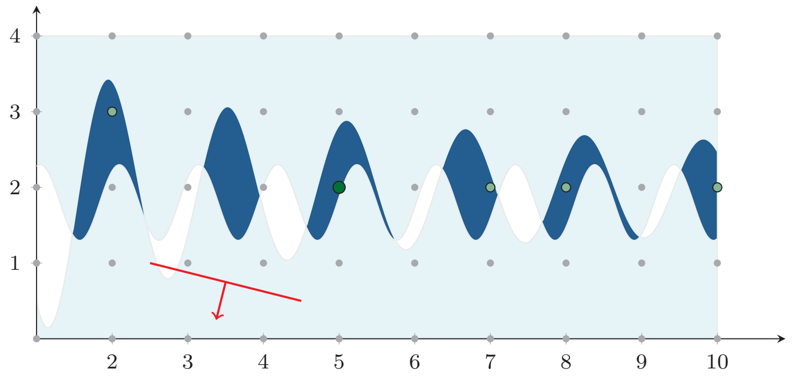


Figure 2.1.: A nonconvex MINLP, the optimal solution is $(5, 2)$.

Example 2.2 (MINLP with two variables and two constraints). *Consider* *the following MINLP:*

$$\begin{aligned}
\min \quad & x_1 + 4x_2 \\
\text{s.t.} \quad & \frac{\cos(6x_1)}{2} - x_2 - 1.8 \leqslant 0, \\
& -\frac{2\sin(4x_1)}{\sqrt{x}} + x_2 - 2 \leqslant 0, \\
& 1 \leqslant x_1 \leqslant 10, \\
& 0 \leqslant x_2 \leqslant 4, \\
& x_1, x_2 \in \mathbb{Z}.
\end{aligned} \tag{2.2}$$

This example is illustrated in Figure 2.1. The light blue areas correspond to regions of the x_1-x_2-plane which fulfill exactly one of the two nonlinear constraints, the dark blue areas correspond to regions which fulfill both nonlinear constraints (the so-called NLP relaxation*, see below). The red line shows the objective function with its normal vector. The small gray points represent the integer lattice $\mathbb{Z}^2 \cap ([1, 10] \times [0, 4])$. The green points illustrate the set of feasible solutions; the dark green point $(5, 2)$ is the unique optimal solution.*

There are many subclasses of MINLP. In this thesis, we particularly focus on the following:

- ▷ If all constraint functions g_i are quadratic, problem (2.1) is called a *mixed integer quadratically constrained program (MIQCP)*.
- ▷ If $\mathcal{I} = \emptyset$, problem (2.1) is called a *nonlinear program (NLP)*.
- ▷ If all constraint functions g_i are quadratic and $\mathcal{I} = \emptyset$, problem (2.1) is called a *quadratically constrained program (QCP)*.
- ▷ If all constraint functions g_i are linear, problem (2.1) is called a *mixed integer program (MIP)*, see Definition 2.3.
- ▷ If all constraint functions $g_i, i \in \mathcal{M}$ are convex on $[l, u]$, we call (2.1) a *convex* MINLP.

For disambiguity, general MINLPs are sometimes equally referred to as *nonconvex MINLPs*. Note that by our definitions, a convex NLP always has a convex feasible set, but a convex feasible set does not necessarily imply that it is defined by a convex NLP.

For nonconvex NLPs, the notion of local optima is common. Let $\tilde{x} \in \tilde{\mathcal{X}}$ be a feasible solution of an NLP. If there exists an $\epsilon > 0$ such that $c^\mathsf{T}\tilde{x} \leqslant c^\mathsf{T}x$ for all $x \in \tilde{\mathcal{X}}$ with $\|\tilde{x} - x\| < \epsilon$, then $\tilde{x}$ is a *local optimum*. An optimal solution $x^\star$, in the above sense that $c^\mathsf{T}x^\star = c^\star$, is then synonymously called a *global optimum*. Obviously, every global optimum is also a local optimum. For convex NLPs, every local optimum is also a global optimum.

The standard way to define MIPs is in matrix and vector notion. The chapters of this thesis which solely relate to techniques for linear optimization problems will refer to this notion. We (re-)define a MIP as follows.

Definition 2.3 (mixed integer program)**.** *Let* $m, n \in \mathbb{Z}_{\geqslant 0}$*. Given a matrix* $A \in \mathbb{R}^{m \times n}$*, a right-hand-side vector* $b \in \mathbb{R}^m$*, an objective function vector* $c \in \mathbb{R}^n$*, a lower and an upper bound vector* $l \in (\mathbb{R} \cup \{-\infty\})^n$*,* $u \in (\mathbb{R} \cup \{+\infty\})^n$ *and a subset* $\mathcal{I} \subseteq \mathcal{N} = \{1, \dots, n\}$*, the corresponding* mixed integer program (MIP) *is given by*

$$\begin{array}{lrl} \min & c^{\mathsf{T}} x & \\ s.t. & Ax \leqslant b & \\ & l_j \leqslant x_j \leqslant u_j & \text{for all } j \in \mathcal{N} \\ & x_j \in \mathbb{R} & \text{for all } j \in \mathcal{N} \setminus \mathcal{I} \\ & x_j \in \mathbb{Z} & \text{for all } j \in \mathcal{I}. \end{array} \tag{2.3}$$

Different variable types play different roles in MIP modeling and solution approaches. In particular, integer variables that only allow for values 0 and 1 are typically used to model logical decisions (“yes/no”,“on/off”). Many MIP techniques such as probing [Sav94], knapsack cover cuts [Bal75, HJP75, Wol75], or OCTANE [BCD$^+$01] work only for 0-1 variables.

Mixed integer programs can be categorized by the classes of variables that are part of their formulation:

▷ If $\mathcal{N} = \mathcal{I}$, problem (2.3) is called a *(pure) integer program (IP).*

▷ If $\mathcal{N} = \mathcal{B}$, problem (2.3) is called a *(pure) binary program (BP).*

▷ If $\mathcal{I} = \mathcal{B}$, problem (2.3) is called a *mixed binary program (MBP)* or a *0-1 mixed integer program.*

▷ If $\mathcal{I} = \emptyset$, problem (2.3) is called a *linear program (LP).*

With a slight abuse of notation, we will use the abbreviation MINLP for the term “mixed integer nonlinear programming” as well as for the term “mixed integer nonlinear program” throughout this thesis. The same holds for all other problem classes. Furthermore, we will sometimes use “a variable $j \in \mathcal{N}$” synonymously to “a variable x_j”.

One of the most striking techniques in MINLP and MIP is the use of relaxations to provide proven lower bounds on the optimal solution of a given problem instance. The NLP relaxation of an MINLP arises by omitting the integrality constraints:

Definition 2.4 (NLP and LP relaxation)**.** *Given an MINLP problem* P *of the form* (2.1)*, the NLP*

$$\begin{array}{lll} \min & c^{\mathsf{T}} x & \\ \text{s.t.} & g_i(x) \leqslant 0 & \text{for all } i \in \mathcal{M} \\ & l_j \leqslant x_j \leqslant u_j & \text{for all } j \in \mathcal{N} \\ & x_j \in \mathbb{R} & \text{for all } j \in \mathcal{N}, \end{array} \tag{2.4}$$

is called the NLP relaxation of P*. The LP relaxation of a MIP is defined analogously.*

By knowing the optimal objective function values of a relaxation and of some (heuristic) solution, we get a *dual bound* and a *primal bound*, respectively, for the optimal solution value of an MINLP. To measure the quality of these bounds w.r.t. the optimal solution value or w.r.t. each other, we use the notion of gap functions.

Definition 2.5 (primal gap)**.** *Let $\tilde{x}$ be a solution for an MINLP, and $x^\star$ be an optimal (or best known) solution for that MINLP. We define the* primal gap $\gamma^p \in [0,1]$ *of $\tilde{x}$ as:*

$$\gamma^p(\tilde{x}) := \begin{cases} 0, & \textit{if } c^\mathsf{T} x^\star = c^\mathsf{T} \tilde{x} = 0, \\ 1, & \textit{if } c^\mathsf{T} x^\star \cdot c^\mathsf{T} \tilde{x} < 0, \\ \frac{\left|c^\mathsf{T} x^\star - c^\mathsf{T} \tilde{x}\right|}{\max\{\left|c^\mathsf{T} x^\star\right|,\ \left|c^\mathsf{T} \tilde{x}\right|\}}, & \textit{otherwise.} \end{cases}$$

Definition 2.6 (dual gap)**.** *Let $\bar{x}$ be an optimal solution of a relaxation of an MINLP, and $x^\star$ be an optimal (or best known) solution for that MINLP. Analogously to the primal gap, we define the* dual gap $\gamma^d \in [0,1]$ *of $\bar{x}$ as:*

$$\gamma^d(\bar{x}) := \begin{cases} 0, & \textit{if } c^\mathsf{T} x^\star = c^\mathsf{T} \bar{x} = 0, \\ 1, & \textit{if } c^\mathsf{T} x^\star \cdot c^\mathsf{T} \bar{x} < 0, \\ \frac{\left|c^\mathsf{T} x^\star - c^\mathsf{T} \bar{x}\right|}{\max\{\left|c^\mathsf{T} x^\star\right|,\ \left|c^\mathsf{T} \bar{x}\right|\}}, & \textit{otherwise.} \end{cases}$$

The *primal-dual gap* is a typical information given by MIP and MINLP solvers during runtime. It is often referred to as *optimality gap*, a name that might be considered slightly misleading since it does explicitly not describe the gap of any of the bounds to optimality, but is, rather, a worst case estimation.

Definition 2.7 (primal-dual gap)**.** *Let $\bar{x}$ be an optimal solution of a relaxation of an MINLP and $\tilde{x}$ be a feasible solution for that MINLP. We define the* primal-dual gap $\gamma^{pd} \in \mathbb{R}_{\geq 0}$ *of $\bar{x}$ and $\tilde{x}$ as:*

$$\gamma^{pd}(\bar{x}, \tilde{x}) := \begin{cases} 0, & \textit{if } c^\mathsf{T} \bar{x} = c^\mathsf{T} \tilde{x} = 0, \\ \frac{c^\mathsf{T} \tilde{x} - c^\mathsf{T} \bar{x}}{\left|c^\mathsf{T} \bar{x}\right|}, & \textit{if } c^\mathsf{T} \bar{x} \cdot c^\mathsf{T} \tilde{x} > 0, \\ \infty & \textit{otherwise.} \end{cases}$$

Different solvers, however, use different definitions of the primal-dual gap. For this thesis, we chose the definition that is used by SCIP and was introduced in [Ach07b]. The most common version, however, is the one that is used by Cplex.

Definition 2.8 (primal-dual gap (Cplex))**.** *Let $\bar{x}$ be an optimal solution of a relaxation of an MINLP and $\tilde{x}$ be a feasible solution for that MINLP. We define the* Cplex gap $\gamma^{pd}_{\text{cpx}} \in \mathbb{R}_{\geq 0}$ *of $\bar{x}$ and $\tilde{x}$ as:*

$$\gamma^{pd}_{\mathrm{cpx}}(\bar{x},\tilde{x}) := \begin{cases} 0, & \textit{if } c^{\mathsf{T}}x^{\star} = c^{\mathsf{T}}\bar{x} = 0, \\ \frac{|c^{\mathsf{T}}\tilde{x} - c^{\mathsf{T}}\bar{x}|}{\max\{|c^{\mathsf{T}}\bar{x}|,\ |c^{\mathsf{T}}\tilde{x}|\}}, & \textit{otherwise.} \end{cases}$$

The three gap functions from Definitions 2.5–2.7 decrease monotonically when the dual bound increases or the primal bound decreases. The CPLEX gap, however, is increasing when the primal and dual bound have opposite signs. The gap functions given in Definitions 2.5 and 2.6 always attain values between zero and one; for the function in Definition 2.8, this holds unless both bounds have opposite signs (which is a rare case in practice). The primal-dual gap given in Definition 2.7 can attain values greater than 100 %, even when both bounds have the same sign. As a simple example, for a dual bound of one and a primal bound of five, SCIP would report 400 % gap, whereas CPLEX would report 80 % gap.

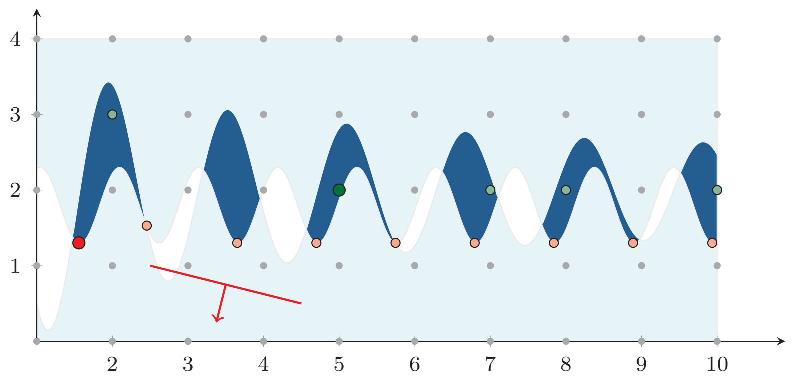


Figure 2.2.: The NLP relaxation of MINLP (2.2), consisting of six components. The global optimal solution (large red point) is approximately $(1.55689, 1.30174)$; there are eight further local optima (small red points).

Example 2.2 cont. *The NLP relaxation of MINLP* (2.2) *has nine local optima, as can be seen in Figure 2.2, some of them with an objective function value which is worse than that of the integer optimum. The unique global optimum of the NLP is approximately* $(1.55689, 1.30174)$*, with an objective function value of about* 6.76385*. The optimal solution of* (2.2) *has an objective function value of* 13*, hence the dual gap of the NLP optimum is* $^{13-6.76385}/_{13} \approx 47.97\,\%$*.*

For two MINLPs P_1 and P_2 with NLP relaxations $\bar{P}_1$ and $\bar{P}_2$, respectively, we call P_1 a *sub-MINLP* of P_2, if $\tilde{\mathcal{X}}(\bar{P}_1) \subseteq \tilde{\mathcal{X}}(\bar{P}_2)$. Particularly, a sub-MINLP can be obtained by adding constraints or tightening the variable domains.

For a point $\bar{x} \in [l,u]$ (i.e., $l_j \leqslant \bar{x}_j \leqslant u_j$ for all $j \in \mathcal{N}$) the index set of all *fractional variables* is defined as $\mathcal{F} := \{j \in \mathcal{I} \mid \bar{x}_j \notin \mathbb{Z}\}$. This terminology comes from mixed integer *linear* programming, for which the values of an LP relaxation are indeed rational, i.e., representable as fractions, when the input data is rational. For the ease of notation, we will use the term “fractional” also in the nonlinear case.

In this thesis, we sometimes speak of *integer feasible* points or integer points, by which we mean that all integer variables shall take an integral value; the continuous variables might take arbitrary values (within their domain). Shifting the values of all fractional variables to one of the two nearest integers is called a *rounding*; formally:

Definition 2.9 (rounding)**.** *Let $\bar{x} \in [l, u]$. The set*

$$\mathcal{R}(\bar{x}) := \{x \in \mathbb{R}^n \mid x_j \in \{\lfloor \bar{x}_j \rfloor, \lceil \bar{x}_j \rceil\} \textit{ for all } j \in \mathcal{I}, l_j \leqslant x_j \leqslant u_j \textit{ for all } j \in \mathcal{N}\}$$

is called the set of roundings *of $\bar{x}$.*

Heuristic algorithms for special classes of optimization problems typically use very problem-specific information. In the case of heuristic algorithms for general MINLP, there is little global information available that can be used for decision-making. Among the most important concepts are the so-called *variable locks* which have been introduced for constraint integer programs by Achterberg [Ach07b].

Definition 2.10 (up- and down-locks)**.** *Let g_i $\mathbb{R}^n \to \mathbb{R}$ be a constraint function of an MINLP. The constraint $g_i(x) \leqslant 0$ up-locks (down-locks) variable x_j if there exists a vector $\bar{x}$ and a scalar $\varepsilon > 0$ ($\varepsilon < 0$) such that $g(\bar{x}) \leqslant 0$, but $g(\bar{x} + \varepsilon e_j) > 0$, with e_j being the j-th unit vector. The number of constraints which up-lock (down-lock) variable x_j is denoted by $\overline{\kappa}_j$ ($\underline{\kappa}_j$) and called the* up-locks (down-locks) *of x_j.*

Variable locks are an indicator for the impact of variable shifts on the feasibility. More specifically:

Remark 2.11. *Let $\bar{x}$ be a vector that is feasible for the NLP relaxation of a given MINLP and $\bar{x}_j \neq u_j$. If we shift variable x_j by an $\varepsilon \in (0, u_j - \bar{x}_j]$, the vector $\tilde{x} + \varepsilon e_j$ will violate at most $\overline{\kappa}_j$ constraints.*

For MIPs, the number of up- and down-locks is completely defined by the signs of the coefficients of matrix A. It holds that $\overline{\kappa}_j = |\{i \in \mathcal{M} \mid A_{ij} > 0\}|$ and $\underline{\kappa}_j = |\{i \in \mathcal{M} \mid A_{ij} < 0\}|$, see, e.g., [Ber06].

In the following paragraphs, we will give some pointers to the computational complexity of MINLP. For a comprehensive survey of complexity results for different variants of MINLP, see [HKLW10, Köp12]. When discussing the complexity of *nonlinear* programming, additional issues have to be taken into account as compared to linear programming. First, it is not given a priori that the constraint functions can be evaluated in polynomial time. Further, it is possible for the optimal solution to be irrational (consider the simple NLP $\min x$ s.t. $x^2 \geqslant 2$) or even transcendental. It is typically assumed that the nonlinear functions are presented by oracles and that the

MINLP is to be solved up to a given ε-tolerance (for optimality and constraint feasibility).

0-1 integer linear programming is among the 21 problems that Richard Karp showed to be $\mathcal{NP}$-complete in his famous paper "Reducibility among combinatorial problems" [Kar72]. This is a result that can be extended to IP [BT76] and MIP, see, e.g. [Fuk11].

Since deciding the feasibility of a BP is $\mathcal{NP}$-complete, BP is $\mathcal{NP}$-hard by definition. MINLP includes BP as a special case and is therefore $\mathcal{NP}$-hard, too. Further, each BP can be formulated as an NLP by replacing the integrality constraints $x_j \in \{0, 1\}$ by nonconvex constraints $x_j(1 - x_j) = 0$ for all $i \in \mathcal{I}$. Thus, NLP is $\mathcal{NP}$-hard. Surprisingly, it has been shown that pure-integer quadratic programming, and therefore MINLP, is undecidable in finite time [Mat70].[6] The proof relies on unbounded variables; since if all variables are bounded (and integer), the problem can obviously be decided in finite time by enumeration. It follows as a corollary that MINLP is not even in $\mathcal{NP}$.

Solving linear programs, however, is possible in polynomial time. This was first shown by Khachiyan [Kha79]. Further, convex NLP is solvable in polynomial time (up to a given ε-tolerance), see Grötschel et al. [GLS81]. Minimizing a nonconvex quadratic function over a polyhedron, however, is $\mathcal{NP}$-complete [Vav90].

In this thesis, we will primarily focus on the actual measured performance of algorithms rather than their theoretical worst-case complexity. For this, we typically consider indicators such as the running time or the number of branch-and-bound nodes that a solver requires with and without using a certain feature. Of those numbers, we take averages over a test set of MINLP problems. We distinguish two different methods for determining a mean value:

Definition 2.12 (arithmetic and shifted geometric mean)**.** *Let* $n \in \mathbb{Z}_{\geqslant 0}$*,* $\mathcal{V} = \{v_1, \ldots, v_n \mid v_i \in \mathbb{R}_{\geqslant 0}$ *for all* $i\}$ *and* $s \in \mathbb{R}_{\geqslant 0}$*. The* arithmetic mean *of* $\mathcal{V}$ *is defined as*

$$\psi(\mathcal{V}) := \frac{1}{n} \sum_{i=1}^{n} v_i$$

and the shifted geometric mean *with of* $\mathcal{V}$ *with shift* s *is defined as*

$$\phi(\mathcal{V}, s) := \sqrt[n]{\prod_{i=1}^{n} (v_i + s)} - s.$$

We typically use a shift of $s = 10$ for time (measured in seconds) and $s = 100$ for nodes in order to reduce the effect of very easy instances in the mean values. Further, using a *geometric* mean prevents hard instances at, or

[6]This solved, or rather "unsolved", Hilbert's tenth problem [Hil00]

close to, the time limit from having a huge impact on the measures. Thus, the shifted geometric mean has the advantage that it reduces the influence of outliers in both directions.

2.2. Algorithms and global solvers for MINLP

Bixby and Rothberg noted in 2007 that MIP has gone through an "inflection point" [BR07] from a technology which is powerful but hard to apply, needing a high degree of customization towards the general applicability of state-of-the-art solvers which handle general, large-scale models out-of-the-box. Analogous to Bixby and Rothberg, it is our opinion that the state of affairs for MINLP is comparable to that of MIP twenty years ago. It is "viewed as a temptingly powerful modeling paradigm" [BR07], there are numerous (potential) applications, the academic literature features hundreds of publications describing methods to solve MINLPs, and the development of general purpose solvers has begun, but they have not yet reached the full maturity of, e.g., commercial MIP solvers. To stay in the rhetoric of Bixby and Rothberg, computational MINLP still is before the "tipping point", maybe even directly in front. This section gives a brief overview of the solvers for MIP and MINLP currently available and the solution methods they apply.

LP-based branch-and-bound

Branch-and-bound [LD60] is the most widely used algorithm to solve mixed integer programs. State-of-the-art MIP solvers such as SCIP [Ach09] , FICO Xpress [FIC], Gurobi [Gur], and IBM ILOG Cplex [IBM] all use *LP-based* branch-and-bound as a basic algorithm that is enhanced by various tricky subroutines to make the solvers efficient in practice. Examples of these subroutines are presented throughout this thesis.

The idea of branch-and-bound is simple, yet effective: an optimization problem is recursively split into smaller subproblems, thereby creating a search tree and implicitly enumerating all potential assignments of the integer variables.

The task of *branching* is to successively divide the given problem instance into smaller subproblems until the individual subproblems are easy to solve. The best of all solutions found in the subproblems yields the global optimum. During the course of the algorithm a *branching tree* is created with each node representing one of the subproblems.

The intention of *bounding* is to avoid the complete enumeration of all potential integer assignments for the initial problem, which usually are exponentially many. If a subproblem's lower (dual) bound is greater than or equal to the global upper (primal) bound, that subproblem can be pruned. Lower bounds are calculated with the help of a relaxation, which is expected to be easy to solve. Upper bounds are found if the solution of the relaxation is also (integer) feasible for the corresponding subproblem.

Most commonly, an *LP-relaxation* is solved for bounding. For (linear) MIPs, the LP relaxation is simply constructed by dropping the integrality conditions, see Definition 2.4. For MINLP, an LP relaxation can be constructed from the bounds of the variables, gradient cuts[7] for convex constraints and linear over- and underestimators of the nonconvex terms [McC76, TS02]. Note that for nonconvex MINLP, it is possible that the LP relaxation is integral and cannot be strengthened further by gradient cuts, while some of the nonconvex constraints are still violated. In this case, *spatial branching* can be applied, i.e., branching on variables contained in violated nonconvex constraints, including continuous variables (see, e.g., [HT96]). Subsequently, the LP relaxation can be tightened in the created subproblems; thereby, the infeasible relaxation solution is cut off.

Various techniques have been developed to improve this basic algorithm. Besides involved strategies for making good branching and subproblem selections, this includes supplementary procedures that help in tightening the lower and upper bounds. At each subproblem, domain propagation can be performed to exclude values from the variables' domains. The relaxation may be strengthened by adding further valid constraints (typically linear inequalities), which cut off the optimal solution of the relaxation, but retain all feasible solutions of the MINLP. In the case where a subproblem is found to be infeasible, conflict analysis might be performed to learn additional valid constraints. Primal heuristics are used as supplementary methods to improve the upper bound. Good overviews on the state-of-the-art in computational mixed integer linear and nonlinear programming can be found in [Ach07b] and [Vig12], respectively.

Other algorithms for MIP

Interestingly, years before the explorative branch-and-bound algorithm was introduced, another, more involved procedure to solve MIPs had been presented: the *cutting plane method* [Gom58, Gom60]. The basic idea of the cutting plane method is to iteratively solve and strengthen the LP relaxation of a MIP. To do so, in each iteration one or more linear inequalities are added to the LP relaxation. These inequalities have to fulfill two requirements:

1. they are violated by the current optimum $\bar{x}$ of the LP relaxation and
2. they are valid for each feasible solution $\tilde{x} \in \tilde{\mathcal{X}}(P)$

Since they "cut off" the LP optimum from the relaxation, such inequalities are called *cutting planes* or *cuts*. Algorithms that compute cutting planes are sometimes called *separators*. In [Gom58, Gom60], Gomory presented a finite algorithm to solve integer programs by generating cuts from simplex tableau rows.

[7]If a convex nonlinear constraint $g(x) \leqslant 0$, $g_i \in \mathcal{C}^1([l,u],\mathbb{R})$, is violated at some $\bar{x}$, then $\bar{x}$ can be cut off by the *gradient cut* $\nabla g_i(\bar{x})^\mathsf{T}(x-\bar{x}) + g_i(\bar{x}) \leqslant 0$.

The cutting plane method has two major drawbacks. First, the method is highly sensitive towards numerical issues – at least when using generic tableau-based cuts such as Gomory. The coefficients in the cuts are often highly fractional (in the sense that they do not have an accurate finite precision floating point representation) and the cuts get more and more "shallow" (in the sense that the Euclidean distance between the LP solution and the supporting hyperplane of the cut gets very small). Such errors accumulate over time and can cause the method to either produce an invalid result or to stall. Nevertheless, cutting plane separators are typically employed as supplementary procedures within modern branch-and-bound based MIP solvers. However, strict limits and methods like safe rounding are used to protect the solver against numerical issues. The numerical challenges of applying Gomory cuts are discussed in Cook et al. [CDFG09]. The implementation of cutting plane separators in SCIP and their integration within a MIP framework is described by Wolter [Wol06].

The second drawback of a pure cutting plane method is that it does not compute an incumbent solution; the first solution that it finds is optimal. This implies that if the procedure does not terminate within a given time limit, no feasible solution will be present. This is undesirable if hard MIPs have to be solved within industrial applications.

A possible categorization of algorithms to solve optimization problems is to subdivide them into *primal* and *dual methods*. Loosely speaking, a primal method is an algorithm that produces a sequence of feasible, sub-optimal solutions until it meets a criterion proving that the current incumbent solution is optimal. By contrast, a dual method is an algorithm that produces a sequence of infeasible, "super-optimal" solutions until it finds a first feasible point – which will be an optimum. As examples, consider the primal and the dual simplex algorithm.

The added advantage of LP-based branch-and-bound is that it produces two sequences during the course of the algorithm, providing dual and primal bounds at the same time.

By the above classification, a pure cutting plane algorithm, like the one described by Gomory, is a dual method; it approaches the set of feasible solutions "from the outside", solving a sequence of relaxations. As soon as the relaxation finds a point which is feasible for the MIP, the proof of optimality comes "for free".

Since the mid-1990's, there has been a rising interest in primal methods to solve MIPs. The principal ideas of primal methods, however, date back to the 60's and 70's. *Test set algorithms* and *integral basis methods* are two important groups of primal methods for (mixed) integer programming. Both procedures require a known feasible solution as a starting point.

Test set algorithms are motivated by the Ford-Fulkerson [FF56] algorithm to compute maximum flows through a network. A *test set* for a given integer program P is a finite set of n-dimensional integral vectors $\mathcal{T} \subset \mathbb{Z}^n$, such that $c^\mathsf{T} t < 0$ for all $t \in \mathcal{T}$ and for every non-optimal feasible solution $\tilde{x}$ of P, there exists a $t \in \mathcal{T}$ such that $\tilde{x} + t$ is a feasible solution of P. If given a feasible start solution and a test set for an IP, the algorithmic idea is straightforward; iteratively find an element of the test set that maintains feasibility when added to the solution. If no such element exists, the current solution is optimal. When Graver introduced the idea of test sets for integer programs, he showed that finite test sets exist for every feasible IP [Gra75]. Weismantel gives a good overview on different methods to computationally obtain test sets for IPs [Wei98].

The “Simplified Primal Integer Programming Algorithm” suggested by Young [You68] can be seen as one of the first versions of an *integral basis method* or a *primal cutting plane method* for integer programming. The idea is to solve IPs only by means of the primal simplex algorithm, hence starting from a feasible solution (and an associated basis) and only performing simplex pivots that improve the objective, maintaining primal feasibility and integrality of the basic solution. Since this might not be possible in general, the constraint matrix is manipulated. In [You68], this is done by adding a Gomory cut (and its slack variable) for the row that is determined as pivot when conducting the ratio test of the simplex algorithm. This cut itself will then be chosen as pivot row instead and by construction the coefficient of the pivoting variable in the cut and the pivot ratio cancel out. As a consequence, both, the cut's slack and the pivot column take integral values in the new linear system that has been enhanced by one column and one row. A different understanding of this procedure is that it cuts off neighboring fractional points of the incumbent feasible solution until it can make a simplex step that leads to a new incumbent.

Several extensions of this algorithm have been suggested. Notably, Haus et al. present an integral basis method that manipulates the columns of the matrix *without any* cuts being added [HKW01]. In contrast, Letchford and Lodi [LL02] suggest several enhancements that make Young's algorithm converge quicker by adding *more* cuts. The main improvements come from separating classes of cuts other than only Gomory cuts and by potentially adding several cuts per round, which is typical in dual cutting plane algorithms.

There is a smooth transition between primal methods and primal heuristics. On the one hand, some primal heuristics such as Local Branching or Proximity Search can be modified such that they become complete algorithms, see [FL03, FM12]. In this case, each of their iterations will take an integer solution as an input and have either an improved integer solution or a proof of optimality as output, which is the general concept of primal methods. On the other hand, complete primal methods such as test set algorithms or the integral basis method could of course be run for a limited time as a primal heuristic within a branch-and-bound-based MIP solver.

Primal methods show impressive results for some particular classes of mixed integer programs, see, e.g., [ABH^{+}00]. In an MINLP context, primal methods have been used to prove complexity results, see, e.g., Hemmecke et al. [HKLW10]. However, to the best of the author's knowledge, there is no publication that describes a primal algorithm that is computationally competitive to LP-based branch-and-bound on the majority of instances of a general MIP benchmark such as the Miplib[8]. Further, we are not aware of any of the major commercial MIP solvers implementing any of the named methods, not even as a heuristic.

Other algorithms for MINLP

For *convex* MINLP, several solution approaches have been suggested and implemented as an alternative to LP-based branch-and-bound.

A straight-forward extension of LP-based branch-and-bound for MIP is *NLP-based branch-and-bound* for convex MINLP [GR85]. This involves dropping the integrality constraints and solving (naturally convex) NLP relaxations to obtain dual bounds during the tree search. Recall that convex NLP, just like LP, is solvable in polynomial time, which is not the case for nonconvex NLP. NLP solvers, however, cannot be as easily hot-started as the simplex algorithm, which is a disadvantage of NLP-based branch-and-bound as compared to using LP-based branch-and-bound for MINLP. Besides other approaches, Leyffer [Ley01] used sequential quadratic programming to solve and warm-start NLPs within a branch-and-bound algorithm. The clear advantage of NLP-based branch-and-bound in comparison to using a LP relaxation is that the dual bounds are stronger.

Many improvements over using pure nonlinear relaxations come from using different relaxations within one framework. The classical *outer approximation* algorithm by Duran and Grossmann [DG86] was proposed for problems where the nonlinearity only occurs in the continuous variables. It solves, alternatingly, MIP relaxations and NLP subproblems. The MIP solutions are used to define the NLP problems, the NLP solutions are then used to generate cuts for the MIP relaxation. Westerlund and Pettersson [WP95] suggest to generate cuts directly from the MIP solution without solving an additional NLP.

Quesada and Grossmann [QG92] combine LP and NLP relaxations in one tree search. Their algorithm uses LP relaxations for the majority of the search, exploiting the hot-start capabilities of the simplex algorithm. Only when the LP optimum is integer, an NLP relaxation is solved. Mahajan et al. provide the suggestion of using QP approximations instead of LP relaxations [MLK12]. By this, second-order information is captured, but the QP is not necessarily a relaxation. Thus, the full NLP relaxation might also need to be solved at nodes that will be pruned.

Since for nonconvex MINLPs, solving the continuous relaxation is in principle as hard as solving the original problem[9], there are few general methods

[8]Note, however that the method of Haus et al. [HKW01] succeeded in solving seven of the 65 Miplib 3.0 instances

[9]Recall that integrality constraints of bounded variables can be expressed as polynomials over continuous variables.

that are based upon solving NLPs. A notable exception is the approach of Androulakis et al. [AMF95] who use a convex NLP relaxation to solve nonconvex MINLPs.

Software for MIP and MINLP

State-of-the-art MIP solvers are nowadays capable of solving a variety of different types of MIP instances that arise from real-world applications within reasonable time [KAA$^+$11], establishing a growing market for mathematical optimization software.

Among the first commercial mathematical programming softwares were IBM's MPS/360 [Per68] and its predecessor MPSX, which were introduced in the 1960's; see also [Spi04]. Interestingly, their input data format `.mps` is still the standard format for all state-of-the-art MIP solvers about half a century later.

Today, there is a large variety of commercial MIP solving software, including Xpress [FIC], Gurobi [Gur], Cplex [IBM], and Mosek [Mos]; all of them being capable of solving many MIPs of practical relevance to proven optimality. There are also several academic, noncommercial alternatives, such as cbc [Cbc], glpk [Glp], lpsolve [Lps], Symphony [Sym], or SCIP [Sci]; the best of them being only a factor of five away from the best commercial packages, in terms of average running time. Recently, Linderoth and Lodi compiled an overview of current MIP software, see [LL10]. Every two years, Robert Fourer publishes a list of currently available codes in the field of linear and integer programming, the 2013 edition being the latest at the time of writing this thesis [Fou13].

Only a few software packages solve general nonconvex MINLPs to global optimality, including the free solver Couenne [BLL$^+$09] and the solvers Antigone [MF14, MF13], baron [Sah96, TS04], and LindoGlobal [LS09, Lin]. Others, such as Alpha-ECP [WL01], Bonmin [BBC$^+$08, Bon] and sbb [Sbb], guarantee global optimality only for convex problems, but can be used as heuristic solvers for nonconvex problems. minotaur [Mio] is a framework for implementing MINLP algorithms. For a comprehensive survey of available MINLP solver software, see [BKL12, BV10, DL11].

Recently, the solver SCIP [Ach09, Sci] has been extended to solve nonconvex MIQCPs [BHV11] and MINLPs [Vig12] to global optimality. SCIP is currently one of the fastest noncommercial solvers for MIP [KAA$^+$11, Mit], MIQCP [Mit] and MINLP [Vig12].

Among the commercial MIP solvers, it seems hard to declare a single solver as the state-of-the-art. Koch et al. [KAA$^+$11] showed that for the Miplib 2010 benchmark set, all of the three solvers Cplex, Gurobi, and Xpress have a nearly identical geometric mean of the solution times on a 12 thread environment, with each solver being the single fastest on several individual instances. In 2005, a computational study by Neumeier et al. [NSHV05] compared nine different MINLP solvers, appointing baron the fastest solver at that time. In his thesis, Vigerske [Vig12] evaluated the performance of baron, Couenne, Lindo API, and SCIP on the MinlpLib [BDM03]. He found that SCIP outperforms the other solvers w.r.t. the number of solved instances, the average running time and the quality of both, primal and dual bounds.

2.3. Heuristic methods in MINLP

Primal heuristics are algorithms that try to find feasible solutions of good objective function quality for a given optimization problem, following some intuition of how such a solution can be constructed. More generally, a *heuristic algorithm* is a procedure that takes a decision to a given problem within short time, given limited information. These “definitions” involve a lot of hand-waving – What is good? What is short? What is limited? – but they capture the main characteristic of heuristic algorithms: they are expected to show a good trade-off between computational effort and success rate.

Furthermore, if an optimization problem could be solved by complete enumeration of a discrete set, a primal heuristic will typically perform only a partial search that is incomplete by design. As a consequence, one might interpret heuristics as the opposite of complete algorithms, which are guaranteed to converge to a globally optimal solution. Such complete (or global) algorithms, however, often feature heuristic algorithms to make crucial decisions.

Heuristics everywhere...

Branch-and-cut and the other algorithms named in Section 2.2 are designed to find a globally optimal solution of a given MIP or MINLP (up to numerical tolerances, see also the last paragraph of this section). Within state-of-the-art implementations of MIP and MINLP solvers, heuristic algorithms are ever-present, which we will demonstrate in the following.

A vanilla branch-and-bound algorithm, as it is described by Land and Doig [LD60] or Dakin [Dak65], has two main steps at which a decision is made: the branching selection and the node selection. Both selections have a major impact on the overall running time of the algorithm, and both are normally made by a heuristic criterion that proved to be good on average. For node selection, this is often a variant of depth-first-search. For branching selection heuristics, an overview is presented in Chapter 10. Note that for SAT, Liberatore proved that finding a branching that leads to a minimal search tree is at least as hard as solving the SAT problem itself [Lib00].

Cutting plane separators also involve many heuristic components. State-of-the-art MIP solvers like SCIP generate mixed integer rounding cuts from (heuristically chosen) aggregations of linear constraints, see, e.g., [MW01]. Cuts are filtered w.r.t. their numerical stability, including checks against several threshold values that are empirically tested to work well in practice.

Finally, only a subset of the filtered cuts will be actually added to the LP. This set of cuts is selected by a combination of heuristic criteria such as the orthogonality amongst each other, the parallelism w.r.t. the objective and the efficacy of the cuts, see, e.g., [Ach07b].

Presolving and domain propagation typically apply a fixed set of deduction rules until a fix point of the domain space is reached. However, even for this part of MI(NL)P solving, which is based on pure logic rather than on intuition, heuristic algorithms can be found. *Probing* [Sav94] and *Optimality-based bound tightening (OBBT)* [QG93] are time-consuming, explorative presolving techniques which solvers like SCIP, BARON, or CPLEX only apply to a limited set of variables. These processes abort prematurely if they do not prove successful, see, e.g., [GW13]. The order the variables are examined for probing or for OBBT is determined by a heuristic criterion such as the number of constraints in which a variable appears. Presolving for knapsack constraints uses a so-called clique partition [SS94], which is computed by a greedy algorithm in SCIP.

In LP solving, pivoting rules are heuristic algorithms. The commonly used ones work well in practice, but can be arbitrarily bad on some involved polytopes that are constructed to mislead them [KM72]. The analysis of infeasible LPs to generate conflict constraints is based on a heuristic filtering of the dual ray, see Chapter 9.2. Further, the factorization of basis matrices is typically performed in a way that few fill-in is created. To do so, heuristic algorithms are used, see, e.g. the study of Luce et al. [LTL$^+$09].

In conclusion, heuristic algorithms can be found in almost every part of state-of-the-art MIP and MINLP solvers. The most visible part, however, is the search for hidden feasible solutions prior and during the tree search: primal heuristics.

Primal heuristics

For mixed integer linear programming (MIP) it is well known that general-purpose primal heuristics, like the Feasibility Pump [AB07, FGL05, FS09], are able to find high-quality solutions for a wide range of problems. Over time, primal heuristics have become a substantial ingredient of state-of-the-art MIP solvers [Ber06, BFG$^+$00, LPT$^+$09]. Discovering good feasible solutions at an early stage of the MIP solving process has several advantages:

- ▷ The bounding step of the branch-and-bound [LD60] algorithm depends on the quality of the incumbent solution; a better primal bound leads to more nodes being pruned and hence to smaller search trees.
- ▷ The same holds for certain presolving and domain propagation strategies, such as reduced cost fixing [NW88]. Better solutions can lead to tighter domain reductions, in particular more variable fixings. Consequently, this might lead to better dual bounds and the generation of stronger cutting planes.

▷ In practice, it is often sufficient to compute a heuristic solution whose objective value is within a certain quality threshold. For hard MIPs that cannot be solved to optimality within a reasonable amount of time, it might still be possible to generate primal solutions of high quality quickly.

▷ Improvement heuristics, e.g., rins [DRP04] or Local Branching [FL03], need a feasible solution as a reference point.

Similar statements hold for other classes of mathematical programs. Often, techniques such as reduced cost fixing or cutting planes are more heavily, or even exclusively, applied at the root node of a branch-and-bound search tree. Therefore, already knowing good solutions during root node processing is more beneficial than finding them later during tree search.

Consequently, the last fifteen years have seen several publications on general purpose primal heuristics for MIP, including [AB07, BCD$^+$01, BSW04, BFL07, Ber14, BFS10, FS09, Gho07, GLW00, HMU06, Løk02, Rot07, Wal10]. For literature overviews, see [Ber06, FL10, Lod13, Mar11]. Despite the rising interest of the research community in recent years, primal heuristics for MIP date back to the early days of computational integer programming, for example, [EC68].

The performance improvement of a MIP or MINLP solver achieved by applying primal heuristics is a natural reason to study their application inside complete solvers. However, there is another, more surprising motivation. Recent experiments by Hans Mittelmann indicate that even for the default application of standalone primal heuristics – finding a single feasible solution quickly – the portfolios of embedded heuristics in state-of-the-art MIP solvers are competitive or superior to applying a single standalone heuristic. On the feasibility test set of the Miplib 2010, the solvers Cplex, Gurobi, Xpress and the state-of-the-art heuristic code Feasibility Pump 2.0 [FS09] (which uses Cplex as a subroutine), find solutions for 28 or 29 instances each, with the Feasibility Pump being a factor of 1.5 to 2.4 slower than the MIP solvers [Mit13]. One reason for this is that MIP solvers have access to a whole portfolio of internal primal heuristics; each of Xpress, Cplex, Gurobi, SCIP, and cbc feature a double-digit quantity of primal heuristics. We conclude that for forwarding the state-of-the-art in finding feasible solutions as fast as possible, studying and improving primal heuristics inside global solvers is a promising way to go.

The present thesis features literature overviews for rounding heuristics, see Section 4.2, feasibility pumps, see Section 5.2–5.3, and large neighborhood search heuristics, see Section 6.2–6.3. The following part gives an overview on classes of primal heuristics for MIP and MINLP that are *not* covered in one of the named sections.

MIP and MINLP primal heuristics: literature overview

Diving heuristics are a kind of "folklore": Most solvers and many custom codes use them, but there are few publications on the topic. Generally speaking, diving heuristics iteratively round one or more fractional variables and reoptimize a relaxation, thereby simulating a depth-first-search in the tree. One of the first references mentioning "diving" in a pure heuristic context are Bixby et al. [BFG+00] who state that at this time (2000), "all [of Cplex'] heuristics involve diving."[10] Danna et al. [DRP04] introduce guided diving, i.e., rounding a fractional variable that is closest to its value in the incumbent to that value. Berthold [Ber06] gives an overview on six different diving heuristics that are implemented in SCIP.

The *Active Constraint Branching* of Patel and Chinneck [PC07], the probabilistic *Force Change Branching* of Pryor and Chinneck [PC11], and the *Foundation Constraint Method* by Mahmoud and Chinneck [MC13] are all aimed at applying branching strategies that quickly find feasible solutions and neglect the proof of optimality. By this, they actually work like a typical diving heuristic and should be considered in this context.

Bonami and Gonçalves [BG12] generalize the *Fractional Diving* heuristic and the *Vectorlength Diving* heuristic from [Ber06] to convex MINLP by solving NLP relaxations instead of LP relaxations. Mahajan et al. [MLK12] suggest a diving algorithm that uses quadratic programming relaxations. While their algorithm is designed as a global method, it can be easily adopted to be used as a primal heuristic for a single dive. The author of this thesis, in cooperation with Stefan Vigerske from GAMS, implemented nonlinear equivalents of all six (linear) diving heuristics in SCIP. Additionally, we experimented with diving rules that prefer variables from a minimum cover, see Section 8.3, and that are fractional in an LP and NLP relaxation at the same time. Both ideas improved the heuristic's behavior.

Pivoting heuristics use knowledge of the index set of integer variables of a MIP. They manipulate the primal simplex algorithm in such a way that it tries to force integrality on those variables. Recall that variables in the non-basis are always at one of their bounds. This implies that continuous variables and slack variables of constraints should be preferred over binary variables to be put into the basis. A pivoting heuristic for pure BPs called *Pivot-and-Complement* is presented by Balas and Martin [BM80]. It performs three kinds of simplex iterations, starting from an LP-optimal basis:

1. pivots that maintain primal feasibility and exchange a slack variable in the non-basis for a fractional binary variable in the basis,

2. pivots that maintain primal feasibility and complement a binary variable (i.e., switch its value from 0 to 1 or vice versa), while reducing the sum of fractionalities, and

[10]Note that performing an initial dive (that is *not* discarded) at the beginning of a branch-and-bound search has been proposed before, see, e.g., Beale [Bea79].

3. pivots that sacrifice primal feasibility in order to bring a slack into a basis in exchange for a fractional binary variable.

Furthermore, they included a *complementing* step in which sets of two or three variables are complemented at once, while reducing the sum of fractionalities. This is performed in conjunction with a reduced cost fixing step.

Balas, Schmieta, and Wallace [BSW04] introduce a heuristic called *Pivot-and-Shift*, based on a technical report by Balas and Martin [BM86]. In addition to Pivot-and-Complement, it considers continuous variables and general integer variables with (small) finite domains, partitioning the latter into binary variables. As a consequence, the complementing step is replaced by a *shifting* step that involves shifting the (integral) solution value of an integer variable to one of the neighboring integral values. Further, the improvement phase is amended by a Local Branching procedure, see Section 6.2.

Eckstein and Nediak present *Pivot-Cut-and-Dive* [EN07], a heuristic for mixed binary programs. Its main procedure is a rounding method that is based on simplex pivot operations. This involves combining a concave merit function, e.g., the Euclidean distance, with the original objective to rate the potential pivots. This basic algorithm is enhanced by an explicit probing on pivots when the method reaches a local optimum (w.r.t. the merit function). If this probing phase fails, an intersection cut can be generated. As a last resort, the heuristic performs diving (see above) by fixing groups of variables, thus this procedure is named Pivot-Cut-and-Dive.

A heuristic called *Pivot and Gomory Cut* by Ghosh and Hayward [GH05] uses Gomory cuts [Gom58] to guide the pivot selection and to avoid cycling: like Pivot-And-Complement, it seeks to perform pivots that reduce the overall fractionality, under the additional restriction to make a previously generated Gomory cut less violated. Løkketangen et al. [LJS94] describe two ways of integrating tabu search into Pivot-and-Complement: allowing non-improving moves in order to escape local minima and defining neighborhoods in a post-processing improvement phase.

Saltzman and Hillier [SH92] present a so-called *ceiling point* algorithm that searches for the closest integer points to one or more of the hyperplanes defined by the linear constraints. Here, “closest” refers to a local viewpoint: a point is called “closest” to a hyperplane when within its 1-neighborhood, there is no point with a smaller Euclidean distance to that hyperplane. An enumeration of such points is combined with a simple rounding procedure and a 2-opt postprocessing, compare Chapter 4.

Bastert et al. [BHdV10] propose a generalization of a problem specific heuristic by Wedelin [Wed95] that was developed to solve airline crew scheduling problems. The basic idea is to iteratively solve Lagrangian relaxations of a mixed integer program. Therefore, the Lagrangian multipliers are computed from reduced costs and a preference matrix that collects history information over the iterations.

Glover and Laguna [GL97a, GL97b] propose a scatter search algorithm that takes convex combinations of integral points that are derived by roundings of LP solutions. Their algorithm generates intersection cuts to diversify the search.

The literature on primal heuristics for general MINLPs can be roughly subdivided into large neighborhood search strategies, Feasibility Pumps, variable neighborhood search approaches, and general meta-heuristics. For a literature overview on large neighborhood search for MINLP, see Section 5.3. For a literature overview on nonlinear Feasibility Pumps, see Section 6.3.

Large neighborhood search heuristics explore a single subproblem, which is defined as the neighborhood of a reference point. This subproblem typically is of considerable size compared to the original problem. Thresholds like fixing (only) 50 % of the variables are common. *Variable neighborhood search* heuristics iteratively explore different neighborhoods, updating the reference point, thresholds, and the neighborhood size (often growing over time) after each iteration. Sometimes even changing the way to define neighborhoods is changed during the course of the algorithm.

RECIPE, an acronym for *Relaxed-Exact Continuous-Integer Problem Exploration*, was introduced by Liberti et al. [LNM10, LMN11]. The neighborhoods in RECIPE are defined by Local Branching constraints, see Section 6.2, and by restricting the bounds of general integer variables. In this neighborhood, random points are sampled as starting points to find a local optimum of an NLP relaxation. Such a local optimum is then used as a reference point for a convex MINLP solver which is used to find an integer point – as a solution candidate and as a reference point for the iterative neighborhood definition.

The iterative rounding heuristic by Nannicini and Belotti [NB12] alternately solves NLPs and MIPs that are obtained by the relaxation of the original MINLP, Local Branching constraints, and restrictions of variable domains. Typically, a couple of integer points close to the NLP solution are generated by solving a Feasibility Pump-like MIP, and excluding the previously visited points by no-good cuts.

The *Restrict-and-Relax* algorithm for MBPs by Guzelsoy et al. [GNS13] can be modified to run as a variable neighborhood search heuristic. Restrict-and-Relax fixes variables with integral LP solution value and large reduced costs. However, during a branch-and-bound-like search, variables may be unfixed again, depending on the feasibility status and the objective value of the LP. Additionally, other variables are fixed, thereby defining a new neighborhood to be searched.

Meta-heuristic approaches are often used to compute promising reference points for multistart heuristics for nonconvex NLP and MINLP. As an example consider the *OptQuest/NLP* algorithm by Ugray et al. [ULP$^+$07]. LOCALSOLVER [BEG$^+$11, Gar13] is a commercial heuristic software that is based on local search. It combines several neighborhood search implementations with constraint propagation, MIP and NLP techniques.

Please note again, that literature overviews on the broad topics of rounding heuristics, Feasibility Pumps, and LNS heuristics are given in the corresponding Chapters 4, 5, and 6 of this thesis.

A note on numerics

Exactly speaking, state-of-the-art MIP solvers have an intrinsic heuristic flavor, since they use floating point arithmetic and numerical tolerances. This means that they only solve an approximation (though typically an extremely tight one) of the original problem and might accept solutions as feasible that are only “very close to feasible”. Additionally, they might reject solutions which are “nearly infeasible”. A recent study by Cook et al. [CKSW13] shows that for standard MIP benchmark instances, an exact and an inexact version (in a numerical/computer-algebraic sense) of SCIP report identical optimal objective values for the majority of the cases, and that discrepancies are of negligible magnitude, if they occur. However, for particularly ill-conditioned problems the situation may change drastically.

Numerical flaws are a topic of interest for computational MIP, and even more MINLP, but will not be covered in this thesis. We refer the interested reader to [Gol91] for an overview and a discussion of the pitfalls of floating point arithmetic. Finally, the papers of Cook et al. [CDFG09, CKSW13], and Neumaier and Shcherbina [NS04] discuss approaches for solving MIPs without floating point errors.

3. Measuring the impact of primal heuristics

In modern MIP and MINLP solvers, primal heuristics play a major role in finding and improving feasible solutions early in the solution process. However, classical performance measures such as time to optimality or number of branch-and-bound nodes reflect the impact of primal heuristics on the overall solving process very poorly. Two reasons for this are that standard performance measures typically depend on the convergence of the dual bound and that they only consider instances which can actually be solved within a given time limit – whereas employing heuristics is particularly worthwhile for hard instances which cannot be solved to proven optimality within reasonable time, as we will show in Chapter 11.

In this chapter, we introduce a new performance measure, the "primal integral", which depends on the quality of solutions found during the solving process as well as on the points in time when they are found. We argue why this measure better captures the benefit of using primal heuristics inside a complete solver and confirm this claim by computational experiments. Our results reveal that heuristics improve the performance of state-of-the-art MIP solvers in terms of the primal integral by around 80%. The main part of this chapter has been published in Operations Research Letters [Ber13].

This chapter is organized as follows. After a short introduction in Section 3.1, we review existing measures and discuss the challenge to design a robust performance indicator in Section 3.2. In Section 3.3, we formally define the primal integral. Our computational study is presented in Section 3.4. Finally, we discuss several ways to extend and generalize our definition in Section 3.5, before we conclude in Section 3.6.

3.1. Introduction

When implementing optimization software, two questions naturally arise: how does the new code perform with respect to existing codes and which are the best settings for a particular algorithm? This goes back to the early days of operations research: Hoffman et al. reported a first computational experiment to compare different implementations of linear programming algorithms in 1953 [HMSW53]. Just as researchers and software vendors want to distinguish their code on general test sets, a user wants to tune an optimization software for a particular set of problems. However, all parties require suitable criteria for measuring the performance of a software implementation.

With the rise of computational research, standards and guidelines for conducting computational experiments were proposed [CDM78, JBNP91, Hoo94, McG96, GGM+97] or recently [Coo08, KMP13]. One key issue of the cited articles is the choice of suitable performance indicators. In mathematical programming, the running time to optimality is the "gold standard" for performance comparisons. For branch-and-bound based algorithms, the number of branch-and-bound nodes is another typical measure. Similarly, when using a simplex or an interior point based solver, the number of iterations is commonly used. Both, the number of iterations and the number of nodes, attempt to estimate the running time by a measure that is less dependent on the hardware and at the same time better reflects the sheer computational complexity.

In this chapter, we will use mixed integer linear programming as a showcase for our computational experiments. One advantage of branch-and-bound based algorithms, as opposed to, e.g., pure cutting plane algorithms, is that suboptimal incumbent solutions often are available early during the course of the algorithm. Primal heuristics boost this characteristic even further. As a consequence, solutions with a small optimality gap might be available long before the branch-and-bound search terminates.

Knowing good solutions early during optimization helps to prune the search tree and to simplify the problem via dual reductions. Further, they help to quickly check the correctness of the underlying model. Moreover, a practitioner might be satisfied with a solution that is proven to be within a certain gap to optimality. As a matter of fact, a typical use case in industrial applications of optimization is to find the best possible solution within a strictly predetermined time limit.

3.2. Trading off speed against solution quality

The time needed to find a first feasible solution, an optimal solution, or a solution within a certain gap to optimality (see, e.g., [HP93]) are performance criteria that concentrate on the primal part of the solution process. Each of these has its individual strengths and weaknesses. The time to first solution entirely disregards the solution quality: for about one quarter ($^{23}/_{87}$) of the Miplib 2010 [KAA+11] benchmark instances, a trivial solution of all variables set to their lower bound (or all to their upper bound) is feasible[11], but most of the times such a solution does not provide valuable information to the user. Particularly when analyzing heuristics embedded in a complete solver, the time to the first solution mainly measures the time needed for preprocessing and solving the root node relaxation: the MIP solvers Cplex, Gurobi and Xpress find solutions for 72, 70, and 64 of the 84 feasible Miplib 2010 benchmark instances during root node processing. The time to

[11]The "trivial" heuristic in SCIP checks the feasibility of these two solutions before preprocessing starts.

optimal solution, however, ignores that slightly suboptimal but practically sufficient solutions might have been found long before. Finally, taking the time to a certain gap is an attempt to balance this, but the choice of the threshold is arbitrary by design.

Altogether, the important consideration for primal heuristics is the trade-off between speed and solution quality. None of the above measures entirely meets this requirement. In marked contrast, two of the named measures, time to first solution and time to optimality, rather represent extreme points. It is our goal to introduce a new performance measure that reflects the development of the solution quality over the complete optimization process.

We argue that standard performance criteria are not well suited to describe the impact that primal heuristics have within a solver. Take the following observation. On the one hand, for state-of-the-art MIP solvers, the impact of primal heuristics on the overall running time and the number of branch-and-bound nodes is typically minor to negligible,[12] whereas other components such as cutting planes or branching rules change these numbers by a factor of two or three.

On the other hand, the solver vendors, such as Cplex, Gurobi or Xpress, seem to consider primal heuristics to be a “trade secret”. It stays unrevealed, which heuristics they use, when those are called, or just how many of them a solver features – whereas for other components, there are plenty of user parameters and statistical output available.[13] One interpretation of this discrepancy might be that primal heuristics are considered a – if not the – crucial part of the software, and their value is simply not reflected by the performance measures that we commonly use.

3.3. The primal integral

In this section, we introduce a new performance measure, in particular for benchmarking primal heuristics, that takes into account the whole solution process. The goal is to measure the progress of the primal bound’s convergence towards the optimal solution over the entire solving time. Therefore, we make use of the *primal gap* of a feasible solution, consider this as a function over time, and compute the integral of that function. Recall Definition 2.5, the primal gap of a feasible solution:

Definition (primal gap)**.** *Let $\tilde{x}$ be a solution for an MINLP, and $x^\star$ be an optimal (or best known) solution for that MINLP. We define the* primal gap

[12]Presentations by software vendors mention values in the range of five to ten percent [Bas07, BFG$^+$00].

[13]Consider the recent study [AW13] of two Cplex main developers: They present exhaustive computational results for ten different classes of cutting planes and nine presolving strategies. The only published heuristic for which they give numbers is rins [DRP04].

$\gamma^p \in [0,1]$ *of* $\tilde{x}$ *as:*

$$\gamma^p(\tilde{x}) := \begin{cases} 0, & \text{if } c^\mathsf{T} x^\star = c^\mathsf{T}\tilde{x} = 0, \\ 1, & \text{if } c^\mathsf{T} x^\star \cdot c^\mathsf{T}\tilde{x} < 0, \\ \frac{|c^\mathsf{T} x^\star - c^\mathsf{T}\tilde{x}|}{\max\{|c^\mathsf{T} x^\star|,\ |c^\mathsf{T}\tilde{x}|\}}, & \text{otherwise.} \end{cases}$$

Note that for two feasible solutions $\tilde{x}_1$, $\tilde{x}_2$ with $\operatorname{sgn}(c^\mathsf{T}\tilde{x}_1) = \operatorname{sgn}(c^\mathsf{T}\tilde{x}_2) = \operatorname{sgn}(c^\mathsf{T}x^\star)$ and $c^\mathsf{T}\tilde{x}_1 < c^\mathsf{T}\tilde{x}_2$ it holds that $\gamma^p(\tilde{x}_1) < \gamma^p(\tilde{x}_2)$. Now, assume that we have available the objective function values of intermediate incumbent solutions and the points in time when they have been found – for a given MIP solver, a certain problem instance, and a fixed computational environment. This information can be gathered from the log files that standard MIP solvers produce.

Definition 3.1 (primal gap function)**.** *Let* $t_{max} \in \mathbb{R}_{\geqslant 0}$ *be a limit on the solution time of a MIP solver. Its* primal gap function $p\colon [0, t_{max}] \mapsto [0,1]$ *is defined as follows:*

$$p(t) := \begin{cases} 1, & \text{if no incumbent until time } t, \\ \gamma^p(\tilde{x}(t)), & \text{with } \tilde{x}(t) \text{ being the incumbent at time } t, \text{ otherwise.} \end{cases}$$

The primal gap function $p(t)$ is a step function that changes whenever a new incumbent is found. It is monotonically decreasing, one at $t = 0$, and zero from the point on at which the optimal solution is found.

Definition 3.2 (primal integral)**.** *Let* $T \in [0, t_{max}]$ *and let* $t_i \in [0, T]$ *for* $i \in \{1, \ldots, I-1\}$ *be the points in time when a new incumbent solution is found,* $t_0 = 0$, $t_I = T$. *We define the* primal integral $P(T)$ *of a run as:*

$$P(T) := \int\limits_{t=0}^{T} p(t)\,dt = \sum_{i=1}^{I} p(t_{i-1}) \cdot (t_i - t_{i-1}).$$

We suggest using $P(\mathrm{t}_{\max})$ for measuring the quality of primal heuristics. It features two simple, but important, attributes: First, whenever a better solution is found at the same point in time, $P(\mathrm{t}_{\max})$ decreases. Second, whenever the same solution is found at an earlier point in time, $P(\mathrm{t}_{\max})$ decreases. Briefly: the primal integral favors finding good solutions early. For the performance measures discussed in the introduction, at most one of these two attributes holds in general.

The fraction $P(\mathrm{t}_{\max})/\mathrm{t}_{\max}$ can be seen as the average solution quality during the search process. In other words, the smaller $P(\mathrm{t}_{\max})$ is, the better is the expected quality of the incumbent solution if we stop the solver at an arbitrary point in time. Taking the development of the primal bound as a measure even reflects some psychological consideration: a quick drop in the primal bound and the early availability of a high-quality solution will most likely be perceived as an indicator of “good performance” by users of optimization software. The primal integral is an absolute measure in the sense that it is only defined by a single solver, unlike, for instance, a performance profile[14] which compares relative performance and is defined by a set of solvers.

Note that, for pure feasibility problems (instances with a zero objective function) the primal integral and the overall running time will give the same measure, up to a constant scaling factor. This follows from the simple observation that for feasibility instances the primal gap is one, before the solution is found (and the solution process thereby terminates) and zero afterwards. Hence when for two different runs the running time differs by a factor of k, the primal integral $p(t)$ will also differ by a factor of k.

In a recent work on rounding heuristics for MINLP, Nannicini and Belotti [NB10] used the percentage of total running time for which one given algorithm gave rise to a strictly better solution than another one to compare two solution processes. This can be formulated in terms of our notation as

$$\frac{1}{T}\int\limits_{t=0}^{T} \chi_{\{p_1>p_2\}}(t)\,dt$$

with p_1 and p_2 the primal gap functions of two runs of a solver (with different settings) and $\chi_{\{p_1>p_2\}}(t)$ a characteristic function, being one if $p_1(t) > p_2(t)$ and zero otherwise. Main differences between this measure and the primal integral are that this measure neither takes the actual objective function values of the solutions into account, nor is it an absolute measure, since it compares the relative performance of two algorithms.

In Section 3.4, we will present computational experiments comparing the performance of state-of-the-art MIP solvers with and without primal heuristics. Further, we will use the primal integral for our final evaluation in Chapter 11.

3.4. Computational experiments

We conducted our computational experiments with five state-of-the-art MIP solvers: cbc 2.7.0 [Cbc], FICO Xpress 23.01.06 [FIC], Gurobi 5.1.0 [Gur], IBM ILOG Cplex 12.5.0 [IBM], and SCIP 3.0.1 [Ach09] compiled with SoPlex 1.7.1 [Sop] as underlying LP solver. Cplex, Gurobi, and Xpress are among the fastest commercial MIP solvers, and cbc and SCIP are among the fastest open-source MIP solvers [Mit]. Further, these are the five codes that have been used to compile the Miplib 2010 [KAA$^+$11] benchmark set

[14] Performance profiles [DM02] represent the relative performance of a set of algorithms compared to a virtually best solver as a curve in a graph. A performance profile shows what percentage of the instances (which is the ordinate) of a given test set a given solver could solve within a time factor (which is the abscissa) of the best solver.

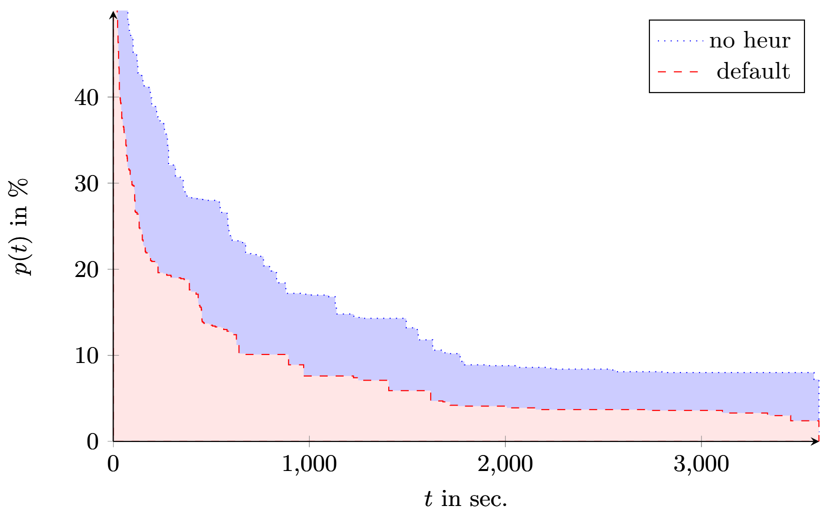


Figure 3.1.: Course of the primal gap when running SCIP with and without primal heuristics

which we chose as a test set for our computational experiments. We excluded the three infeasible instances `ash608gpia-3col`, `enlight14`, and `ns1766074`. Additionally, we excluded the instance `mspp16`, because SCIP and CBC ran out of memory. This leaves 83 instances in the test set.

The results for running the solvers in default mode are taken from the benchmarks for optimization software webpage of Hans Mittelmann [Mit], as of 20. February 2013. Additional results with disabled primal heuristics were obtained on the actual same computer, a 64bit Intel Xeon X5680 CPU at 3.20GHz with 12 MB cache and 32 GB main memory, running an OPEN-SUSE 12.1 with a GCC 4.6.2 compiler.[15] Turboboost was disabled. In all experiments, there was only one job at a time to avoid random noise in the measured running time that might be caused by cache misses if multiple processes share common resources.

As a first test, we compare the performance of SCIP and CPLEX when running with and without primal heuristics. We show the evolution of the primal gap in Figures 3.1 and 3.2. The red dashed line corresponds to the average (taken over 83 instances) primal gap function, when running the solver in default mode. The red shaded area corresponds to the average primal integral. Accordingly, the blue dotted line and the blue shaded (plus the red shaded) area correspond to the average primal gap function and the average primal integral when running the solver without heuristics.

For SCIP, the average value of $P(\mathrm{t_{max}})/\mathrm{t_{max}}$ was 9.05 % when using primal heuristics and 16.18 % without. For CPLEX, it was 1.92 % and 3.58 %, respectively. This indicates that, on this test set, for these two solvers, primal heuristics lead to an improvement of 78.8 % and 86.5 % in the primal bound,

[15]We acknowledge Hans Mittelmann for his help with this experiment.

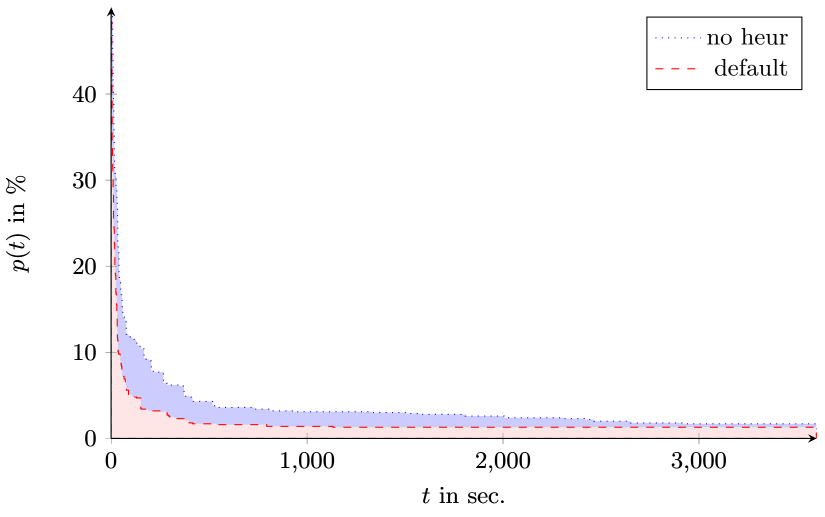


Figure 3.2.: Course of the primal gap when running Cplex with and without primal heuristics

on average. As a comparison, the running time to optimality was 819.0 seconds with and 910.5 seconds without primal heuristics for SCIP (11 %); 181.1 seconds and 239.9 seconds for Cplex (32 %). Both solvers solved four instances less when not using primal heuristics. Note that for both solvers the "default" function is strictly smaller than the "no heur" function. This implies that, independent of the chosen time limit, using primal heuristics is consistently superior in terms of the primal integral.

We conducted the same experiment with SCIP on the challenge test set of the Miplib 2010. We removed all instances which are known to be infeasible, for which the feasibility status is unknown (as of April 2013) and four large instances for which SCIP runs out of memory before even solving the root node LP. The reference solution values were compiled from the Miplib 2010 webpage and from additional 24-hour runs of Cplex 12.5 and Gurobi 5.5. We show the evolution of the primal gap in Figure 3.3. The underlying results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20GHz with 12 MB cache and 48 GB main memory, running an openSUSE 12.1 with a gcc 4.6.2 compiler, running each job exclusively. For this test set of notoriously hard MIP instances, the measured impact of using primal heuristics was even larger. The average value of $P(\mathrm{t}_{\max})/\mathrm{t}_{\max}$ was 39.02 % when using primal heuristics and 78.22 % without. Hence using primal heuristics reduced the primal bound by a factor of two on average. Note that from 151 instances in our test set, there was only one, `bnatt400`, which was solved to optimality by either version within the time limit of one hour. Thus, where comparing running time to optimality is not an option, the primal integral allows for a meaningful comparison which complies with the previous results.

Of course, we can use the primal integral as another metric to compare the performance of solvers against each other. It has been argued in [KAA⁺11]

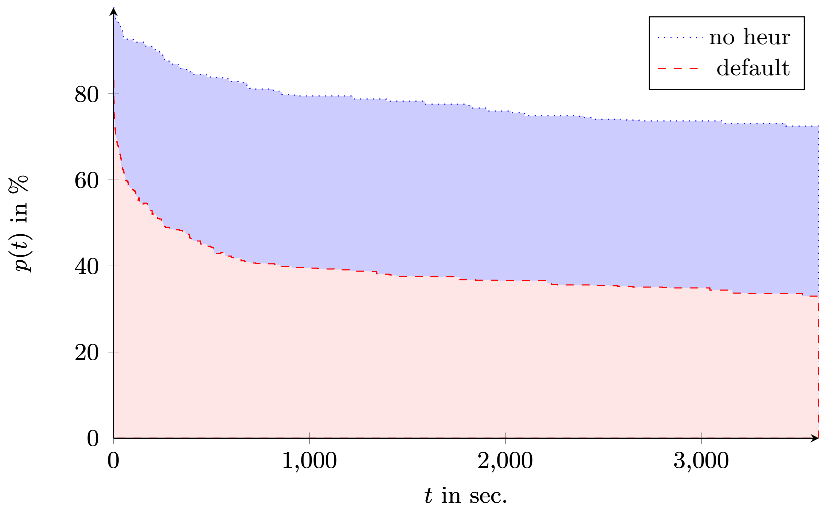


Figure 3.3.: Course of the primal gap when running SCIP on the MIPLIB 2010 challenge test set with and without primal heuristics

(and previously in [JBNP91]) that more than one performance indicator should be used to evaluate mathematical programming software, ideally based on different aspects of the optimization process. Figure 3.4 shows the course of the average primal gap function for CBC, SCIP, XPRESS, GUROBI, CPLEX and a *virtual best solver* (VBS). VBS takes for each instance the minimum of the primal bounds of the five solvers at each point in time. The results for the average primal integral, in particular the order of the solvers, are clearly different from those for the mean time to optimality at [Mit], which shows yet again that different solvers have different strengths.

Even more important, averages tell you little about a single instance. The primal integral of the virtual best solver in Figure 3.4 is a factor 3.8 smaller than the primal integral of the best individual solver, meaning that the portfolio of solvers is significantly better than any single solver. For 82 of the 83 instances, at least one solver found an optimal[16] solution within 800 seconds. In contrast, for each individual solver, there are at least four instances for which the solver did not find an optimal solution after one hour. Each solver contributes to VBS, meaning that each solver has the single best primal bound for some instances for some time. Altogether, this shows once again (compare [KAA$^+$11]) that having a portfolio of MIP solvers is beneficial.

3.5. Variants and extensions

Two main directions for modifications of the primal integral are (i) using a different base measure $p(t)$ and (ii) extending the integral function $P(T)$.

Concerning (i), analogously to the primal integral, we can define a *dual*

[16]Here, we consider a solution $\tilde{x}$ optimal when $\gamma^p(\tilde{x}) \leq 10^{-6}$.

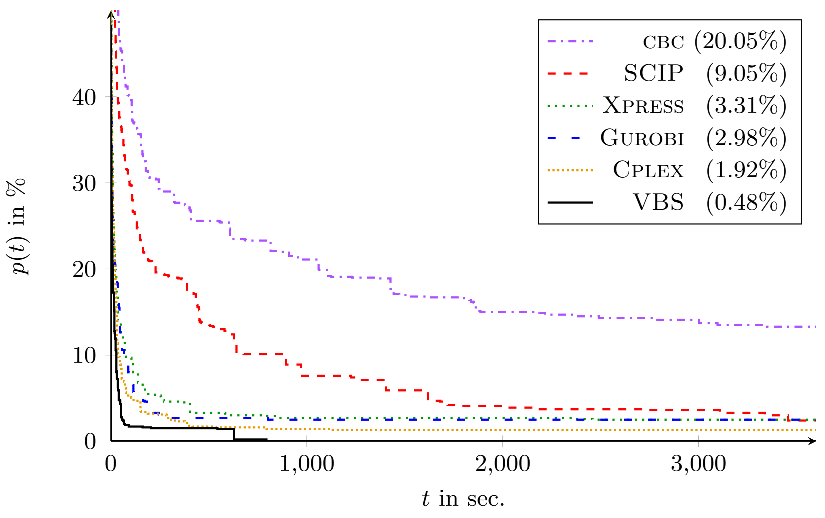


Figure 3.4.: Course of the primal gap for five different MIP solvers plus a virtual best solver

integral by considering a dual gap function between the current and a best known dual bound. Using the "gap closed" (one of the first appearances was in 1985 [JKS85]) as a measure for the performance of a cutting plane algorithm has the same pitfall as using the objective function of the best found solution for primal heuristics: it only considers the final state, ignoring the path that led there.

Taking the integral over the primal-dual gap of a MIP solver can serve as a measure of its convergence speed. Again, this might be particularly worthwhile when a test set contains many instances which hit an imposed time limit. Reporting the gap at termination is prone to variations caused by bound changes around the time limit. Using a *primal-dual integral* instead reduces the impact of events that happen around the time limit: if as an extreme example one solver improves the bounds after 3599 seconds and the other after 3601 seconds, the primal-dual integral will differ by less than 0.1 %, whereas the primal-dual gap after one hour can be arbitrarily different. Unlike the primal integral, a primal-dual integral does not even require the value of an optimal or best known solution as an input.

Concerning (ii), logarithmic scales are often used to put the focus on the factor between measured values rather than on their absolute difference. It can be argued for the time axis as well as for the gap axis that it makes sense to rather use a logarithmic scale, and incorporate the logarithm into the definition of $P(T)$.

We used mixed integer linear programming as a showcase in this chapter, but never exploited specific structures of this problem class. The suggested measures can be used for any class of optimization problem which features a meaningful primal or dual bound, and any algorithm that produces a monotonic sequence of bound values. Similarly, instead of a gap function, any performance measure which evolves monotonically over time could be used.

3.6. Conclusion

In this chapter, we introduced a new performance measure to evaluate mathematical programming software. The *primal integral* takes the development of the incumbent solution over time into account, thereby favoring finding good solutions early – which is extremely important in practice. It is less prone, though not immune, to common weaknesses of standard performance measures, notably the dependence on an (arbitrarily chosen) time limit.

We argued that the primal integral is particularly useful to measure the progress of an optimization procedure w.r.t. solution quality. To this end, we showed that for two state-of-the-art MIP solvers, the primal integral changes by a factor of nearly two when disabling primal heuristics. The running time to optimality only increased moderately in the same experiment.

We conclude that the primal integral and its variants (e.g., a primal-dual integral) are a valuable extension of the portfolio of available performance measures. We will use it for our final evaluation in Chapter 11, where we also apply it for MINLP test sets.

Although we only suggested this new performance measure very recently, it already found its way into several professional software packages. It has been implemented in the paver 2.0 [BDV14] environment for performance analysis, which is maintained by GAMS Development Corp and it is being used to tune emphasis settings of Cplex [Ach]. Further, the primal integral has been employed for computational experiments in some recently published papers and preprints on primal heuristics [AS14, FLM$^+$13, FM13, FMS14, Sal14].

4. Rounding and propagation heuristics for MIP

The start heuristics presented in Chapters 5, 7, and 8 and many other heuristics from the literature [BCD$^+$01, BM80, BSW04, EN07] either solve sequences of linear programs or auxiliary MIPs/MINLPs. Moreover, they rely on an optimal relaxation solution being at hand. Finding this may itself take a significant amount of time.

This chapter focuses on primal heuristics that only employ computationally quick procedures such as rounding and logical deductions. We give an overview of existing rounding and propagation heuristics for mixed integer linear programming. Further, this chapter presents in detail the Shift-and-Propagate heuristic, a primal heuristic that does not require a previously found relaxation solution. For this reason, it is applicable already before the initial root LP relaxation has been solved. Shift-and-Propagate applies domain propagation techniques to quickly drive a variable assignment that satisfies the integrality restrictions towards feasibility.

Shift-and-Propagate is a joint work with Gregor Hendel; he presented a preliminary version of Shift-and-Propagate in his bachelor's thesis [Hen11]. This chapter is based on a paper that has been accepted for publication in Journal of Heuristics, a preprint of this submission has been made available as ZIB-Report [BH15].

This chapter is organized as follows. After a short introduction in Section 4.1, we will recall some rounding and improvement heuristics known from the literature in Section 4.2. All of them are already implemented in SCIP. Then, we give an overview of MIP domain propagation techniques in Section 4.3. After that, we introduce key ideas and implementation details of the Shift-and-Propagate heuristic in Section 4.4, including individual discussions on two of its main components: the shifting value selection and the variable ordering. Finally, we present computational results in Section 4.5.

4.1. Introduction

Chapters 4–8 of this thesis deal with *primal* heuristics. Among those, this chapter is the only one solely concerned with mixed integer *linear* programming. We focus on rounding and propagation heuristics that are applied inside a complete MIP solver. *Rounding heuristics* set each fractional value of an LP solution to an integral value, to make an LP-feasible solution feasible.

Lodi conjectures in [Lod10, Section 16.3.1.2] that a better handling of general integer (as opposed to binary) variables is one of the main challenges for future MIP code development. In constraint programming, using general, though mostly bounded, integer variables is the typical rather than the exceptional case. CP technologies, in particular domain propagation, are designed for handling general integers. *Propagation heuristics*, also known as probing heuristics [Ach04], use domain propagation techniques (see Section 4.3) to reduce the search space and drive a partially assigned solution towards feasibility. In contrast to diving heuristics, see Chapter 2.3 and [Ber06, DRP04], propagation heuristics do not solve LPs to ensure feasibility of the linear constraints. Note that, although rounding heuristics are very fast procedures, they are only applied after the LP relaxation of a given MIP has been solved. This can sometimes take a long time by itself; compare, e.g., the XXL instances from Miplib 2010 [KAA$^+$11].

The main contribution of this chapter is the introduction of a new propagation heuristic, called Shift-and-Propagate, and its computational comparison against three rounding heuristics and two improvement heuristics when used inside a complete MIP solver. Both improvement heuristics employ a light version of domain propagation. The common feature of all these heuristics is that they do not solve LPs – or even sub-MIPs, as opposed to heuristics like rins (see Chapter 6 and [DRP04]).

Shift-and-Propagate is intended to be a component of a complete solver rather than a standalone procedure. For this purpose, we designed it to be a quick procedure, which might sacrifice success on some instances to achieve a good trade-off between the number of found solutions and average running time. It does not require an LP solution as a starting point and can therefore be applied earlier during the solver's search than most other heuristics. Another MIP start heuristic which does not require an LP solution is Rapid Learning, see Chapter 9. It follows a hybrid propagation and large neighborhood search approach.

4.2. Rounding and improvement heuristics

Primal heuristics, in particular those that only employ computationally cheap procedures such as rounding and logical deductions (propagation), are an important component of state-of-the-art MIP solvers. In this section, we describe rounding heuristics and two simple improvement heuristics from the literature [ABH12, Wal10]. They will serve as a comparison for the newly proposed Shift-and-Propagate heuristic later.

4.2.1. Rounding heuristics

The goal of rounding heuristics is to convert an LP-feasible solution $\bar{x}$ into a feasible solution for the MIP by applying rounding strategies to the set of fractional variables $\mathcal{F} := \{j \in \mathcal{I} : \bar{x}_j \notin \mathbb{Z}\}$.

The three rounding heuristics that are implemented in SCIP use the notion of up- and down-locks, see Definition 2.10. To recall: for a MIP, we call the number of positive coefficients $\overline{\kappa}_j := |\{i \mid A_{ij} > 0\}|$ the up-locks of the variable x_j; the number of negative coefficients is called the down-locks $\underline{\kappa}_j$ of x_j. If a variable $j \in \mathcal{F}$ satisfies $\overline{\kappa}_j = 0$ or $\underline{\kappa}_j = 0$, it can be trivially rounded up or down: let $\bar{x}$ be an LP-feasible solution and $\tilde{x}$ be the solution obtained by rounding x_j into the direction of zero locks. Then it holds for all constraints that $A_{i\cdot}^{\mathsf{T}}\tilde{x} \leqslant A_{i\cdot}^{\mathsf{T}}\bar{x} \leqslant b_i$ and hence $\tilde{x}$ is LP-feasible.

The Simple Rounding heuristic [Ber06] uses this property to produce feasible solutions by rounding variables $\tilde{x}_j \leftarrow \lceil \bar{x}_j \rceil$ if $\overline{\kappa}_j = 0$, or $\tilde{x}_j \leftarrow \lfloor \bar{x}_j \rfloor$ if $\underline{\kappa}_j = 0$. It will terminate either with a feasible solution or after detecting a variable $j \in \mathcal{F}$ with both $\overline{\kappa}_j > 0$ and $\underline{\kappa}_j > 0$.

ZI Round [Wal10] reduces the fractionality of an LP-feasible solution step-by-step by shifting fractional values towards integrality, but not necessarily rounding them. For each fractional variable $j \in \mathcal{F}$, the heuristic calculates bounds for both possible rounding directions of $\bar{x}_j$ such that the obtained solution stays LP-feasible. The heuristic then shifts $\bar{x}_j$ by the corresponding bound into the direction which reduces the fractionality $\min\{\bar{x}_j - \lfloor \bar{x}_j \rfloor, \lceil \bar{x}_j \rceil - \bar{x}_j\}$ most. ZI Round might process the set of fractional variables several times. The heuristic either terminates with a feasible solution or aborts if the integer infeasibility could not be decreased anymore or if the heuristic reaches a predefined iteration limit.

In contrast to Simple Rounding and ZI Round, Rounding [Ber06] also performs roundings that potentially lead to a violation of some linear constraints, trying to recover from this infeasibility by further roundings later on. The solutions in the search space of Rounding are a superset of the ones in the search space of Simple Rounding. Like Simple Rounding, the Rounding heuristic takes up- and down-locks of an integer variable with fractional LP solution value $\bar{x}_j$ into account. As long as no linear constraint is violated, the algorithm iterates over the fractional variables and applies a rounding into the direction of fewer locks, updating the *activities* $A\tilde{x}$ of the LP rows after each step, with $\tilde{x}$ being the partially rounded LP solution. If there is a violated linear constraint, hence $A_{i\cdot}^{\mathsf{T}}\tilde{x} > b_i$ for some i, the heuristic will try to find a fractional variable to round into a direction that decreases the violation of constraint i. The number of up- and down-locks serve as a tie breaker. If no rounding can decrease the violation of the constraint, the procedure aborts.

Rounding is a typical example of a heuristic that follows what we call a *fail fast* strategy. *Fail first* is a common principle for child selection heuristics, in particular in artificial intelligence and constraint programming. It has been originally introduced by Haralick and Elliott, who stated: “To succeed, try first where you are most likely to fail” [HE80]. In analogy, we claim that it is a good strategy for primal heuristics to take the most critical decisions first, e.g., try to fix variables on which it seems hardest to achieve feasibility first. This has two advantages. Firstly, it will be easier to repair infeasibility when there are more unfixed variables left. Secondly, if an early decision leads to a failure of the heuristic, at least it did not not use much running time; hence the term *fail fast*. As a marked example, the Rounding heuristic in SCIP fixes variables in non-increasing order of the number of locks in their intended rounding direction.

A few more rounding strategies have been suggested by Christophel in his Diploma thesis [Chr05]. Maybe the most natural one is rounding to the nearest integer values, which has been used before in, e.g., [BKR98] for instances from chemical production planning models. For general MIPs, this strategy rarely gives rise to feasible solutions [Chr05]. When starting from an optimal solution of a relaxation, rounding all fractional variables towards the direction of a good objective function is doomed to failure: the resulting vector would have a better objective function than the relaxation optimum and can therefore not be feasible. Rounding in the opposite direction, however, can work well in particular for pure inequality systems. In [Chr05], this strategy is called *unfavorable rounding*. In the same thesis, a combined strategy called *gap rounding* is suggested, which rounds nearly integral values to the nearest integer and the remaining variables against the objective function. Recently, Naoum-Sawaya [NS13] introduced *(recursive) central rounding*, which is based on the idea to round the analytic center (see [Son86]) of an LP or NLP relaxation of a MIP or MINLP to the nearest integer vector. The intuition is that, at least for general integer variables, a point in the "middle" of the relaxation's feasible region is more likely to have an integer feasible solution in its vicinity as compared to an extremal solution of the relaxation. The Octane heuristic [BCD+01] by Balas et al. can be understood as a rounding heuristic if it is only applied to the space of fractional variables. This heuristic exploits the polyhedral duality of hypercubes and hyperoctahedra and has only been described for BPs. Octane is a ray shooting algorithm that starts from the LP optimum and proceeds along a ray that is directed into the "inner" of the LP's polyhedron. It tests those 0-1 vectors for MIP feasibility that correspond to the first k facets of the unit hyperoctahedron which are hit by the ray.

In [Ber06, ABH12], a propagation heuristic named *Shifting* is described. The Shifting heuristic is similar to Rounding, but it tries to continue in the case that no rounding can decrease the amount of infeasibility of a violated linear constraint. In this case, the value of a continuous variable or an integer variable with integral value will be shifted in order to decrease the violation of the constraint. To avoid cycling, the procedure terminates after a certain number of *non-improving shifts*. A shift is called *non-improving*, if it neither reduces the number of fractional variables nor the number of violated rows. Achterberg [Ach07b] suggests a modified version called Integer Shifting, that is especially designed for MIPs with continuous variables. It relaxes the continuous variables, performs the aforementioned Shifting algorithm, reintroduces the continuous variables and solves a final LP to get optimal values for them.

4.2.2. Improvement heuristics

In addition to the described rounding heuristics, we will also employ two "natural" improvement heuristics, called 1-Opt [Ach07b] and 2-Opt [Hen11].

Taking a MIP-solution $\tilde{x}$ as input, 1-Opt determines for every integer variable $j \in \mathcal{I}$ with $c_j \neq 0$ the shift $\delta_j \in \mathbb{Z}$ with maximum $|\delta_j|$ such that $c_j\delta_j \leqslant 0$ and $A_{i\cdot}^{\mathsf{T}}\tilde{x} + A_{ij}\delta_j \leqslant b_i$ for all i. Hence, shifting the value of x_j in $\tilde{x}$ by δ_j preserves feasibility for all constraints. Shifting any variable x_j with $|\delta_j| \geq 1$ by δ_j will give an improved solution. All found improving shifts are then sorted by increasing value of $c_j\delta_j$ and executed one by one, except a previously executed shift rendered them infeasible. This is a basic version of *constraint propagation*: variable values are inferred from constraint activities.

The 2-Opt heuristic shifts pairs of variables at a time rather than single variables. In the SCIP implementation of 2-Opt [Hen11], integer variables are grouped into smaller blocks based on a predefined ratio of rows that two variables share. During its execution, 2-Opt searches within every such block for variable pairs $\{j_1, j_2\} \subseteq \mathcal{I}^2$ which allow shifts $\delta_1, \delta_2 = \pm\delta_1$ improving the objective $(c_1\delta_1 + c_2\delta_2 < 0)$ and maintaining the feasibility of the solution. Similarly to 1-Opt, all found improving shifts are sorted w.r.t. to the objective improvement and then applied starting with the most improving shift. This approach can easily be extended to a k-Opt or Lin-Kernighan-like [LK73, ACR03] heuristic.

Combining either of these improvement heuristics with any of the rounding heuristics from the previous section yields a greedy algorithm for MIP. Starting from an LP-feasible solution, all presented rounding algorithms apply strategies that favor feasibility over optimality. If a solution is found, it can then be driven towards optimality by the improvement heuristics until it cannot be further improved by switching single variable values or pairs of them.

Note that 1-Opt and 2-Opt can be seen as a variant of Limited Discrepancy Search [HG95], starting with a feasible, but potentially sub-optimal path, and using a discrepancy limit of one and two, respectively. Limited Discrepancy Search, however, typically uses domain propagation, 1-Opt and 2-Opt do not.

This combination is a fast and promising approach to find good solutions early during the solving process, in particular for set covering and packing problems, see also Section 4.3. Here, the described rounding heuristics will always yield a solution where all or at least the vast majority of variables are set to 1 (for covering) or 0 (for packing). The 1-Opt heuristic greedily flips variables in the order determined by their objective coefficients until it reaches a local optimum.

There exist, however, MIPs for which even solving the LP relaxation is hard, such that the LP solver is unable to find a feasible LP-solution within reasonable time. This preempts the application of any the above heuristics. Examples for such MIPs include the XXL instances from Miplib 2010 [KAA+11]. Therefore, we propose the Shift-and-Propagate heuristic, which does not depend on a previously found LP-feasible solution, but uses domain propagation procedures for MIP to find a feasible solution.

4.3. Domain propagation for MIP

Domain propagation (see, e.g., [Bes06, Tac09]) denotes the process of inferring sequences of local domain reductions at the current node of the branch-and-bound tree. The goal is to shrink the size of the current subproblem as much as possible at affordable computational cost. This natural idea is known under many different names, e.g., node preprocessing, bound tightening, range reduction, filtering in different communities such as mathematical programming, constraint programming, satisfiability testing, and artificial intelligence.

In this section, we want to give an overview of existing domain propagation rules for linear constraints. The (local) domain of a variable x_j is the set of values within the (local) lower and upper bounds of x_j,

$$D_j := \{z \in M \mid l_j \leqslant z \leqslant u_j\}$$

for $M \in \{\mathbb{R}, \mathbb{Z}\}$ depending on the variable type of x_j. In mixed integer programming, domain reductions typically consist of tightened variable bounds; holes in the interval $[l_j,\ u_j]$ are not considered. Reductions on variable bounds from the *activity* of linear constraints were first established in Brearly et al. [BMW75]. Savelsbergh [Sav94] extended these methods by probing techniques on binary variables and constraints, while Andersen and Andersen [AA95] exploited further presolving techniques for linear programming. For an overview on presolving techniques in MIP, see [Mah10], for a recent progress report, see [GKM$^+$13].

For the suggested Shift-and-Propagate heuristic, domain propagation is a crucial step. In this section, we review which domain propagation rules the different constraint types in SCIP use. For more information and implementation details, see Achterberg [Ach07b].

Note that domain propagation rules are applied at two different stages of the MIP solving process. First, they are applied during preprocessing before the branch-and-bound search starts. In this case the deductions hold globally for the problem. Second, they are used locally at nodes within the branch-and-bound tree to infer reductions from the branching decisions. In the following, we focus on local propagation during search, and use l, u for local bounds at a branch-and-bound node.

General linear constraints

In this chapter, we denote the (column) vector corresponding to the i-th row of the constraint matrix A by $A_{i\cdot}$. Thus, the i-th linear constraint of a given MIP can be written as $A_{i\cdot}^{\mathsf{T}} x \leqslant b_i$. Given a vector $\bar{x} \in \mathbb{R}^n$ and a linear constraint $A_{i\cdot}^{\mathsf{T}}x \leqslant b_i$, the value $A_{i\cdot}^{\mathsf{T}}\bar{x}$ is called the *activity* of the constraint w.r.t. $\bar{x}$. Taking into account the domains of all variables in a particular linear constraint, its *minimum and maximum activity* [BMW75] $\underline{\alpha}$ and $\overline{\alpha}$ are defined by

$$\underline{\alpha} := \min\left\{A_{i\cdot}^{\mathsf{T}}x \mid l \leqslant x \leqslant u\right\} \text{ and } \overline{\alpha} := \max\left\{A_{i\cdot}^{\mathsf{T}}x \mid l \leqslant x \leqslant u\right\}.$$

In the same way, we obtain the *minimum and maximum residual activity* for variable x_j,

$$\begin{aligned}\underline{\alpha}'_j &:= \min\left\{A_{i\cdot}^{\mathsf{T}}x - A_{ij}x_j \mid l \leqslant x \leqslant u\right\} \text{ and}\\ \overline{\alpha}'_j &:= \max\left\{A_{i\cdot}^{\mathsf{T}}x - A_{ij}x_j \mid l \leqslant x \leqslant u\right\} \text{ resp.,}\end{aligned}$$

by excluding the contribution of variable x_j.

By these definitions, it is possible to deduce bounds on the variables from the constraints they appear in. For a positive coefficient A_{ij} of variable x_j in the i-th row/constraint, it holds that

$$x_j \leqslant \frac{b_i - \underline{\alpha}'_j}{A_{ij}}.$$

This provides a tighter variable bound whenever $u_j > \frac{b_i - \underline{\alpha}'_j}{A_{ij}}$. If x_j is an integer variable, the new upper bound can be rounded down. An analogous inference rule holds for lower bounds in the case of negative coefficients.

Besides the tightening of variable domains, minimum and maximum activities can also proof local redundancy or infeasibility of a constraint [BMW75].

Special classes of linear constraints

For linear constraints of special form, there often exist stronger or faster propagation algorithms. In his thesis [Ach07b], Achterberg summarized special techniques for the following linear constraint classes: variable lower and upper bounds, knapsack, set covering, set partitioning, and set packing.

Of course, all of them could be propagated by algorithms for general linear constraints, but their special structure allows for a more efficient implementation of the propagation routines. In this section, we briefly describe propagation algorithms for knapsack and set covering constraints.

Knapsack constraints

A *knapsack* constraint is a linear constraint in which all involved variables are binary variables, and all coefficients $A_{ij} =: w_j$ plus the right hand side b_i are positive integer values, called the *weights* and the *capacity*, respectively. Note that every linear constraint $A_{i\cdot}^{\mathsf{T}}x \leqslant b_i$ in which all variables are binary and all coefficients plus the right hand side are rational values can be transformed into a knapsack constraint, see, e.g., [Ach07b]. Propagation routines for knapsack constraints can use efficient integer arithmetic – instead of floating point arithmetic. Second to that, the only possible reduction is fixing a variable x_j to 0 if the weighted sum of the variables fixed to 1 and the weight w_j together exceed the right hand side. Let

$$K^1 := \{j \in \mathcal{I} \mid w_j > 0 \ \wedge \ x_j \text{ fixed to } 1\}$$

denote the set of variables which are already fixed to 1 and $w^1 := \sum_{j \in K^1} w_j$ the sum of their weights. The domain propagation rule then reads

$$(j \text{ unfixed}) \ \wedge \ (w^1 + w_j > b) \ \Rightarrow \ (x_j \leftarrow 0)$$

Since the weights are nonnegative, and all involved variables are binary, sorting the variables in nonincreasing order of their weights allows for a performance improvement.

SCIP also features methods to extract (negated) *clique-information* about the binary variables of a problem. A (negated) clique is a set of binary variables of which at most one variable can take the value 1 (0) in a feasible solution.

Let therefore denote K^0 the set of variables fixed to 0, $C \subseteq \mathcal{I} \setminus (K^1 \cup K^0)$ denote a negated clique of unfixed variables, $w(C)$ be the sum of weights of variables in C, $j_{\max} := \operatorname{argmax}_{j \in C} w_j$ be a clique variable of maximum weight, and $w_{\min}(C) := w(C) - w_{j_{\max}}$ be the minimum weight of this negated clique in any feasible solution. The following reductions are now possible considering negated clique information:

1. $(w^1 + w_{\min}(C) > b) \ \Rightarrow$ (the subproblem is infeasible)
2. $(j \in C \setminus \{j_{\max}\}) \ \wedge \ (w^1 + w_{\min}(C) - w_j + w_{j_{\max}} > b) \ \Rightarrow \ (x_j \leftarrow 1)$

See Winkler [Win14] for more information on propagation of (negated) cliques.

Set covering constraints

Set covering constraints have the form

$$x_{j_1} + \cdots + x_{j_k} \geqslant 1, \text{ for } \{j_1, \ldots, j_k\} =: K \subseteq \mathcal{N},$$

where K contains only binary variables. The only domain reduction to be inferred from a set covering constraint is to fix a variable x_j to 1 if all other variables $j' \in K \setminus \{j\}$ have already been fixed to 0. The state-of-the-art algorithm to keep track of the bound changes in this case was introduced in [MMZ+01]: the *two-watched-literals scheme* provides a significant speedup in propagation.

For an overview on presolving and propagation strategies for set partitioning constraints, see Borndörfer [Bor98]. An recent overview on presolving and domain propagation techniques for set covering constraints is given by Winkler [Win14].

4.4. Shift-And-Propagate

In this section, we introduce the Shift-and-Propagate heuristic. The purpose of this primal heuristic is finding a feasible MIP solution at an early stage of the solution process before solving the root LP. In addition, it is designed to be computationally cheap, using domain propagation techniques instead of solving LPs to ensure feasibility of the linear constraints.

The basic idea is as follows: in each iteration, the heuristic selects an unfixed variable $j \in \mathcal{K}$ and a fixing value t_j^* inside the domain of x_j, to which it shifts the variable. Then, the heuristic applies domain propagation routines to infer further reductions from this fixing. If domain propagation detects that fixing $x_j \leftarrow t_j^*$ is infeasible, a one-level backtracking strategy is applied. Otherwise, the heuristic proceeds with the next unfixed variable. The goal of Shift-and-Propagate is to find a good start solution, before the root node processing of a MIP solver starts, in particular prior to the first LP. This solution might then serve as a reference point for improvement heuristics (see Section 4.2.2) and for inferring further domain reductions (e.g., by propagating the maximum activity of the objective function, see Section 4.3).

The general algorithm is described in Algorithm 4.1. The main degrees of freedom are the variable selection in line 3, the choice of a promising fixing value (line 4), and the backtracking strategy including an appropriate stopping criterion (line 7). These are the main topics for the remainder of this section. We will first discuss different variable orders, then introduce an algorithm to select a best shifting value, and finally present a full version of the Shift-and-Propagate algorithm, including considerations on backtracking. It was a crucial step in our implementation, to limit the number of backtracks and propagation rounds.

Implementation details

For the ease of presentation, we assume from now on, w.l.o.g., that all variables have a lower bound of 0. Otherwise, we apply a suitable problem transformation: variables with finite lower bound are shifted, variables with infinite lower bound but finite upper bound are negated (and shifted), and free variables are decomposed into a negative and a positive part.

Shift-and-Propagate starts with the *zero-assignment* $\tilde{x} \leftarrow 0$ which respects all variable bounds. Therefrom, the activity of all linear constraints is zero as well. Hence, an assignment is feasible for a linear constraint, if and only if it has a nonnegative right hand side $b_i \geqslant 0$. Subsequent fixing steps $D_j \leftarrow t_j^*$ are then processed as shifts, which only affect those constraints in which the variable x_j is involved in. Instead of updating the activity explicitly after every shift, activities are maintained implicitly by changing the right hand side.

Note that we distinguish between an assignment to a variable, which is a temporary solution value, equal to one of the bounds of the variable, and

Figure 4.1.: The basic Shift-and-Propagate algorithm

```
Input   : MIP problem $P$
Output: a feasible solution of $P$, or NULL
1  $\mathcal{K} \leftarrow \mathcal{I}$, $\tilde{x} \leftarrow 0$;
2  while $\mathcal{K} \neq \emptyset$ do
3      select $j \in \mathcal{K}$;
4      choose $t_j^* \in D_j$;
5      propagate $D_j \leftarrow \{t_j^*\}$;
6      if propagation detects infeasibility then
7          apply one-level backtracking strategy;
8      else
9          $\tilde{x}_j \leftarrow t_j^*$ ;
10         $\mathcal{K} \leftarrow \mathcal{K} \setminus \{j\}$ ;
11 if $\tilde{x}$ is feasible for $P$ then
12     return $\tilde{x}$;
13 return NULL;
```

a fixing, which is a final choice and can be equal to a value in the interior of the variable's domain. An assigned value may be altered multiple times through the effects of domain propagation, a fixing is made (at most) once per variable. Once a variable is fixed, it cannot be selected any longer in line 3 of Algorithm 4.1. Domain propagation in line 5 uses only the values of fixed variables and the bounds of unfixed ones. The assignments of the unfixed variables do not affect the propagation. Note that bounds can be changed, in particular variables can be fixed, as an effect of domain propagation.

Continuous variables are not handled directly by Shift-and-Propagate but treated as row slacks and projected out of the problem formulation. For a row $A_{i\cdot}x \leqslant b_i$, let x_j be a variable with positive row coefficient $A_{ij} > 0$ and domain $D_j = [l_j, u_j]$. Then, it holds that

$$A_{i\cdot}x \leqslant b_i \Rightarrow A_{i\cdot}x - A_{ij}x_j \leqslant b_i - A_{ij}l_j.$$

By subtracting the minimum activity of the variable from the right hand side $b_i \leftarrow b_i - A_{ij}l_j$ and setting $A_{ij} \leftarrow 0$, we obtain a relaxed formulation of the i-th row without x_j. For negative coefficients, the argument remains the same except that the lower bound l_j is replaced by the upper bound u_j. Note that the transformed row is trivially satisfied whenever the variable bound is infinite.

If the heuristic finds a solution on the integer variables in this transformed space, a final LP with all integer variables fixed to their heuristic values is solved to obtain values for the continuous variables. There are two advantages of this final LP compared to solving an LP relaxation beforehand. First, the final LP will be smaller and often significantly easier to solve, since all integer variables are fixed, and the fixings have been propagated. In particular, for pure IPs, this stage is completely omitted. Second, it is only employed when the heuristic found a consistent assignment for the integer variables and merely the continuous ones need to be adjusted.

As for the variable order, the heuristic sorts the variables nonincreasingly w.r.t. the initial number of violated rows they appear in,

$$|\{i \mid A_{ij} \neq 0 \text{ and } b_i < 0\}|\,.$$

We also tested a different variable order using the *importance* of a variable column, which Hendel defined in his bachelor's thesis [Hen11] as

$$\sum_{i=1}^{m} |A_{ij}| + |\{i \mid A_{ij} \neq 0\}|\,.$$

We compare different variable orders in Section 4.5.

The choice of a fixing value for the selected variable is obtained by a *best shift selection* which is based on the feasibility state of the row w.r.t. the current assignment. Therefore, we keep track of how many violated rows can be made feasible and vice versa by a certain shift.

Definition 4.1. *For an LP-row $i : A_{i\cdot}^{\mathsf{T}} x \leqslant b_i$ and an unfixed variable $j \in \mathcal{K}$, we define*

$$\begin{aligned} \Psi_i^j :\ & D_j \to \{-1, 0, 1\} \\ & t \mapsto \begin{cases} 1, & \text{if } b_i \geqslant 0 \text{ and } b_i - A_{ij} \cdot t < 0, \\ -1, & \text{if } b_i < 0 \text{ and } b_i - A_{ij} \cdot t \geqslant 0, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

We call Ψ_i^j the row violation function *of row i. The row violation functions of all rows sum up to*

$$\begin{aligned} \Psi^j :\ & D_j \to \mathbb{Z} \\ & t \mapsto \sum_{i=1}^{m} \Psi_i^j(t). \end{aligned}$$

We call Ψ^j the row violation sum function *of x_j. For a particular $t \in D_j$ we call $\Psi^j(t)$ the* violation balance *of t.*

For every row violation function, it holds that $\Psi_i^j(0) = 0$, and that it changes its value at most once on D_j. The violation balance $\Psi^j(t)$ is a measure of how the overall feasibility of a partial assignment changes by a particular shift t of the variable. A negative value means that more rows will be made feasible than infeasible by shifting variable x_j by a value of $t \in D_j$. The function Ψ^j is a step function with at most M_j steps, M_j

Figure 4.2.: Algorithm to determine the best shift of a variable for the current partial solution

```
   Input  : MIP $P$, integer variable $x_j$ with domain $D_j$
   Output: Best shift $t^*$ for variable $x_j$
1  $Q \leftarrow \emptyset$;
   /* collect row violation functions of the variable                */
2  foreach row $i$ with $A_{ij} \neq 0$ do
3      if $b_i < 0$ and $A_{ij} < 0$ then
4          $t \leftarrow \lceil \frac{b_i}{A_{ij}} \rceil$;    /* min. shift of $x_j$ to make row $i$ feas.  */
5          if $t \in D_j$ then $Q \leftarrow Q \cup (t, -1)$;
6      else if $b_i \geqslant 0$ and $A_{ij} > 0$ then
7          $t \leftarrow \lfloor \frac{b_i}{A_{ij}} \rfloor + 1$;    /* min. shift of $x_j$ to violate row $i$ */
8          if $t \in D_j$ then $Q \leftarrow Q \cup (t, 1)$;
9  if $Q = \emptyset$ then return 0;
10 $\sigma \leftarrow 0$, $t^* \leftarrow 0$, $t_{\text{before}} \leftarrow 0$, $\Psi^* \leftarrow 0$;
   /* summation gives the row violation balance                      */
11 foreach $(t_i, \Psi_i^j(t_i)) \in Q$ in nondecreasing order of $t_i$ do
12     if $t_i > t_{before}$ and $\sigma < \Psi^*$ then
13         $\Psi^* \leftarrow \sigma$, $t^* \leftarrow t_{\text{before}}$;
14     $t_{\text{before}} \leftarrow t_i$;
15     $\sigma \leftarrow \sigma + \Psi_i^j(t_i)$;
   /* takes highest step value into account                          */
16 if $\sigma < \Psi^*$ then $t^* \leftarrow t_{\text{before}}$;
17 return $t^*$;
```

being the number of nonzeros for variable x_j. The best shift selection, see Algorithm 4.2, searches for a value t^* which minimizes the violation balance,

$$t^* \leftarrow \operatorname*{argmin}_{t \in D_j} \Psi^j(t).$$

Since $\Psi^j(0) = 0$, the heuristic will prefer a shifting value different from 0 if and only if there is a value $t' > 0$ with $\Psi^j(t') < 0$, i.e., if shifting by t' reduces the number of currently violated rows. The general idea of selecting values that minimize the number of violated constraints is not new and has, e.g., been used in [MJPL90]. For a recent overview of variable and value selection methods in CP, see Pesant et al. [PQZ12].

In Algorithm 4.2, the row violation functions are interpreted as tuples

$$(t_i, \Psi_i^j(t_i)) \in D_j \times \{-1, 1\}$$

where t_i is the smallest (always positive) value for which the row changes its feasibility state, and $\Psi_i^j(t_i) \in \{-1, 1\}$, depending on the kind of change in the feasibility state. The t_i is called the *step value*. An empty tuple means that no value in the variable domain alters the feasibility state of the row. Algorithm 4.2 collects the row violation functions for all rows i with nonempty tuples $(t_i, \Psi_i^j(t_i))$ and sorts them in nondecreasing order of the t_i. Since different rows $i \neq i'$ might have the same step value $t_i = t_{i'}$ the sum of row violation functions processed so far is stored in the variable σ and yields the row violation balance $\Psi^j(t_{\text{before}})$ when $t_i \gneqq t_{\text{before}}$, i.e., two subsequent step values are different. The method returns a value, $t^* \in D_j$, minimizing Ψ^j. The best shift selection could also be used for continuous variables, omitting the rounding in lines 4 and 7 of Algorithm 4.2.

The complete Shift-and-Propagate procedure is shown in Algorithm 4.3. It uses the best shift selection as a subroutine to select a promising fixing value for some variable x_j from the set of unfixed variables $\mathcal{K}$. The fixing is then performed by shrinking the variable domain to the single value t_j^*. The **propagate**-callback of SCIP internally calls domain propagation algorithms, some of which were mentioned in Section 4.3, to deduce domain reductions for other variables. This is repeated until no further reduction is found or until a predefined iteration limit is reached. In our experiments, we used an iteration limit of 10.

An empty domain of some variable after the propagation will cause the algorithm to apply a one-level backtrack in line 13. If the attempted shift value was one of the variable bounds, $t_j^* \in \{l_j, u_j\}$, the variable domain is tightened by $D_j \leftarrow D_j \setminus \{t^*\}$ and repropagated. If the repropagation of the shrunk domain also detects an empty domain, the execution method of the heuristic is stopped (line 17).

Two cases remain to be handled: either the original shifting value was in the inner of the variable's domain (only for general integer variables) or the propagation of the shrunk domain did not lead to infeasibility. In both cases, the variable is assigned to the set of suspicious variables $\mathcal{S}$ in line 12 and thus removed from the set of unfixed variables $\mathcal{K}$ (line 20). This ensures that the best shift selection is only performed at most once for every variable. In our implementation, we use a limit on the number of backtracks performed; if it is exceeded, the heuristic stops.

We conclude this section with a small example MIP, for which we describe the course of the algorithm in detail. Figure 4.4 visualizes the example.

Example 4.2. *Let P_{ex} be the following MIP with three integer variables $\mathcal{I} = \{1, 2, 3\}$ and $m = 3$ constraints:*

$$\begin{aligned} \min \quad & 0 \\ 2x_1 - x_2 - x_3 &\leqslant 1 \\ -x_1 - x_2 &\leqslant -2 \\ -3x_1 + x_3 &\leqslant -3 \\ x_1, x_2, x_3 &\in \{0, 1, 2\}. \end{aligned}$$

Figure 4.3.: The complete Shift-and-Propagate algorithm

```
Input  : MIP $P$ with constraint matrix $A$ and right hand side $b$,
         a limit $bt_{lim}$ on the number of performed backtracks
Output: a feasible solution $\tilde{x}$ for $P$, or NULL
1  $P_{\text{orig}} \leftarrow P$;
2  relax continuous variables s.t. $\mathcal{R} = \emptyset$;
3  $\mathcal{K} \leftarrow \mathcal{I}$;
4  sort $\mathcal{K}$ w.r.t. to the number of initially violated rows;
5  $\mathcal{S} \leftarrow \emptyset$, $\tilde{x} \leftarrow 0$;            /* $\mathcal{S}$: variables that caused cutoff */
6  while $K \neq \emptyset$ and $b \ngeqslant 0$ do
7      transform $P$ s.t. $l = 0$;      /* has to be ensured each round */
8      select next $j \in K$;
9      $t_j^* \leftarrow$ bestShift($P, j, D_j$);
10     propagate $D_j \leftarrow \{t_j^*\}$;
11     if $\exists k \in \mathcal{K}$: $D_k = \emptyset$ then
12         $\mathcal{S} \leftarrow \mathcal{S} \cup \{x_j\}$;
13         if $|S| > bt_{lim}$ then return NULL;
           /* if empty domain, undo propagation, unfix $x_j$       */
14         one-level backtrack;
15         if $t_j^* \in \{l_j, u_j\}$ then
16             propagate $D_j \leftarrow D_j \setminus \{t_j^*\}$;
17             if $\exists k \in \mathcal{K}$: $D_k = \emptyset$ then return NULL;
18     else
19         $\tilde{x}_j \leftarrow t_j^*$;
20     $\mathcal{K} \leftarrow \mathcal{K} \setminus (\{j \in \mathcal{I} \mid |D_j| = 1\} \cup \mathcal{S})$;
21 if $b \geqslant 0$ then
22     foreach $j \in \mathcal{K} \cup \mathcal{S}$ do
23         $\tilde{x}_j \leftarrow 0$;          /* all unfixed variables are set to 0 */
24     $\tilde{x}_{\text{orig}} \leftarrow$ retransform $\tilde{x}$;        /* undo transform. from line 7 */
25     if $P_{orig}$ contains continuous variables then
26         fix integer variables to $\tilde{x}_{\text{orig}}$ and solve the remaining LP;
27         if LP infeasible then return NULL;
28     return $\tilde{x}_{\text{orig}}$;
29 return NULL;
```

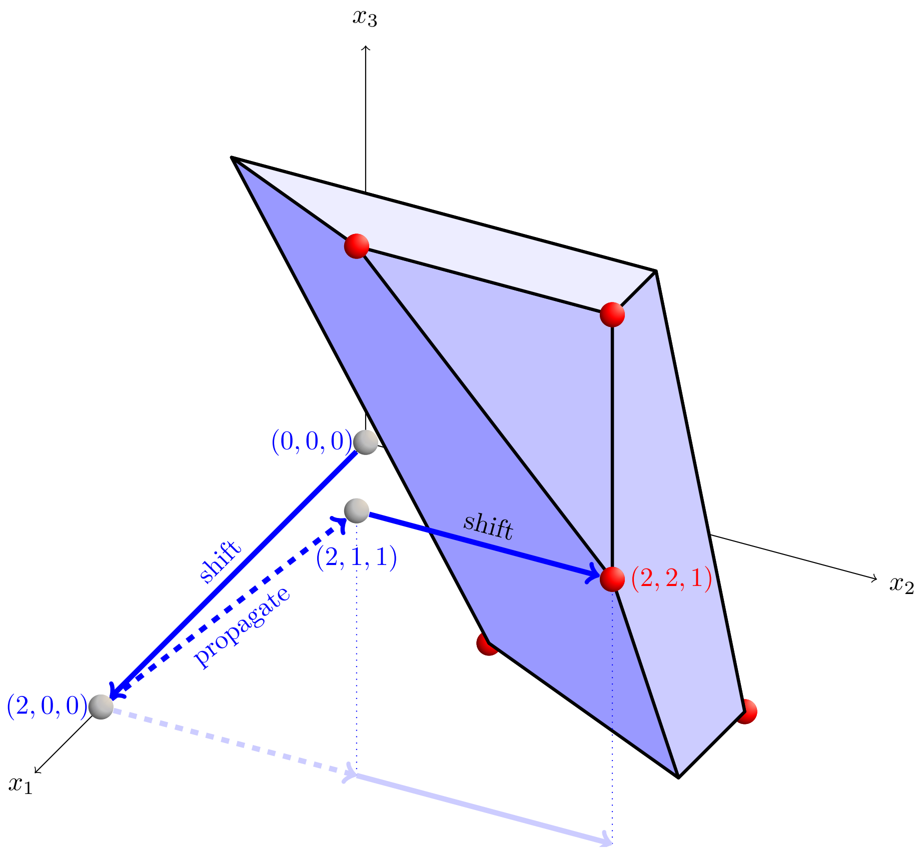


Figure 4.4.: Shift-and-Propagate for the MIP defined in Example 4.2. The shaded volume shows the polytope defined by the inequalities and variable bounds. The path shows the course of the Shift-and-Propagate algorithm. Solid arcs depict shifting (value selection) steps; the dashed line indicates the effects of propagation after fixing $x_1 = 2$.

P_{ex} is a pure feasibility problem. The zero-assignment is not feasible, because both the second and the third row have a negative right hand side.

The heuristic starts with x_1, x_2, and x_3 appearing in two, one and one violated row, respectively, so the sorting will keep the variables in place and select x_1 to start with.

The row violation functions and the row violation sum function for variable x_1 from the example P_{ex} are depicted in Figure 4.5. The first row will be made (temporarily) infeasible by a shift of 1 or 2, whereas the third will be made feasible by a shift of either 1 or 2. The second row, however, requires a shift of 2 to be made feasible. Therefore, the algorithm will select $t_1^* = 2$ as shifting value for x_1 because it minimizes the violation balance $\Psi^1(2) = -1$. Fixing the domain of x_1 to the single value $D_1 \leftarrow \{2\}$ and subtracting the

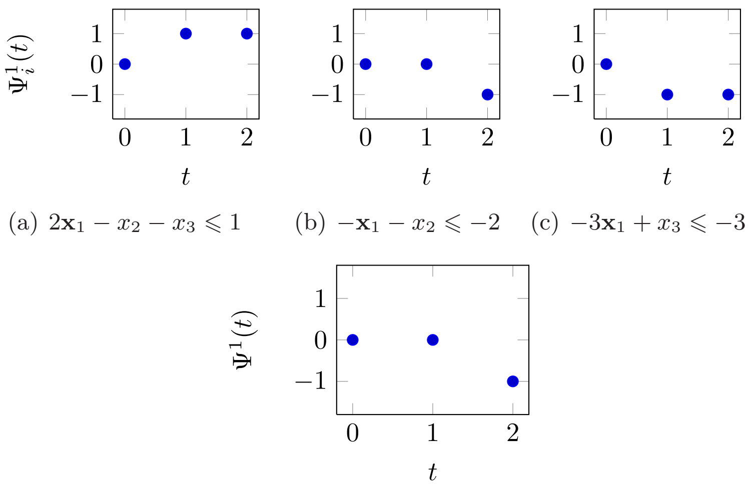


Figure 4.5.: The row violation functions for variable x_1 from P_{ex} for every row (smaller pictures) and the resulting row violation sum function $\Psi^1(t)$

contribution from the right hand side yields a reduced problem

$$\begin{aligned} -x_2 - x_3 &\leqslant -3 \\ -x_2 &\leqslant 0 \\ x_3 &\leqslant 3 \\ x_2, x_3 &\in \{0,1,2\}. \end{aligned}$$

The feasibility state of every row has changed by the shift of variable x_1. The second and third row are trivially satisfied in this reduced problem. The propagation from Section 4.3 applied to the first row leads to further reductions: 0 is excluded from the variable domains $D_2 = D_3 = \{1,2\}$. Since the heuristic requires lower bounds 0 for every variable, the variables are formally replaced by $\bar{x}_{\{2,3\}} \leftarrow x_{\{2,3\}} - 1$, which leads to a changed right hand side vector. The first row now reads

$$\begin{aligned} -\bar{x}_2 - \bar{x}_3 &\leqslant -1 \\ \bar{x}_2, \bar{x}_3 &\in \{0,1\}. \end{aligned}$$

A shift of variable $\bar{x}_2$ by 1 is selected by Algorithm 4.2 and finally makes the row feasible again. Since the right hand side $\bar{b}$ is now nonnegative, the remaining variable $\bar{x}_3$ can be set to 0 without violating a row. Untransforming all bound changes, the heuristic finds the solution $\tilde{x} = (2,2,1)$.

4.5. Computational experiments

In this section, we present the results of two computational experiments. First, we evaluate the impact of different variable orders on the performance

Table 4.1.: Description of variable sortings tested for Shift-and-Propagate

setting	Variable x_j precedes x_i, $i \neq j$, if x_j ...
$\lvert\{b < 0\}\rvert \downarrow$	... appears in more initially violated rows than x_i.
$\lvert\cdot\rvert \downarrow$	... has a higher importance than x_i.
$\lvert\{b < 0\}\rvert \uparrow$	... appears in less initially violated rows than x_i.
$\lvert\cdot\rvert \uparrow$	... has a smaller importance than x_i.

of Shift-and-Propagate. Second, we compare Shift-and-Propagate to rounding and improvement heuristics that are implemented in SCIP in order to get a performant setting for applying cheap heuristics at the root node of a MIP solver.

We used SCIP 3.0 with SoPlex 1.7.0 [Sop] as the underlying LP-solver. The results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20 GHz with 12 MB cache and 48 GB main memory, running an openSuse 12.1 with a gcc 4.6.2 compiler. Hyperthreading and Turboboost were disabled. In all experiments, we ran only one job per node to avoid random noise in the measured running time that might be caused by delays if multiple processes share common resources, in particular the memory bus.

For all benchmarks, we chose the mmm test set which comprises 168 MIP instances from the three publicly available libraries Miplib 3.0 [BCMS98], Miplib 2003 [AKM06], and the Miplib 2010 benchmark set [KAA$^+$11]. We excluded the three instances `ash608gpia-3col`, `enlight14`, and `ns1766074` which are infeasible and `ex9` which SCIP 3.0 solves to optimality by preprocessing plus solving the root LP.

As a measure for the quality of a solution $\tilde{x}$ for an instance P, we consider the primal gap, see Definition 2.5. By the choice of $\gamma^p(\cdot)$, every solution has a gap $\gamma^p(\tilde{x}) \leqslant 100\,\%$.

In a first experiment, we compare the impact of using different variable orders. We regard sorting variables by their importance, see Section 4.4, by the initial number of infeasible rows, or randomly. The importance-based and the violation-based sorting are denoted by the symbols $|\cdot|$ and $|\{b < 0\}|$, respectively, together with an arrow indicating the sense in which the variables are processed: $\downarrow$ for nonincreasing and $\uparrow$ for nondecreasing. The symbol $|\{b < 0\}| \downarrow$, e.g., stands for "sorting w.r.t. the number of initially violated rows, nonincreasing". A description of the four variable settings that we tested (besides random) can be found in Table 4.1.

For all settings, we disabled cutting plane routines in order to avoid the LP relaxation to be re-solved and to make sure that the primal solutions were indeed obtained by the Shift-and-Propagate heuristic.

The performance of the five variable sortings is shown in Table 4.2 (see Table B.1 in the appendix to see results on particular instances) and compared with respect to three criteria: The average solution gap $\bar{\gamma^p}$, the absolute number of solutions obtained (#sols), and the geometric mean of the time

Table 4.2.: Aggregated results for tested variable sortings

setting	$\bar{\gamma}^p$ (%)	#sols	t_{heur} (s)
$\lvert\{b < 0\}\rvert \downarrow$	84.73	88	1.21
$\lvert\cdot\rvert \downarrow$	84.98	84	1.20
random	85.95	84	1.17
$\lvert\{b < 0\}\rvert \uparrow$	87.66	83	1.17
$\lvert\cdot\rvert \uparrow$	86.85	81	1.20
Best	78.61	102	1.11

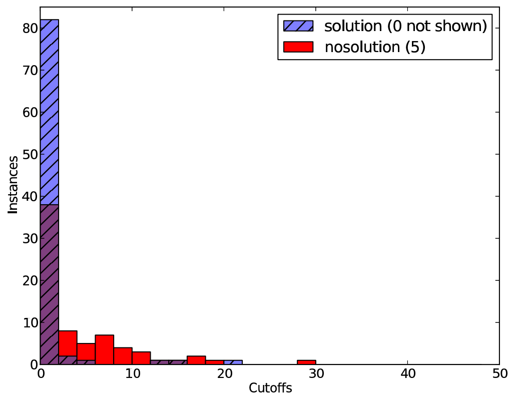
spent by the heuristic t_{heur} (s). The average gap ranges from 84.7 % to 87.7 %, and the number of found solutions differs by up to 7. The setting $|\{b < 0\}| \downarrow$ scores best regarding the number of found solutions and also the average gap, but is the slowest setting w.r.t. its geometric mean time.

A closer look at the results on specific instances shows more variety than the overall results: A feasible solution for `rail507` was found with two of the five settings, namely $|\cdot| \uparrow$ and $|\{b < 0\}| \uparrow$, which took 13.78 s and 14.42 s, respectively. This seems moderate compared to other settings, which terminate unsuccessfully after up to 3600 s (the time limit) with $|\{b < 0\}| \downarrow$. This is particularly undesirable knowing that the LP solve only takes 10 s on this instance.

Reasons for long running times on some instances are the propagation itself, combined with a huge number of backtracking operations due to decisions which lead to cutoffs, and/or the number of iterations of the internal LP solve, which can take itself a long time if the number of continuous variables is large. For the second experiment, where limits on the number of backtracks and propagation rounds have been employed, these times reduce drastically, see below.

Table 4.2 has a sixth setting named Best presenting the union of the five settings, choosing the quickest setting that reaches the best gap in an instance. The total number of instances for which Best succeeds is 102, which is 14 more than the best single setting in this respect. Best reaches an average gap of 78.61 %, which is considerably better compared to the individual settings.

Figure 4.6 presents histograms of how many cutoffs were effectively produced until a feasible solution was found (blue bar), or until the heuristic terminated without a solution (red bar). Some instances are not included in the table since the horizontal axis is truncated at 50. Those numbers are indicated in the legend instead. We show these histograms for two different settings: $|\{b < 0\}| \downarrow$ and random. These charts reveal two pieces of information. First, the number of instances on which the heuristic finds a solution without or with only a single backtrack (0 or 1 cutoff, the leftmost bar) is higher for the $|\{b < 0\}| \downarrow$ order than for the random order by almost 10 in-



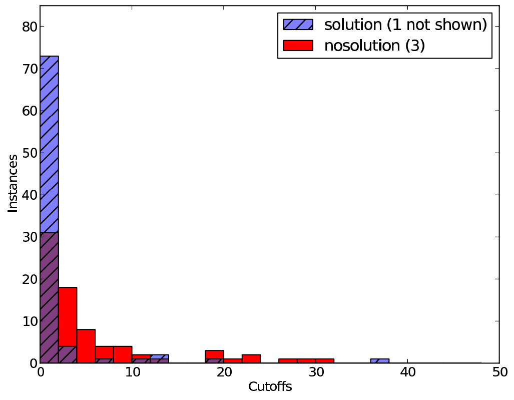


Figure 4.6.: Distribution of the instances over the number of cutoffs Shift-and-Propagate produced during its search for two different variable sortings: $|\{b < 0\}| \downarrow$ and random

stances. Second, the bars for $|\{b < 0\}| \downarrow$ lie underneath those for the random sorting, indicating that the sorting method produces less cutoffs on average. Thus, the sorting method $|\{b < 0\}| \downarrow$ has the advantage to find more feasible solutions, while producing less cutoffs. We will use this method for the next experiment.

The observation that solutions were typically obtained after a small number of cutoffs further suggests that a small absolute limit on the number of cutoffs as, e.g., 10, will only slightly decrease the number of produced solutions but trigger a quicker termination on a significant amount of instances for which the heuristic is not successful. In particular, for the instance `rail507`, using a cutoff limit of 10 leads to a running time of 8.25 s instead of hitting the time limit.

The second experimental setup compares the number of found solutions and the average solution quality obtained with the rounding and improvement heuristics from Section 4.2 to the quality of the Shift-and-Propagate heuristic after processing the root node. Here, the setting *RandI* uses only rounding and improvement heuristics. The setting *SandP* uses Shift-and-Propagate

Table 4.3.: Results from the root node solve for three different settings

setting	$\bar{\gamma}^p$ (%)	#sols	t (s)
SandP	85.05	85	2.40
RandI	81.40	60	2.36
Both	70.27	96	2.41

as only primal heuristic. A third setting *Both* refers to a combination of those heuristics. Neither Shift-and-Propagate nor rounding heuristics take the objective function into account, but improvement heuristics do. It can therefore not be expected that *SandP* outperforms *RandI* in terms of solution quality. The hope is rather that it is beneficial in terms of found solutions, and that we can observe a combined effect in the *Both* setting.

In this experiment, we used limits on the number of backtracks (at most fifteen in total) and propagation rounds (at most ten per iteration). As a consequence, the running times are now less than a minute for all instances. There are only three instances, namely `mspp16`, `neos-476283`, and `netdiversion`, for which the running time is more than ten seconds. We conclude that the employed limits enable us to make efficient use of Shift-and-Propagate within a global solver.

All instances are presented in Table B.2 in the appendix, together with their optimal or best known solution. For each setting, the table depicts the achieved primal bound as well as the solving time (which includes presolving and the processing of the root node) and the time spent on heuristics.

In Table 4.3, we present the results obtained for these different settings. The column $\bar{\gamma}^p$ shows the average gap for all instances from the test set. In the column #sols, the total number of instances is shown for which a solution was found. The last column shows the geometric mean of the overall solving time. The table shows that the use of Shift-and-Propagate leads to a primal feasible solution on 85 instances, whereas rounding and improvement heuristics alone can only contribute solutions to 60 instances of the test set. Both settings combined find a solution on 96 instances or 60 % more than without Shift-and-Propagate. The average gap obtained by SandP was 85.05 %, compared to an average gap of 81.40 % when using RandI. Here, instances for which no solution could be found are accounted for by a gap of 100 %. The best result w.r.t. the average gap is the combined setting, Both, which finishes the root node with an average gap of 70.27 %. The increase of the overall geometric mean solving time is 0.05 s or 2.1 %, from 2.36 s with RandI heuristics alone, to 2.41 for the setting Both.

Figure 4.7 shows the distribution of solution qualities, in terms of the primal gap, over all instances from 0 % to 100 % in steps of 20 %. The "none" bar gives the number of instances for which the corresponding setting did not provide any solution. For Shift-and-Propagate the solution quality on individual instances is often worse than for rounding and improvement heuristics.

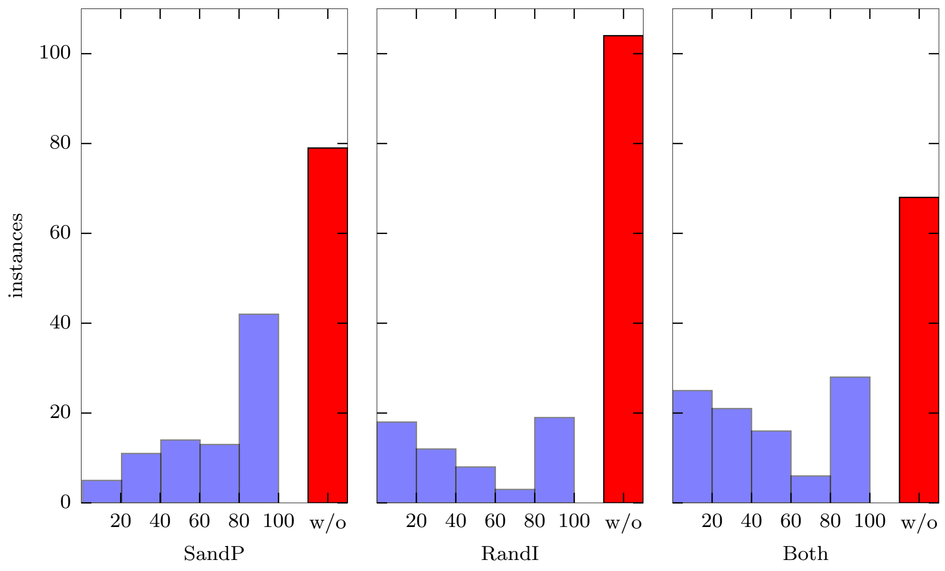


Figure 4.7.: Distribution of solution quality after root node processing

In particular, the latter produce more solutions with a gap between 0 % and 40 % than Shift-and-Propagate. The third setting combines both advantages: the higher number of solutions found with Shift-and-Propagate and the better gap obtained with rounding and improvement heuristics.

As a motivation for Shift-and-Propagate, we mentioned the XXL instances from Miplib 2010. Nine of these instances could be loaded into SCIP given the limitation of 48 GB RAM. Out of those, Shift-and-Propagate found a solution for five instances, the rounding and improvement heuristics only for three. This result is in favor of Shift-and-Propagate, but given the small size of the test set, it is of limited conclusiveness.

4.6. Conclusion

In this chapter, we gave an overview of rounding and improvement heuristics for mixed integer linear programming and we introduced Shift-and-Propagate, a new pre-root primal heuristic for MIP. It alternately shifts variables to promising fixing values in order to make linear constraints feasible and propagates these fixings to get tighter domains for choosing subsequent fixing values. Shift-and-Propagate differs from most other MIP heuristics (exceptions include Rapid Learning, see Chapter 9) in that it is does not require a feasible LP solution as a starting point, and that it is specifically designed as a quick start heuristic inside a global solver. The main contributions are the presentation of a quick and reliable procedure to select a shifting variable and an analysis of different variable orders, which cover the two main degrees of freedom in the heuristic.

We conducted two experiments, which revealed that Shift-and-Propagate alone finds solutions on more instances than three rounding heuristics and two combinatorial improvement heuristics together. Combining the existing rounding and improvement heuristics with Shift-and-Propagate increases the number of instances on which a feasible solution is found by 60 %. Furthermore, the average primal gap at the end of root node processing could be reduced by 14 %, while the running time increased only by 2 %.

We conclude that Shift-and-Propagate complements existing LP-based root node heuristics nicely. It is now one of the default heuristics applied in SCIP. As we recently found out [Ach, Gur14], Cplex and Gurobi feature similar heuristics which have been independently developed, but not been published yet.

5. Feasibility Pump(s)

The Feasibility Pump (FP) is probably the best known primal heuristic for mixed integer programming. The original work by Fischetti, Glover, and Lodi [FGL05] has been succeeded by more than a dozen follow-up publications [BFL07, AB07, FS09, BCLM09, BG12, HLM10, BC11, BEE^{+}11, BEET12, DFLL10, DFLL12, DSLR14, DSLR13] which improve the performance of the FP and extend it to other problem classes. One aspect of this chapter is to provide an overview of the Feasibility Pump literature.

The fundamental idea of all FP algorithms is to construct two sequences of points which hopefully converge to a feasible solution of a given optimization problem. One sequence consists of points which are feasible for a continuous relaxation (e.g., the LP relaxation of a MIP), but possibly integer infeasible. The other sequence consists of points which are integral, but might violate some of the imposed constraints. The next point of one sequence is always generated by minimizing the distance to the last point of the other sequence, using different distance measures in either cases (e.g., the ℓ_1 or the ℓ_2 norm).

Among all the cited work there has been only one approach, by D'Ambrosio et al. [DFLL10, DFLL12], to extend the FP algorithm to the class of problems which are foremost considered in this thesis: nonconvex MINLPs. Within this chapter, we propose new ideas for a Feasibility Pump algorithm for nonconvex MINLP. These result from joint work with Pietro Belotti from Clemson University (now Fair Isaac Europe Ltd); a joint publication is in preparation.

The outline of this chapter is as follows. In Section 5.1, we give a short introduction to the FP idea. Section 5.2 describes the FP paradigm for MIP (for which FP was originally introduced) and presents various FP variants from the literature. Section 5.3 extends the FP description to MINLP and relates to previous work on this subject. Section 5.4 presents, in detail, our new contributions towards an FP for nonconvex MINLP, while Section 5.5 shows the results of computational experiments[17] carried out on a set of difficult MINLP instances. Section 5.6 provides concluding remarks.

5.1. Introduction

Feasibility Pump algorithms follow the idea of decomposing a mathematical programming problem into two parts: integer feasibility and constraint feasi-

[17]Unlike other chapters of this thesis, the implementation of the algorithm has not been performed in SCIP. Instead, it is based on COUENNE [BLL^{+}09], an open-source solver for MINLP.

bility. At least for MIP and convex MINLP, both are “easy” to achieve: the former by rounding, the latter by solving an LP or a convex NLP, respectively, which can be done in polynomial time. Consequently, two sequences of points $\{\tilde{x}^k\}_{k=1}^K$ and $\{\bar{x}^k\}_{k=1}^K$, for $K \in \mathbb{Z}_{\geqslant 0}$, are generated such that $\tilde{x}$ contains integral points that may violate constraints and $\bar{x}$ contains points that are feasible for a continuous relaxation to the original problem but might not be integral. These two sequences are related to each other in that the points of $\tilde{x}^k$ are obtained through rounding of points in $\bar{x}^k$ and these, in turn, are obtained via projections of points in $\tilde{x}^k$.

One focus of this chapter is the application of a Feasibility Pump inside a global solver. This has an impact on the design choices carried out in developing the heuristic, in particular balancing efficiency versus completeness. Nowadays, the state-of-the-art commercial and non-commercial MIP solvers CBC [Cbc, FLH05], FICO XPRESS [FIC], GLPK [Glp], GUROBI [Gur], IBM ILOG CPLEX [IBM], SCIP [Ach09], and SYMPHONY [Sym] all feature Feasibility Pump implementations in their portfolios of primal heuristics. These typically differ from the published algorithms in that they are designed with a focus on a low average running time, sacrificing success on some instances. The enumeration phase of the Feasibility Pump presented in [BFL07] is a typical example of a component that is crucial for its impressive success rate as a standalone algorithm, but it will most likely not be applied when the Feasibility Pump is used inside a global solver, see, e.g., [Ber06].

This chapter features three novel contributions aimed at a more flexible use of an FP within a nonconvex MINLP solver:

(i) using a hierarchy of rounding procedures, ranked by efficiency (typically converse to the quality of the provided points) for finding integral points; which routine to use is decided and automatically adapted at runtime,

(ii) an improved distance function for the rounding step, taking into account second-order information of the continuous NLP relaxation, and

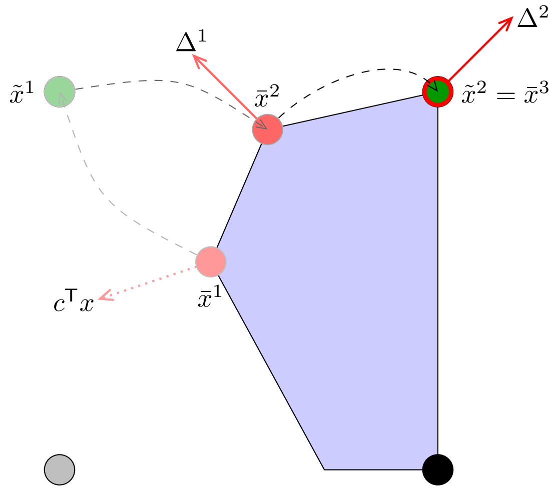
(iii) the separation of linearization cuts that approximate the *convex envelope* of the nonconvex feasible set of (2.1), as opposed to employing gradient cuts only for the convex part of the problem.

The computational experiments of this chapter are the only ones for which we did not use SCIP, but COUENNE. This is mainly due to a technical reason: to adapt the idea of an Objective Feasibility Pump [AB07] to MINLP, we wanted to deal with nonlinear objective functions directly rather than putting them into a separate constraint. This is possible in COUENNE, but not in SCIP.

5.2. Feasibility pumps for MIP

The *Feasibility Pump (FP)* algorithm was originally introduced by Fischetti, Glover, and Lodi in 2005 [FGL05] for mixed binary programs, i.e., for the

Figure 5.1.: Feasibility Pump for MIP, sequences of constraint-feasible points (red) and integer feasible points (green), original objective (dotted red) and distance functions (solid red).



special case of MIPs in which $l_j = 0$ and $u_j = 1$ for all $j \in \mathcal{I}$. The principal idea is as follows. The LP relaxation of a MIP is solved. The LP optimum $\bar{x}$ is then rounded to the closest integral point:

$$\tilde{x} = \begin{cases} [\bar{x}_j] & \text{if } j \in \mathcal{I} \\ \bar{x}_j & \text{if } j \notin \mathcal{I}, \end{cases} \tag{5.1}$$

where $[\cdot]$ represents scalar rounding to the nearest integer. This part of the FP algorithm is called the *rounding step*. If $\tilde{x}$ is not feasible for the linear constraints, the objective function of the LP is changed to an ℓ_1 distance function:

$$\Delta(x, \tilde{x}) := \sum_{j \in \mathcal{I}} |x_j - \tilde{x}_j| = \sum_{j \in \mathcal{I}:\, \tilde{x}_j = 0} x_j + \sum_{j \in \mathcal{I}:\, \tilde{x}_j = 1} (1 - x_j) \tag{5.2}$$

and a new $\bar{x}$ is obtained by minimizing $\Delta(x, \tilde{x})$ over the LP relaxation of the MIP. The process is iterated until $\tilde{x} = \bar{x}$ which implies feasibility (w.r.t. the MIP). The operation of obtaining a new $\bar{x}$ from $\tilde{x}$ is known as the *projection step*, as it consists of projecting $\tilde{x}$ to the feasible set of a continuous relaxation of the MIP along the direction $\Delta(x, \tilde{x})$. Two iterations of the algorithm are illustrated for a simple example in Figure 5.1.

The algorithm thus produces two sequences $\{\bar{x}^k\}_{k=1}^K$ and $\{\tilde{x}^k\}_{k=1}^K$ for a finite K, which is either the iteration at which a feasible solution for (2.1) is found or an iteration limit set to guarantee termination. All points of the sequence $\bar{x}^k$, with k denoting the iteration count of the FP, are feasible for the LP relaxation, all points $\tilde{x}^k$ are integral, i.e., $\tilde{x}_j^k \in \mathbb{Z}$ for all $j \in \mathcal{I}$. Thus, $\tilde{x}^k = \bar{x}^k$ implies integrality and constraint-feasibility, which means that the corresponding point is feasible for the MIP.

The main obstacle for the original Feasibility Pump algorithm (and most of its successors) is *cycling*: after some iterations, it may hold that $\tilde{x}^k = \tilde{x}^{k'}$ with $1 \leqslant k' < k$. In this case, the procedure would enter a loop, re-visiting the sequence $\tilde{x}^{k'} \dots \tilde{x}^{k-1}$ (and $\bar{x}^{k'} \dots \bar{x}^{k-1}$) over and over again. Since the central idea of FP is to bring the sequences closely together, the risk of cycling is "naturally encoded" in the procedure and occurs very frequently in computational experiments. In the original Feasibility Pump, this issue is handled via a simple random perturbation: some of the variables in $\tilde{x}^k$ are flipped to the other bound before continuing the procedure. It is a crucial component of many FP extensions that cycling is addressed directly, made more unlikely, or avoided completely.

Fischetti, Glover, and Lodi demonstrated in [FGL05] that the Feasibility Pump is very effective in finding feasible solutions, but these often are of minor quality – not surprising when regarding the fact that the original objective is only considered in the very first iteration. The authors suggest to use subsequent runs of the Feasibility Pump to get better solutions. After each successful run, a primal bound constraint $c^{\mathsf{T}}x \leqslant \alpha c^{\mathsf{T}}\bar{x} + (1-\alpha)c^{\mathsf{T}}\tilde{x}$ is added to the MIP, with $\alpha \in (0,1)$, $\bar{x}$ being an optimal solution of the original LP relaxation, and $\tilde{x}$ being the solution from the previous FP run.

Bertacco et al. [BFL07] introduced an FP variant for mixed integer programs with general integer variables. Therefore, the authors use an auxiliary variable d_j and two auxiliary constraints for each general integer variable x_j to represent the two linear pieces of the absolute values $|x_j - \tilde{x}_j|$. Then, the objective function for the projection step is a modified version of function (5.2):

$$\Delta(x,\tilde{x}) := \sum_{j\in\mathcal{I}:\,\tilde{x}_j=l_j} (x_j - l_j) + \sum_{j:\,\tilde{x}_j=u_j} (u_j - x_j) + \sum_{j\in\mathcal{I}:\,l_j<\tilde{x}_j<u_j} d_j$$

with $d_j \geqslant x_j - \tilde{x}_j$ and $d_j \geqslant \tilde{x}_j - x_j$ for all $j \in \mathcal{I}$ with $l_j < \tilde{x}_j < u_j$.

Further, the authors suggest to split the FP procedure into different stages. In stage I, the ℓ_1-norm objective function is defined only on the binary variables. The auxiliary variables and constraints are only used in stage II, which also considers distances on general integers. Finally, stage III is an enumeration phase consisting of a truncated MIP search. For this, the objective function of the original MIP is replaced by the distance w.r.t. the point x from the previous stages that was closest to the LP relaxation. This is in the spirit of the Proximity Search algorithm by Fischetti and Monaci [FM13], see also Chapter 6. Hanafi et al. [HLM10] introduced an FP variant in which they apply a MIP search each time the Feasibility Pump is about to cycle.

The principal idea of the Feasibility Pump is to decompose the original problems into two problems, retaining linear and integrality constraints. Many of the extensions described below aim at solving each of the decomposed problems with an eye on the constraints of the other problem, in an attempt to accelerate convergence.

Improving the rounding step

Fischetti and Salvagnin [FS09] observed that rounding a variable can be interpreted as a temporary fixing. They suggest to propagate the minimum and maximum activities of a linear constraint using these fixings as local bounds. The propagation is done via the well-known bound strengthening techniques described in [BMW75], see also Chapter 4.3. This procedure avoids rounding other variables to values that can be proven to not lead to a feasible solution (given the previous fixings). Using this propagation engine, the rounded solution might be further away from the last LP optimum than for the original FP, but it will be closer to the feasible region. The so-called Feasibility Pump 2.0 needs fewer iterations and produces slightly better solutions.

Baena and Castro [BC11] and Boland et al. [BEE$^+$11] recently introduced variants of the Feasibility Pump that use integral reference points $\tilde{x}$ which are closer to the interior of the LP polyhedron. Therefore, both publications suggest to connect the LP optimum $\bar{x}$ with the analytic center [Son86] of the LP and search for integer points that are roundings of points on that line segment. The analytic center x^{ac} of a bounded polyhedron given in equality form $(Ax = b, x \geqslant 0)$ is defined as

$$x^{\mathrm{ac}} = \operatorname{argmin}\{-\sum_{j \in \mathcal{N}} \ln x_j \mid x > 0, Ax = b\}.$$

Baena and Castro sample points on the line segment $\bar{x} - x^{\mathrm{ac}}$, which are then rounded and tested for feasibility. If none of the points is feasible, a new integral reference point $\tilde{x}$ is chosen that minimizes the ℓ_∞ distance of the rounded point to the line segment point it has been rounded from. Boland et al. [BEE$^+$11] extend this procedure by several innovative ideas. First, they observe that the set of all integral points which are roundings of some point of the line segment can be computed very efficiently. This improves the sampling step. The main overhead of the procedure in [BC11] lies in the computation of the analytic center in order to get a direction pointing from the LP optimum towards the interior of the polyhedron. Boland et al. suggest to use a conic combination of the normal vectors of all constraints violated by $\tilde{x}$ as a cheap heuristic approximation for a ray pointing towards the center. To the other extreme, Naoum-Sawaya [NS13] recently proposed a version of the Feasibility Pump with analytic centers that additionally applies a *recursive central rounding* procedure, which iteratively fixes some of the integer variables and recomputes the analytic center.

Improving the projection step

The main direction of modification for the projection part of the FP was the use of different objective functions for the LP. Achterberg and Berthold [AB07] showed a simple trick to overcome a main weakness of the FP: despite success on many instances, the produced solutions are often of poor quality.

The authors suggest to replace function (5.2) by a convex combination of (5.2) and the original objective $c^\mathsf{T}x$:

$$\Delta_\alpha(x, \tilde{x}) := (1-\alpha)\Delta(x, \tilde{x}) + \alpha \frac{\sqrt{|\mathcal{I}|}}{\|c\|} c^\mathsf{T} x$$

with $\alpha \in [0,1]$. Here, $\|\cdot\|$ is the Euclidean norm of a vector. The convex combination factor α, and hence the influence of $c^\mathsf{T}x$, is reduced in every iteration. As a nice side effect, this often enables the algorithm, called Objective Feasibility Pump, to avoid cycling since the objective function $\Delta_\alpha(x, \tilde{x})$ depends on the iteration count and will be different even when the same point $\tilde{x}$ is visited more than once.

Eckstein and Nediak [EN07] interpreted the Feasibility Pump as an implementation of a Frank-Wolfe algorithm [FW56], taking the ℓ_1 distance as a non-smooth concave merit function:

$$\sum_{j \in \mathcal{I}} \min\{x_j, 1 - x_j\}.$$

Based on this, de Santis et al. [DSLR14, DSLR13] interpreted and suggested the use of other concave penalty functions for non-integrality. Therefore, they weighted the different terms of the distance function with coefficients that depend on the fractionality of the corresponding variable in the last LP solution.

Boland et al. [BEET12] use a similar penalty system, but also introduce the idea to perform several rounds of cutting plane generation to prevent the FP from cycling. This leads to fewer restarts and better and more solutions being found, but at the price of a significant increase in the complexity of the LPs (which are amended with cutting planes) being solved repeatedly, and hence in the total running time.

Besides enhancements of the FP for MIP itself, a natural direction of investigation is the extension of the FP idea to other applications and to broader problem classes. In [Ach10, Ach11], Achterberg and Gu suggest to use a Feasibility Pump like algorithm, called PumpReduce, to generate alternative LP optima which can be used for improved cut generation and filtering. In [BS13], Berthold and Salvagnin use a similar algorithm as a basis for a branching scheme that is based on a set of relaxation optima, see also Chapter 10.

Of particular interest to the research community has been the extension of the Feasibility Pump to MINLP, which shall be the topic of the next section.

5.3. Feasibility pumps for MINLP

When considering MINLPs instead of MIPs, the obvious question is: how to adapt the two FP steps for the new problem class, i.e., what kind of relaxation should be solved in the projection step, and is there a different way to perform the rounding step? *Nonconvex* MINLPs represent an extra burden: even the continuous relaxation might be disconnected and is, in general, nonconvex; hence optimization over it is $\mathcal{NP}$-hard.

The first two MINLP versions of the Feasibility Pump were presented by Bonami et al. [BCLM09] and Bonami and Gonçalves [BG12]. Both teams of authors considered *convex* MINLPs and implemented their ideas in Bonmin [BBC$^+$08].

The paper [BG12] is probably the closest to the original FP. It keeps the rounding step as in [FGL05] and replaces solving an LP in the projection phase by solving a convex NLP, using again the distance function (5.2) as an objective. The perturbation scheme is less aggressive than the one of [FGL05], flipping only a single variable. Recently, Sharma [Sha13] presented an integration of the Objective Feasibility Pump idea by Achterberg and Berthold and the algorithm of Bonami and Gonçalves: a scaled sum of the distance function and the original objective is used as an objective for the convex NLP.

In [BCLM09], the authors suggest using an ℓ_2 norm for the projection step. Further, their implementation of the rounding step differs significantly from all previous FP variants. Instead of performing an instant rounding to the nearest integer, they solve an MIP relaxation which is based on an outer approximation [DG86] of the underlying MINLP:

$$\tilde{x} = \operatorname{argmin}\{\Delta(x,\bar{x}) \mid g(\bar{x})+J_g(\bar{x})(x-\bar{x}) \leqslant 0, x \in [l,u], x_j \in \mathbb{Z}\,\forall j \in \mathcal{I}\} \quad (5.3)$$

where $J_g(\bar{x})$ denotes the Jacobian of the constraint functions (summarized to a single function $g\colon \mathbb{R}^n \mapsto \mathbb{R}^m$) evaluated at the NLP optimum $\bar{x}$. Solving a MIP relaxation, despite of being an $\mathcal{NP}$-hard problem itself, is often computationally much cheaper than solving the original MINLP, see also Chapter 8. Interestingly, the two norms have switched roles in this FP version: Where in [FGL05] and [BG12] the ℓ_1 norm was used for the projection step, and the ℓ_2 norm was used for rounding, the opposite holds for [BCLM09]. An illustration of the algorithm is given in Figure 5.2. Note that for this FP, both steps use a distance function Δ. We denote the Manhattan distance used for the rounding step by Δ_1 and the Euclidean distance of the projection step by Δ_2, using superscripts for the iteration count.

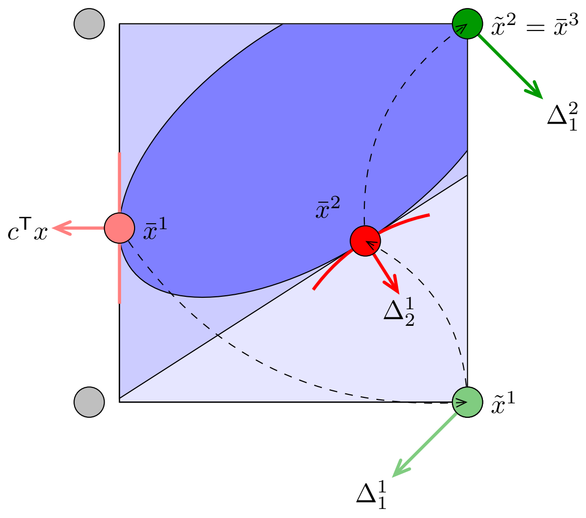
Solving (5.3) instead of performing a simple rounding $\tilde{x} = [\bar{x}]$ of course gives rise to better integral points (since feasibility is explicitly addressed in the rounding step), and has an important effect w.r.t. the main weakness of Feasibility Pump algorithms: cycling. For convex MINLPs, it is always possible to derive a cut

$$(\bar{x} - \tilde{x})^{\mathsf{T}}(x - \bar{x}) \geq 0$$

and add it to the MIP (5.3). By this, cycling is avoided. However, in the computational results presented in [BCLM09], the FP did not cycle even without these cuts.

Similar to the interpretation of the linear Feasibility Pump as a Frank-Wolfe algorithm [DSLR14, DSLR13], D'Ambrosio et al. [DFLL12] gave a

Figure 5.2.: Feasibility Pump for MINLP: The truncated blue ellipsis shows the feasible region of the NLP relaxation, the light blue trapezoid depicts the MIP relaxation for the second iteration, the very light blue rectangle (which include the trapezoid) is the MIP relaxation after the first iteration. The sequence of NLP-optima is shown as red points, the sequence of MIP-optima as green points. Each red and green point also indicates for which function it is optimal: the original objective ($c^\mathsf{T}x$, light red), a ℓ_2 distance function (Δ_2^1, dark red and bent), and ℓ_1 distance functions (Δ_1^1 and Δ_1^2, green).



classification of a nonlinear FP as a successive projection method. The particular difficulty addressed in [DFLL10, DFLL12] is that of handling the nonconvex NLP relaxation if adapting the algorithm of [BCLM09] to the nonconvex case. The authors suggest using a stochastic multistart approach, feeding the NLP solver with different randomly generated starting points, and solving the NLP to local optimality as if it was a convex problem. In the event that this does not lead to a feasible solution, a final NLP is solved in which the integer variables are fixed and the original objective is re-installed on the continuous variables, similar to Step 5 of the Algorithm in Figure 8.3.

Further, D'Ambrosio et al. considered solving a convex MINLP or a convex MIQP instead of an MIP in the rounding step, but gave computational evidence that this is not beneficial. To avoid cycling, their algorithm provides the MIP solver with a tabu list of previously used solutions. Linear constraints for the MIP problem are only generated from convex MINLP constraints. Finally, they showed that using an ℓ_∞ norm instead of ℓ_1 as a MIP objective is competitive.

5.4. New tricks for a nonconvex objective feasibility pump

This section discusses, in detail, new ideas for a Feasibility Pump for nonconvex MINLP. In accordance with the FP paradigm, the suggested algorithm generates two sequences: one fulfilling the integrality constraints, the other satisfying the continuous, nonlinear constraints. The integral points are generated by finding a feasible solution of a MIP in form (2.3), similar to [BCLM09] and [DFLL10, DFLL12]. This MIP is obtained from the MINLP (2.1) through a procedure that uses reformulation techniques and generates a system of linear inequalities which provide a linear relaxation to the continuous relaxation of (2.1), see [McC76, TS04, SP97] and Chapter 2.1.

More specifically, the general version of a nonlinear Feasibility Pump receives as input an optimum (w.r.t. the original objective) of an NLP relaxation $\bar{x}^1$ and generates the two sequences as follows, for $k = 1, \dots, K$:

$$\begin{aligned} \tilde{x}^k = \operatorname{argmin} \quad & \Delta_{\text{int}}(x, \bar{x}^k) \\ \text{s.t.} \quad & Ax \leqslant b \\ & l_j \leqslant x_j \leqslant u_j \quad \text{for } j \in \mathcal{N} \\ & x_j \in \mathbb{Z} \qquad\quad\;\; \text{for } j \in \mathcal{I}; \end{aligned} \tag{5.4}$$

$$\begin{aligned} \bar{x}^{k+1} = \operatorname{argmin} \quad & \Delta_{\text{nl}}(x, \tilde{x}^k) \\ \text{s.t.} \quad & g_i(x) \leqslant 0 \qquad\quad \text{for } i \in \mathcal{M} \\ & l_j \leqslant x_j \leqslant u_j \quad \text{for } j \in \mathcal{N}. \end{aligned} \tag{5.5}$$

The two objective functions Δ_{int} and Δ_{nl} are typically the ℓ_1 norm for Δ_{int} and the ℓ_2 norm for Δ_{nl}. The rounding and the projection step are carried out by solving problems (5.4) and (5.5), respectively. Variable bounds are enforced in both problems (5.4) and (5.5) since they can be dealt with by both MIP solvers and NLP solvers. Though finding a global optimum would be desirable for problem (5.5), implementations of nonlinear Feasibility Pumps typically resort to searching for a local optimum in the case of nonconvex problems.

We introduce several ideas for enhancing the performance of this basic algorithm. The main contributions that distinguish of Feasibility Pump implementation from existing approaches are the following:

- ▷ At every iteration, we choose one from a set of procedures to search for a feasible solution of (5.4). All these procedures are implemented within SCIP. They range from very fast but inaccurate methods (such as rounding in 5.1) to slow, but very effective methods (such as a truncated search of a full-fledged MIP solver). This list contains algorithms that vary broadly in terms of running time and solution quality, and our algorithm features an automatic adaptation that aims at balancing running time and solution quality at every iteration.[18]

[18]This is related to the "goldilock" mechanism used in the RINS heuristic [BRG09a] to

- ▷ We use a parameterized version of both $\Delta_{\text{int}}(x, \bar{x}^k)$ and $\Delta_{\text{nl}}(x, \tilde{x}^k)$, which also takes into account second-order information from the nonlinear constraints and the objective function of problem (2.1) – note that the latter feature already appears in the objective Feasibility Pump for MIP [AB07].

- ▷ Our FP method takes into account not only convex, but also nonconvex constraints directly to refine the linear relaxation, using linearization constraints generated by COUENNE. This is in major contrast with known FP methods for MINLP [BCLM09, DFLL12], where nonconvex constraints are either ignored or approached with a local NLP solver to obtain a local optimum.

In the remainder of this section, we will discuss these three ideas and a few other implementational tricks in detail.

Hierarchy of rounding procedures

In the preceding publications on Feasibility Pump heuristics, several ideas have been proposed to generate good integer points during the rounding step. The original work [FGL05] and one of the nonlinear FPs [BG12] suggest plain rounding in order to get a point which is closest w.r.t. the ℓ_2 norm. The Feasibility Pump 2.0 [FS09] uses an iterated "round-and-propagate" procedure in order to get a point which is close w.r.t. the ℓ_2 norm but "more feasible" for the relaxation. The nonlinear Feasibility Pumps suggested by [BCLM09, DFLL12] solve an MIP to get the closest integer point w.r.t. the ℓ_1 norm that fulfills *all* constraints of the linear relaxation.

The observed shift from "stay close to the previous point" to "stay close, but also fulfill the relaxation" leads us to the idea of trying several rounding procedures which address these goals in various ways. In order to obtain a point $\tilde{x}$ that satisfies the integrality constraints, we select a procedure from the following list:

(i) Solve the MIP relaxation with a node limit and an emphasis on good solutions.

(ii) Solve the MIP relaxation with a node limit (smaller than in (i)), disabling time-consuming cutting plane separation, branching and presolving strategies.

(iii) Solve the root node of the MIP relaxation of method (i), then apply the RENS heuristic, see [Ber14] and Chapter 7.

automatically adapt variables fixing thresholds for the creation of a subproblem or the step size adaptation in VNS heuristics [HM01, HMU06]. The named procedures automatically change the size (and thereby the complexity) of an auxiliary problem that should be considered, whereas we change the complexity of the method used to heuristically exploit a given auxiliary problem.

(iv) Solve the root node of the MIP relaxation, then apply an Objective Feasibility Pump 2.0 [FS09] for MIPs.

(v) Apply a round-and-propagate algorithm of [FS09]: selectively round a fractional variable to the nearest integer and then apply a domain propagation to restrict the feasible set, until either a feasible solution is found or all solutions are eliminated. Compare also Chapter 8.4.

(vi) Choose an integral point from a solution pool (e.g. from suboptimal solutions of applying procedure i), see below.

(vii) Apply a random perturbation to $\bar{x}^{k-1}$ and obtain an integer $\tilde{x}^k$.

Note that the integral points generated by procedures (v) to (vii) might be infeasible for the MIP problem (5.4) of the current iteration.

Having this variety of options to produce integral points, the question remains *which* to apply *when*. At the first iteration of the FP, we employ procedure (ii). If the current procedure successfully provides us with a "new" (not yet visited) integer point for three iterations in a row, we proceed with the next (cheaper and less aggressive) method of the above list. If, at any iteration, the current procedure either does not terminate within the given limits or produces a point that was already visited, we proceed with the previous (more expensive and more powerful) method of the above list. Note that the list of procedures from (i) to (vii) has decreasing complexity and, in general, declines in solution quality. At subsequent iterations of the FP we use the procedure that was successful previously, but switch down to a cheaper routine when there were three successful iterations in a row.

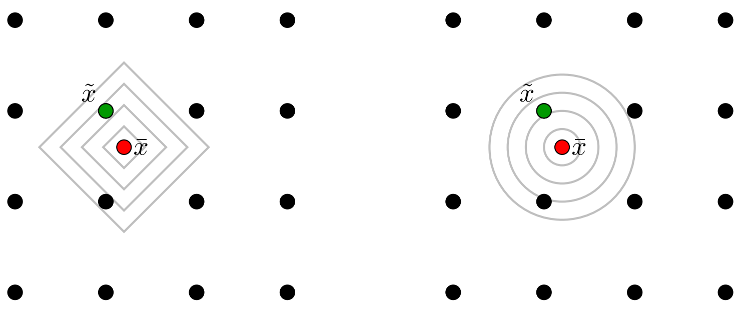
In principle, the list could be prepended at the head to include methods that capture constraint feasibility even better (and are most likely more time consuming), for instance a convex (nonlinear) relaxation of the nonconvex MINLP. However, computational results presented in [DFLL12] indicate that this results in a significant computational overhead with little impact.

If procedures (i), (ii), or (iii) are used for the rounding step, these may produce more than one MIP-feasible solution. The suboptimal points might be used for later iterations, and are therefore stored in a solution pool. This approach is motivated by two observations: first, in (i) we solve similar MIPs over and over again, mainly using a different objective (plus some new cuts). Each known feasible point from a previous call may be used as a starting solution in subsequent calls. Procedures (iii) and (iv) will also benefit from a given upper bound as they consider a restricted search space. Second, considering a point which was initially a candidate, but was then discarded, carries more information about the problem than one generated through random perturbation. Thus, the previously collected points are used as a second last option in (vi). We rank the points in the pool by the value of the distance function at the current iteration.

Improved distance functions

The ℓ_1 and ℓ_2 norms were used in nearly all FP variants that we described in Sections 5.2 and 5.3. The Feasibility Pumps for MIP use the ℓ_1 for the projection step; the nonlinear FPs by Bonami et al. [BCLM09] and D'Ambrosio et al. [DFLL12] use ℓ_2 for the projection and ℓ_1 for rounding. Either way, exclusively using these norms as objective functions for auxiliary optimization problems ignores the fact that a "close" solution is not necessarily a "good" one: the original objective function is completely neglected and there is no information involved by how much the constraints are violated.

To overcome this issue, we use the norm of a vector obtained from a linear transformation applied to $x - \bar{x}^k$. As a motivating example, consider first an unconstrained integer nonlinear optimization problem $\min\{f(x) \colon x \in \mathbb{Z}^n\}$, where $f \in \mathcal{C}^2(\mathbb{R}^n, \mathbb{R})$. Assume that $\bar{x}$ is a local optimum of the continuous relaxation $\min\{f(x) \colon x \in \mathbb{R}^n\}$. Level curves of the ℓ_1 and the ℓ_2 distance functions w.r.t. $\bar{x}$ are given in Figure 5.3. In either norm, $\tilde{x}$ is the closest integer point.



(a) $\Delta(x, \bar{x}) = ||x - \bar{x}||_1$ (b) $\Delta(x, \bar{x}) = ||x - \bar{x}||_2$

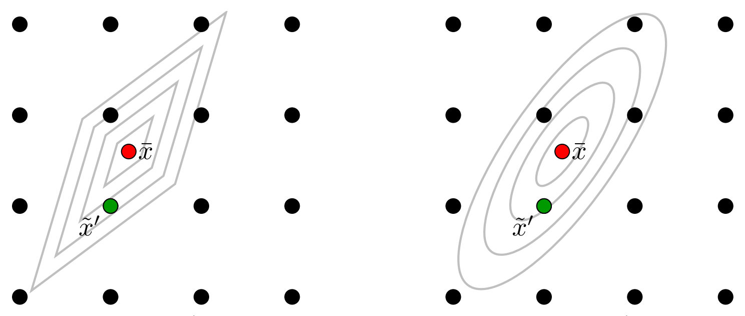
Figure 5.3.: Level curves (gray) of different norm functions Δ $(x,\bar{x})$ for problem (5.4). The closest integer point to $\bar{x}$ is $\tilde{x}$. In 5.3(a) and 5.3(b), the norm $|| \cdot ||_p$ is used for $p = 1$ and $p = 2$ respectively.

Now consider the second degree Taylor series approximation of f at $\bar{x}$:

$$f(x) \approx f(\bar{x}) + \nabla f|_{\bar{x}}^{\mathsf{T}}(x - \bar{x}) + \frac{1}{2}(x - \bar{x})^{\mathsf{T}} H (x - \bar{x}) \tag{5.6}$$

which is convex and quadratic. We want to use (5.6) for constructing an improved distance function which uses information about f. Since the problem is unconstrained, the gradient of f is null, i.e., $\nabla f|_{\bar{x}} = 0$, and its Hessian is positive semidefinite, i.e., $H = \nabla^2 f|_{\bar{x}} \succeq 0$. Thus, minimizing (5.6) is equivalent to minimizing $(x - \bar{x})^{\mathsf{T}} H (x - \bar{x})$ given that the first two terms can be ignored (the first one is constant and the second one has a null gradient).

As shown in Figure 5.4(b), the level curves of this new function are ellipsoids whose axes and axis lengths are defined by the eigenvectors and eigenvalues of H. Note that this relation is inverse proportional, i.e., the



(a) $\Delta(x,\bar{x}) = ||H^{\frac{1}{2}}(x-\bar{x})||_1$ (b) $\Delta(x,\bar{x}) = ||H^{\frac{1}{2}}(x-\bar{x})||_2$

Figure 5.4.: Level curves (grey) of different norm functions using second order information associated with the objective function of the original problem. The distance function in 5.4(b) and its piecewise linear approximation in 5.4(a) both lead to $\tilde{x}'$ as best integral point in the vicinity of $\bar{x}$.

larger the eigenvalue, the steeper the ascent, the *shorter* the corresponding axis. A convex, piecewise linear approximation of this objective function is $||H^{\frac{1}{2}}(x-\bar{x})||_1$. Its level curves are represented in Figure 5.4(a). This is a "distance" function that incorporates information about the original objective function. Both functions that are displayed in Figure 5.4 find $\tilde{x}'$ as best (w.r.t. the Hessian) integral point near $\bar{x}$. Hence, $\tilde{x}'$ might be a better candidate for the next iteration of a Feasibility Pump algorithm. Note that, in general, neither $||H^{\frac{1}{2}}(x-\bar{x})||_1$ nor $||H^{\frac{1}{2}}(x-\bar{x})||_2$ yield a minimum in $\mathcal{R}(\bar{x})$ (see Definition 2.9), i.e., the hypercube $[\lfloor\bar{x}\rfloor, \lceil\bar{x}\rceil]$ containing $\bar{x}$. The advantage of using the Hessian H, which incorporates second-order information about the current optimal solution of the nonlinear problem, is that a minimizer of $||H^{\frac{1}{2}}(x-\bar{x})||_1$, while possibly far from $\bar{x}$ in terms of the ℓ_1 norm, corresponds to an integer point whose objective function is close to that of $\bar{x}$, hence providing a "good" solution from the objective function standpoint.

Let us now generalize this to the constrained version. If we considered an MINLP with a nonlinear objective function, the Hessian of the objective function would, in general, be indefinite at the optimum $\bar{x}$ of the relaxation (that is, there might be active constraints). In case of a linear objective, as in Definition 2.1, the Hessian is constant zero. Therefore, we use the Hessian of the Lagrangian function of the NLP relaxation of the original MINLP.

Note that we explicitly assume that the objective function of the MINLP might be nonlinear in this chapter. Unlike SCIP, Couenne handles nonlinear objectives directly. This was the main reason to perform the implementation and the computational experiments in this chapter with Couenne.

Definition 5.1 (Hesse-distance)**.** *Let $\tilde{H} \in \mathbb{R}^n \times \mathbb{R}^n$ be the Hessian of the*

Lagrangian and a reference point $\bar{x} \in [l, u]$ be given. We call

$$\tilde{\Delta}_{int}(x, \bar{x}) = ||\tilde{H}^{\frac{1}{2}}(x - \bar{x})||_1$$

the Hesse-distance *of x to $\bar{x}$.*

We suggest to incorporate the Hesse-distance into the objective functions of the auxiliary MIPs that are solved in the rounding step of the nonlinear Feasibility Pump. In the spirit of the Objective Feasibility Pump, we came up with the following combinations:

$$\Delta_{\text{int}}(x, \bar{x}) \quad = \quad \alpha_{\text{dist}}||x - \bar{x}^k||_1 + \alpha_{\text{H}}\tilde{\Delta}_{\text{int}}(x, \bar{x}) + \alpha_{\text{orig}}c^{\mathsf{T}}x;$$

if the original objective of the MINLP is a linear function $c^{\mathsf{T}}x$, and

$$\Delta_{\text{int}}(x, \bar{x}) \quad = \quad \alpha_{\text{dist}}||x - \bar{x}^k||_1 + \alpha_{\text{H}}\tilde{\Delta}_{\text{int}}(x, \bar{x}) + \alpha_{\text{orig}}z;$$

otherwise, with z being an auxiliary variable that is constrained by a linear approximation of the original (nonlinear) objective. Typically, one would increase α_{dist} in every iteration, making it converge to one and fade out the other two, thereby shifting the focus from solution quality towards pure feasibility. Note that for any value of these parameters the objective function is piecewise linear and convex. Note further that one can easily extend the definition of the Hesse-distance to the Euclidean case and incorporate it into the objective function for the projection phase. Preliminary experiments revealed, however, that this is not beneficial.

Separation of linearization cuts

Techniques to generate a linear relaxation of an MINLP can be used in an incremental fashion as a *separation* procedure: given a solution $\tilde{x}$ to a MIP relaxation, that is not feasible for the MINLP itself, find a linear cut $a^{\mathsf{T}}x \leq d$ that is fulfilled by all solutions of the MINLP, but $a^{\mathsf{T}}\tilde{x} > d$ (or show that no such inequality exists). LP-based branch-and-bound solvers for MINLP, such as COUENNE or SCIP, typically solve such separation problems to improve local dual bounds at each node. For details on branch-and-cut for MINLP, see, e.g., [TS02, Vig12]. In marked difference to previous nonlinear Feasibility Pumps, our implementation also separates linear over- and underestimators for nonconvex functions and not exclusively gradient cuts for convex parts of the problem.

A typical problem occurring in iterative heuristics such as the FP is cycling. Some versions prevent cycling by adding no-good cuts as outlined in the previous section. Our FP variant attempts to avoid cycling in two ways. First, linear inequalities for nonconvex MINLPs are added to eliminate infeasible integer points. For convex constraints, gradient cuts are added. For nonconvex constraints, the situation is more involved. COUENNE, similar to SCIP, uses the standard approach of reformulating nonconvex constraints via an expression tree whose nodes are variables and elementary nonlinear functions. For these nonlinear functions, underestimators are used to produce valid linear relaxations. For instance, nonconvex bilinear terms can be addressed via McCormick underestimators [McC76]. This might still allow for permanently separating the MIP solution from the feasible region.

However, when MIP (5.4) terminates with an optimal solution that is infeasible for the (nonconvex) MINLP, but inside the convex hull of its feasible set, no linear cut can be added to separate the solution from its feasible set. This leads to the second way of avoiding cycling: We forbid particular assignments to the integer variables by adding bound disjunction constraints [Ach07b] to the MIP (5.4); this is effectively an implementation of a tabu list.

Postprocessing

If a MINLP feasible solution $\tilde{x}$ is found, the values for the continuous variables are only optimal for the distance objective used at the last iteration. To check whether there are better solutions, we run a simple local search improvement heuristic. We obtain a restriction the original MINLP by fixing all integer variables to the values of the Feasibility Pump solution $\tilde{x}$ and solve it with a convex NLP solver such as Ipopt, compare Step 5 of the Algorithm in Figure 8.3.

5.5. Computational experiments

We implemented the Feasibility Pump within Couenne 0.4.7 [BLL+09], where the latter is based on cbc 2.8.9 [Cbc]. Within our implementation, the auxiliary MIP problems, see Equation (5.4), are solved by SCIP 3.0.2 [Ach09], linked against SoPlex 1.7.2 [Wun96]. The auxiliary NLPs, see Equation (5.5), are solved by Ipopt 3.11.7 [WB06, Ipo]. The results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20 GHz with 12 MB cache and 48 GB main memory, running an openSuse 12.3 with a gcc 4.7.2 compiler. Turboboost was disabled. In all experiments, we ran only one job per node to reduce fluctuations in the measured running times that might be caused by interference between jobs that share resources, in particular the memory bus.

As a test set, we used 218 instances from MinlpLib [BDM03]. We compared the following six different settings of the nonlinear Feasibility Pump:

- **default** uses a Manhattan distance function, without contributions of the Hessian of the Lagrangian or the original objective; this setting does not add convexification cuts for non-convex parts of the problem; the auxiliary MIP is always solved by running SCIP with a stall node limit of 1000 and aggressive heuristic settings
- **cuts** uses cuts for nonconvex parts of the problem (in addition to standard MIP cuts and gradient cuts); otherwise the same as default

▷ **hierarchy** uses different algorithms to solve the MIP in different iterations of the Feasibility Pump, see Section 5.4; otherwise the same as default

▷ **hessian** constructs the objective of the auxiliary MIP as a combination of the Manhattan distance and the Hessian of the Lagrangian, see Section 5.4; we chose $\alpha_{\text{dist}} = 1 - 0.95^k$ and $\alpha_{\text{H}} = 0.95^k$ at the k-th iteration; otherwise the same as default

▷ **objective** constructs the objective of the auxiliary MIP as a combination of the Manhattan distance, the Hessian of the Lagrangian and a linear approximation of the original objective; we chose $\alpha_{\text{dist}} = 1 - 0.95^k$, $\alpha_{\text{H}} = 0.95^k$, and $\alpha_{\text{orig}} = 0.9^k$ at the k-th iteration; otherwise the same as default

▷ **simple** applies rounding to the nearest integer instead of solving an auxiliary MIP in the rounding phase, compare [BG12]

In Table B.3 in the appendix, each double-column gives the time needed for processing the root node and the objective value of the incumbent solution after root node processing for one of the settings mentioned above. A dash in the "solution" column indicates that no feasible solution was found with this setting. Table 5.1 shows a summary that aggregates the results of Table B.3.

For each of the six settings, we give three performance indicators: feas, the number of instances (out of 218) for which this setting found a feasible solution, bet : wor, the number of instances for which this setting found a better/worse solution (in terms of the objective function value) as compared to the default setting, and time, the running time in shifted geometric mean, including Couenne's presolving and reformulation algorithms being applied. We used a shift of 100 for the number of branch-and-bound nodes and a shift of of 10 seconds for the running time.

Table 5.1.: performance of different Feasibility Pump versions for a single call at the root node (aggregated results)

setting	feas	bet : wor	time
default	150	–	14.9
cuts	155	24 : 48	13.6
hierarchy	157	23 : 20	14.0
hessian	154	25 : 16	22.8
objective	138	45 : 30	23.9
simple	97	17 : 78	12.1

First of all, we observe that each of the five non-default settings outperforms the default in at least one of the three measures of performance. The hierarchy setting is the only one to outperform the default in all three measures.

We further see that the cuts and the hessian setting both lead to slightly more solutions being found. This could be expected since both attempt to better incorporate the structure of the nonlinear feasibility region into the auxiliary MIP formulation. While using the Hessian leads to better solutions being found, applying cutting plane separation for nonconvex constraints deteriorates the quality of the found solution more often than it improves it. Computing the Hessian itself might take considerable time, both settings that make use of this feature show an increase in running time by more than 50 %.

Similar to results for linear Feasibility Pumps, using the original objective often leads to better solutions being produced by the Feasibility Pump, but at the same time reduces the number of solutions being found. Note that the bet : wor statistic includes those cases for which only one of the settings found a solution. Finally, the "simple" setting, which does not use an auxiliary MIP at all, is only slightly faster than the default setting, but much worse in terms of found solutions and solution quality.

For our implementation, we observe another behavior typical for Feasibility Pumps: although they are very successful in finding feasible solutions (about 75 % of the instances for the hierarchy setting), these solutions are often of a mediocre quality. In only 25 of the test instances, our Feasibility Pump implementation found a solution with less than 10 % gap to optimality (using the hierarchy setting). Then again, there were 26 instances with an optimality gap of more than 100 %. In geometric mean, the primal gap of the found solutions was 34 %.

5.6. Conclusion

In this chapter, we gave a literature overview on Feasibility Pump heuristics for MIP and MINLP. Among the fourteen publications that we reviewed, there has been only one approach, by D'Ambrosio et al. [DFLL10, DFLL12], to extend the Feasibility Pump algorithm to nonconvex MINLP. We presented and evaluated three novel ideas for solving nonconvex MINLPs with a Feasibility Pump: the generation of valid cutting planes for nonconvex nonlinearities, using a hierarchy of MIP solving procedures, and applying an objective function for the auxiliary MIPs that incorporates second-order information. For the latter, we introduced the so-called Hesse-distance.

In our computational experiments, the dynamic use of various MIP solving strategies showed the favorable behavior to produce more solutions and better solutions in a shorter average running time. A convex combination of the Hesse-distance function and the Manhattan distance likewise improved the number of found solutions and their quality, but at the cost of an increased running time.

6. Large Neighborhood Search: From MIP to MINLP

Large neighborhood search (LNS) heuristics are an important component of modern MIP solvers, see, e.g., [Ber06, FL10, Lod13]. To define the neighborhood, the feasible region of the MIP is restricted by additional constraints: most often variable fixings or some very restrictive cardinality constraint. For MIP, the LP relaxation plus the integrality constraints suffice to define the search space. However, for more general problem classes, the LP relaxation alone may not contain enough information about the original problem to find feasible solutions via LNS, e.g., if the problem is nonlinear or not all constraints are present in the current relaxation (which may be the case in branch-and-cut applications).

In this chapter, we discuss a generic way to extend LNS heuristics that have been developed for MIP to MINLP. This chapter is based on work together with Stefan Heinz, Marc E. Pfetsch, and Stefan Vigerske [BHPV11] in which we discussed the extension of LNS heuristics to constraint integer programming – more specifically to MIQCP, nonlinear pseudo-Boolean optimization, and resource-constrained project scheduling problems. In this context, the author of this thesis implemented extended versions of the following LNS improvement heuristics: Local Branching [FL03], RINS [DRP04], Crossover [Ber06, Rot07], and DINS [Gho07]. Our results indicate that the suggested generalization considerably improves the success rate of these primal heuristics.

This chapter gives an overview of LNS heuristics for MIP that have been presented in the literature in recent years, describes a generalization to MINLP and presents new computational experiments, extending the results of [BHPV11] via a study on the overall impact of the presented heuristics on the global search. The chapter is organized as follows. After a brief introduction in Section 6.1, we review LNS heuristics for MIP in Section 6.2 and for more general classes of mathematical programs in Section 6.3. Section 6.4 discusses two variants of generalizing LNS heuristics from MIP to MINLP and describes the implementation in SCIP. Finally, we present computational experiments in Sections 6.5 and 6.6 as well as our conclusions in Section 6.7.

6.1. Introduction

Large neighborhood search [Sha98] is a variant of the local search paradigm that has been widely used in constraint programming, operations research, and combinatorial optimization [AL97, PSF04, PG99]. LNS has proved to be an extremely successful metaheuristic for a wide range of applications in recent years, see, for example, Pisinger and Røpke [PR10]. The main idea is to restrict the search for "good" solutions to a neighborhood centered at a particular starting point – typically the incumbent or another feasible solution. The starting point is often synonymously referred to as the *reference solution*. The hope is that such a restricted search space makes the subproblem much easier to solve, while still providing solutions of high quality. Of course, these restricted subproblems do not have to be to solved to optimality; we are mainly searching for an improving solution. Obviously, any LNS heuristic will then benefit from a good performance of (other) primal heuristics on its subproblem.

In mixed integer linear programming, LNS has recently been realized in a series of primal heuristics [FL03, DRP04, Ber06, Rot07, Gho07, Ber14, FM13]. It is one form of the *MIPping* [FLS10] idea that suggests to take crucial decisions within a MIP solver by solving auxiliary MIPs. The so-called Local Branching [FL03] heuristic has been further extended to constraint programs [KLMP07, KLMP12] and mixed integer nonlinear programs [NBL08]. A RINS heuristic for convex MINLP has been suggested in parallel to our work [BHPV11] by Bonami and Gonçalves [BG12].

6.2. Large neighborhood search for MIP

Many MIP primal heuristics published in recent years [FL03, DRP04, Ber06, Rot07, Gho07, Ber14, FM13] are based on large neighborhood search. These heuristics investigate a neighborhood of a single starting point (or a small set of starting points) such as the incumbent solution or the optimal solution of the LP relaxation. They create a sub-MIP of the original MIP, typically by fixing some variables to values that are taken from the given points. For problems with binary variables only, another possibility is to add linear constraints, which restrict the number of variables that are different from the given point. By the use of auxiliary variables, this can be extended to problems with general integer variables while maintaining linearity. Moreover, the objective function might be modified to direct the search into a region with many feasible solutions. Finally, an objective cutoff constraint is added to enforce that a solution found by the sub-MINLP will be better then the current incumbent. This is particularly worthwhile if the incumbent itself was feasible for the sub-MINLP (in order to avoid the same solution being found twice).

Obviously, a good definition of the neighborhood is the crucial point:

- ▷ the neighborhood should contain high quality solutions,
- ▷ these solutions should be easy to find, and
- ▷ the neighborhood should be easy to process.

Naturally, these three goals are conflicting in practice. In the remainder of this section, we will give a brief introduction to large neighborhood search heuristics for MIP that have been proposed in the literature of the last ten years.

Assume that we are in the process of solving a MIP by a branch-and-bound algorithm. In the following, let $\tilde{x}$ be the incumbent solution and $\bar{x}$ be the optimum of the LP relaxation at the current node.

Local Branching

Local Branching [FL03] measures the distance to the starting point in Manhattan norm on the integer variables and only considers solutions which are inside a k-neighborhood of the reference solution, where k is typically between 10 and 20. This is done by adding a linear constraint that sums up the distance to the starting point, typically the incumbent, over all variables:

$$\sum_{j\in\mathcal{I}} |x_j - \tilde{x}_j| \leqslant k \tag{6.1}$$

Inequality 6.1 is sometimes referred to as a "Local Branching constraint". Using the Manhattan norm has the advantage that it is easy to linearize, as opposed to, e.g., the Euclidean norm. Note that for the binary part of the problem, it holds that

$$\sum_{j\in\mathcal{B}} |x_j - \tilde{x}_j| = \sum_{j\in\mathcal{B}} (x_j - 2x_j\tilde{x}_j + \tilde{x}_j) = \sum_{j\in\mathcal{B}} (x_j^2 - 2x_j\tilde{x}_j + \tilde{x}_j^2) = \|x_j - \tilde{x}_j\|^2_{\mathcal{B}},$$

with $\|\cdot\|_{\mathcal{B}}$ denoting the Euclidean norm restricted to the index set $\mathcal{B}$ of binary variables. For general integer variables, auxiliary variables might have to be used for a linearized model of the absolute values.

Originally, Fischetti and Lodi introduced Local Branching as a branching strategy that is tailored towards improving the primal bound quickly.[19] They suggested to interleave the standard branching rules of a MIP solver with branching on general disjunctions $\sum_{j\in\mathcal{I}} |x_j - \tilde{x}_j| \leqslant k \vee \sum_{j\in\mathcal{I}} |x_j - \tilde{x}_j| \geqslant k+1$ as soon as a feasible solution $\tilde{x}$ is available. The idea to use the search space of the smaller branch as a neighborhood for an improvement heuristic came up at the same time. In Cplex 9.0, released in 2002, there was the possibility to use a Local Branching heuristic via a hidden parameter [Ach], with Cplex version 9.1, this became available as a public parameter. This makes Local Branching an excellent example for the connection between branching heuristics and primal heuristics: it can be employed either way.

[19] In [FL03], this is referred to as improving the "heuristic behavior" of a MIP solver.

In a follow-up publication, Fischetti and Lodi suggested to use Local Branching as a repair heuristic, by starting it with an infeasible reference solution [FL08].

Relaxation Induced Neighborhood Search (RINS)

The *Relaxation Induced Neighborhood Search* (rins) [DRP04] is an improvement heuristic based on the fact that an improving MIP solution fulfills three conditions: it is integral, feasible for the linear constraints, and it has an objective function value smaller than the incumbent. rins uses two starting points: The incumbent MIP solution which fulfills the first two requirements and the optimum of the LP relaxation which fulfills the latter two. At each branch-and-bound node, it holds that $c^T\bar{x} < c^T\tilde{x}$ (otherwise the node could be pruned); this implies $\bar{x} \neq \tilde{x}$. The rationale of rins is that those values which coincide in both solutions give rise to a partial solution of good objective value. Therefore, rins defines the neighborhood by fixing all integer variables which take the same value in both solutions.

One can argue that fixing a binary variable principally reduces the search space of a MIP by a factor of two: for a BP the number of potential solutions (i.e. 0-1 points) reduces from $2^{\mathcal{B}}$ to $2^{\mathcal{B}-1}$. Thus, fixing only a few variables can make a big difference in the computational effort which is needed to solve a problem. This consideration is amplified by the fact that MIPs arising from industrial applications often have a hierarchical structure. Fixing a "top-level" variable will often trigger a series of propagations; in an extreme case, the sub-MIP might even decompose. Nevertheless, fixing too few variables might result in a sub-MIP which is not significantly easier than the original.

In SCIP, this case is handled by imposing a fixed threshold for the minimum percentage of variables to be fixed [Ber06]. Gomes et al. [GSS13] suggest to employ this threshold after presolving of the subproblem and use a binary search to find a suitable limit on the problem size. Bixby, Rothberg and Gu [BRG09a] proposed a method that dynamically adjusts the fixing threshold after each call of rins. If rins reaches a node or time limit without finding an improving solution, the threshold is increased; if it proves the reference solution to be optimal for the sub-MIP, it is decreased. They refer to this procedure as the *goldilocks* method. The reasoning behind this is that in the first case the subproblem was to hard – the feasibility status could not be decided within the given limits –, hence at the next call of rins a smaller problem should be considered. In the latter case, the subproblem was to restrictive, thus, we may allow for larger subproblems. If an improving solution is found, the threshold remains unchanged.

Relaxation Enforced Neighborhood Search (RENS)

In contrast to rins, the *Relaxation Enforced Neighborhood Search* (rens), see Chapter 7 and [Ber14] does not require an incumbent solution and thus can be used as a start heuristic. The idea of RENS is to search the space of feasible roundings of a given fractional reference solution. Therefore, RENS fixes all integer variables that take an integral value in the optimal solution of the LP relaxation. For the remaining integer variables, the bounds get tightened to the two nearest integral values. We discuss the details of RENS in Chapter 7.

Crossover

Crossover [Ber06, Rot07] is an improvement heuristic that is inspired by genetic algorithms and requires more than one feasible solution. Crossover seeks to fix variables to values that coincide in (at least) two solutions, motivated by the following observations:

- ▷ Often, variables need to be fixed to a certain value in order to obtain feasibility. Presolving should eliminate easy incarnations of this case. However, presolving does often not detect fixings that are only implied by a conjunction of constraints.
- ▷ Variables might need to be fixed to certain values in order to obtain a good objective function value. This often holds for variables that correspond to strategic decisions.

Thus, the reasoning behind Crossover is that often good feasible solutions have a lot of variable values in common. There might, however, be other reasons for solution values to coincide:

- ▷ Solutions have been found in the same part of the tree, i.e., they share local fixings, or by the same primal heuristic.
- ▷ Chance.

Crossover aims at fixing variables that coincide for the first two reasons while coincidences that result from the latter two reasons might better be ignored. One strategy to avoid variables to be fixed by chance is using not only two "parent" solutions (as is typical for genetic algorithms), but a larger set. Further, the Crossover implementation in SCIP requires that its reference solutions have not been found all by the same heuristic at the same node. Otherwise, Crossover would most likely optimize over the same, or at least a similar, search space as the other heuristic did. In genetic algorithms (as the one suggested by Rothberg to polish MIP solutions [Rot07]), a *mutation* step is implemented as the natural adversary to Crossover. It is used to diversify the set of reference solutions. In a MIP solver with many start heuristics and a solution pool storage [Ach07b] such as SCIP or Cplex, this is typically not required. Interestingly, the solution pool has always[20] been part of SCIP,

[20]Precisely, it has been introduced three months after the project start, before the first running version.

long before Crossover had been developed. CPLEX only introduced it with version 11.0, whereas solution polishing (including a mutation step) has been introduced in version 10.0.

Distance Induced Neighborhood Search (DINS)

Ghosh suggests the *Distance Induced Neighborhood Search* (DINS) [Gho07]. DINS combines the ideas of RENS, RINS, Crossover and Local Branching. It defines the neighborhood by introducing a distance function between the incumbent solution and the optimum of the LP relaxation. When applied during a branch-and-bound search, it further takes into account how variables change their values at different nodes of the tree. The hinge of the DINS algorithm is to search for MIP solutions that are closer to the relaxation optimum $\bar{x}$ than the current incumbent $\tilde{x}$ is, i.e. for which

$$\sum_{j\in\mathcal{I}} |x_j - \bar{x}_j| \leqslant \sum_{j\in\mathcal{I}} |\bar{x}_j - \tilde{x}_j| \tag{6.2}$$

holds, with $\bar{x}$ being the optimal solution of a (local) relaxation and $\tilde{x}$ the current incumbent. To achieve this, DINS fixes general integer variables for which the relaxation and the incumbent differ by less than 0.5 to the value of the incumbent (similar to RINS), tightens the bounds of the remaining general integer variables (as RENS), fixes binary variables for which all previous incumbents took the same value (similar to Crossover) and adds a Local Branching constraint on the remaining binary variables.

Proximity Search

Fischetti and Monaci published a preprint on *Proximity Search* [FM13] in 2013. Proximity Search combines ideas from Local Branching and the Feasibility Pump (see Chapter 5). It is designed at the borderline between global and heuristic algorithms. It does not solve a subproblem, but it modifies the original problem in that it replaces the objective function by a so-called proximity function. Obviously, this does not restrict the set of feasible solution vectors. The proximity objective function is basically identical to the Local Branching constraint (6.1):

$$\min \sum_{j\in\mathcal{I}} |x_j - \tilde{x}_j|.$$

In their paper, Fischetti and Monaci evaluate the performance of Proximity Search for MBPs and convex 0-1 MIQCPs, using the primal integral discussed in Chapter 3 as performance measure. Furthermore, the authors suggest to exploit a combined strategy of Local Branching and Proximity Search; computational results for this, however, are not presented.

6.3. LNS for other problem classes

There have been a few publications on transfering MIP large neighborhood search heuristics to other problem classes. We first give a brief literature overview, before discussing two different possibilities to implement large neighborhood search heuristics in SCIP.

Bonami and Gonçalves describe an extension of the RINS heuristic to convex MINLPs [BG12]. They use an optimum of the NLP relaxation as a second reference solution besides the incumbent. Interestingly, for their implementation in Bonmin, the resulting sub-MINLPs are solved by the Quesada and Grossmann algorithm [QG92], whereas for the original MINLP an NLP-based branch-and-bound algorithm is used. The former is much closer to the way SCIP solves MINLPs (and sub-MINLPs). The authors employed a minimum fixing ratio of 10 % and stop the sub-MINLP after the third improving solution, as it has also been suggested in [Ber06].

In [NBL08], Nannicini, Belotti, and Liberti introduce a Local Branching heuristic for nonconvex MINLPs. It solves a Local Branching MIP which is derived from a linear relaxation of the original MINLP, the integrality constraints, and a Local Branching constraint (6.1). Subsequently, an NLP local search is performed by fixing the integer variables to the values from the Local Branching MIP's incumbent – which is not necessarily feasible for the original MINLP– and solving the resulting continuous problem.

Kiziltan et al. [KLMP07, KLMP12] show how to integrate Local Branching as a search strategy into a constraint programming framework. They present a reduced-cost based propagation rule specifically for the Local Branching constraint and test the efficiency of their approach on instances of the asymmetric traveling salesman problem with time windows. In [PM12], Parisini and Milano introduce *Sliced Neighborhood Search*, which can be understood as a combination of Local Branching and Mutation: $|\mathcal{I}| - k$ integer variables are randomly chosen and fixed to the value of a feasible reference solution; on the remaining k variables, a Local Branching constraint is imposed. Due to this combination, typically larger values for k can be chosen as compared to Local Branching. The authors tested their approach as standalone procedure and integrated it into a CP framework, showing promising results for instances of the Asymmetric Traveling Salesman Problem with Time Windows.

The author is not aware of publications on DINS or Crossover heuristics for MINLP. However, Lübbecke and Puchert [LP12, Puc11] suggest a variant of Crossover that is designed to be used inside the branch-and-price solver GCG [Gam10, GL10]. They define a Crossover neighborhood by fixing variables that coincide in several integer, but infeasible, extreme points which are obtained by solving pricing problems. Proximity Search has been tested for convex MIQCPs in the original publication of Fischetti and Monaci [FM13].

6.4. Two variants of a generalization

In the previous section, we have seen that there are two natural ways of extending LNS heuristics for MIP towards MINLP. First, in [NBL08], Nannicini, Belotti, and Liberti used a linear relaxation plus the integrality constraint to create an auxiliary MIP which is typically not a subproblem of the original MINLP. Second, in [BG12], Bonami and Gonçalves used a proper sub-MINLP of the original MINLP. The first approach comes with the advantage that the new problem might be significantly easier (see Chapter 8 in which we consider sub-MIPs of nonconvex MINLPs). The second comes with the important characteristic that each feasible solution of the new MINLP will be a solution of the original MINLP.

Originally, the LNS heuristics implemented in SCIP created the LNS subproblem by taking a copy of the LP relaxation, adding integrality constraints, and fixing variables (or adding a local branching constraint). This corresponds to the approach of [NBL08]. For the release 2.0 in 2010, we implemented copying procedures in SCIP and redesigned all LNS heuristics to copy the original problem (rather than the relaxation) into a new SCIP instance and to apply the neighborhood search on this copy, in the spirit of [BG12].

For MIP, both approaches give rise to the same subproblem, since a MIP is fully specified through its LP relaxation and the integrality constraints. This, however, is not true for more general problem classes, such as MINLP. In this case, a feasible solution of an LNS problem which is constructed from the LP relaxation plus integrality constraints is no longer guaranteed to be feasible for the original problem. As a consequence, the chances that a LNS heuristic finds a feasible solution reduce when working only on the relaxation of the problem.[21] Summarizing, the first approach, using auxiliary MIPs, solves a potentially smaller instance[22] of an "easier" problem class; the second, using sub-MINLPs, solves a smaller problem of the same problem class; compare our discussion on the Undercover heuristic in Chapter 8.

Copying the whole problem, restricting the search space to a neighborhood of some point, and solving the resulting, hopefully easier, problem seems more promising, but may come with a significant computational overhead. A main part of reworking SCIP's LNS heuristics, also compared to [BHPV11], has been in introducing strategies to "fail fast", cf. Chapter 4.2, on subproblems that are hard to solve. To this end, all LNS heuristics that employ variable fixings now have a minimum fixing ratio, below which the subproblem solving will not be started. Further, all LNS heuristics now use a stall node limit, i.e., search in the subproblem will be aborted after a few (typically 50) nodes without finding a new incumbent solution. Additional improvements towards

[21]In [NBL08], the authors tackled this problem by performing an NLP postprocessing.

[22]The feasible region is restricted (by variables fixings or Local Branching constraints) and relaxed (by considering the LP rows instead of the original constraints) at the same time.

a "fast failure" have been made w.r.t. an issue that is much more likely to appear for MINLP than for MIP: long processing times of individual branch-and-bound nodes. This is mainly due to long cutting plane loops: the (weak) LP relaxation is strengthened further and further by gradient cuts, often generated by the same constraint(s) again and again. In particular when all integer variables are fixed, cutting plane generation is preferred over spatial branching. All of this makes much sense when the goal is to solve the problem to proven optimality, for a partial solve inside an LNS heuristic, however, it is better to leave such nodes unsolved (or with a weak dual bound) and continue search at other parts of the tree. This has been realized by employing additional limits on the number of LPs solved and the number of *enforcement* loops. See Achterberg [Ach09] for the concept of constraint enforcement in SCIP and Vigerske [Vig12] for its implementation in the case of nonlinear problems.

Note that the LNS heuristics from Section 6.2 do not make any particular assumptions on the problem class, except for being able to express linear constraints and objective functions for Local Branching and Proximity Search, respectively. Thus, using a copy of the original problem enables the easy application of the described LNS heuristics to any problem class for which the corresponding SCIP plugins implement the required copy methods.

Taking a different perspective, SCIP can be seen as an interface here: the formulation of a problem as a problem that SCIP can handle[23] allows the access to all methods described in Section 6.2. Unlike for most metaheuristic approaches, no additional problem specific adaption of the heuristic is necessary. An additional benefit comes from the fact that the copied instance allows for stronger constraint propagation than that of linear constraints and separation of problem specific cutting planes.

6.5. Computational experiments

The aim of our computational experiments is to investigate the potential of LNS heuristics, applied inside a branch-and-bound process, for MIQCPs. We performed two experiments: first, we set a node limit of one, i.e., we only solved the root node of the branch-and-bound tree, in order to evaluate the performance for single calls of the LNS heuristics. Second, we made a run without a node limit, but with a time limit of one hour, to evaluate the overall performance when solving instances to proven optimality.

All computations presented in the following used SCIP version 3.0.1.3. As the underlying linear programming solver we choose SoPlex 1.7.1, continuous nonlinear subproblems were solved by Ipopt 3.11, we further used CppAD version `trunk` (20120101.3) for computing function derivatives. The

[23]In [BHPV11], we considered pseudo-Boolean optimization and resource-constrained project scheduling as further examples for transfering MIP heuristics to other problem classes.

results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20GHz with 12 MB cache and 48 GB main memory, running an OPENSUSE 12.3 with a GCC 4.7.2 compiler. Turboboost was disabled. In all experiments, we ran only one job per node to avoid random noise in the measured running time that might be caused by delays if multiple processes share common resources, in particular the memory bus.

For our computational experiments we used a heterogeneous test set of nonconvex MIQCPs, introduced in [BGHV12], from which we removed instances that SCIP 3.0.1.3 reformulates as MIPs during presolve, since we are interested in measuring the performance of LNS heuristics when applied beyond MIP. This leads to a test set of 92 instances. For the second experiment, we additionally had to remove instance `itointqor`, for which SCIP aborts prematurely (in all tested settings).

We performed both experiments with three different parameter tunings, which alter SCIP's default as follows:

- ▷ the *no LNS* setting, that disables all LNS heuristics,
- ▷ the *aux. MIP* setting, which calls all before-mentioned LNS heuristics once at the root and frequently during search, using the LP relaxation as a basis for the neighborhood definition (i.e., the resulting auxiliary problems are MIPs),
- ▷ the *sub-MINLP* setting, which calls all before-mentioned LNS heuristics once at the root and frequently during search, using the MINLP itself as a basis for the neighborhood definition (i.e., the resulting subproblems are MINLPs).

Tables 6.1 and 6.3 give aggregated results for the root node and the overall runs, respectively. Table 6.2 shows information on the performance of individual heuristics. Tables B.4 and B.5 in the appendix depict the detailed results for single instances.

Table 6.1.: Overall performance of LNS heuristics at the root node (aggregated results)

setting	feas	better obj	time
no LNS	52	–	3.0
aux. MIP	56	33	4.2
sub-MINLP	63	49	5.6

In Table B.4 in the appendix, each double-column gives the time needed for processing the root node and the objective value of the incumbent solution after root node processing for one of the settings mentioned above. A dash in the primal bound column indicates that no feasible solution was found with this setting. Table 6.1 depicts the number of instances for which

Table 6.2.: individual performance of LNS heuristics at the root node for sub-MINLP setting

	RENS	Local	Proxy	RINS	DINS	Cross
calls	56	36	42	45	60	28
found	23	18	15	21	24	15
best	6	5	3	8	14	10

a feasible solution was found during root node processing, the number of instances for which the solution was better than for the no LNS setting, and the shifted geometric mean of the root node processing time taken over all 92 instances. Table 6.2 shows the individual performance of the six LNS heuristics described in Section 6.2 for the sub-MINLP setting (the results for the aux. MIP setting are similar). The rows calls, found, and best give the number of instances for which the heuristic has been called, the total number of solutions it found and for how many instances the heuristic gave rise to the best solution found during root node processing. The columns are sorted by the order in which the heuristics have been called within our experiment.

The conditions when to call a LNS heuristic differ in detail for each of them. For example, Proximity Search and Local Branching will only run when the problem has a certain amount of binary variables, Crossover needs at least three feasible solutions, RINS and DINS require a minimum percentage of variables to be fixed by their neighborhood definitions. Table 6.2 shows that the numbers of instances for which the heuristics got called are all different. The success rate in terms of found solutions is quite consistent, being between 35 % and 55 % of the calls. Further, for each of the heuristics there are at least three instances for which it produced the solution which is incumbent after the root. Of course, these numbers have to be taken with a grain of salt since they depend on the order in which the heuristics are called. The dependence might be in either way: on the one hand, a heuristic that is called later will only search regions of an improving objective, thereby exploiting information from its predecessors; on the other hand, a heuristic that is called earlier has the advantage of the "first pick" on high quality solutions. We performed some additional experiments with different orders, the results did not change tremendously.

Considering Table 6.1, we observe that using LNS heuristics leads to more feasible solutions and better objective function values, coming with an increase in running time, which could be expected. This goes even further when the LNS problems are constructed as MINLPs. There are 10 % more instances for which feasibility can be proven in the root, for more than half of the instances the primal bound improves, but at the same time the mean root node running time increases by around 80 %. During this additional 80 % of running time, on average four sub-MINLPs were solved (there were

362 calls of LNS heuristics in total over the whole test set). Note that there is one extreme outlier, `uflquad-40-80`, for which the running time increases from 1.8 seconds to 538.3 seconds.[24] If this one instance was excluded from the test set, the increase would “only” have been 62 %, which is still much. So, the question remains whether the improvements on the primal bound are worth the computational overhead in the long run. This shall be answered by our second experiment.

Table 6.3.: Performance of LNS heuristics during tree search (aggregated results)

setting	feas	obj	time	nodes	prim. int.
no LNS	78	–	24.9	1688	81434
aux. MIP	79	4:2	26.1	1382	76980
sub-MINLP	79	5:1	25.0	1227	75431

In Table B.5 in the appendix, we see double columns for the number of branch-and-bound nodes and the running time a certain setting of SCIP needs to prove optimality. In case that SCIP did not terminate within the time limit of one hour, “timeout” is written in the corresponding column. Table 6.3 gives a summary of these results, showing the number of instances for which feasibility could be proven, for which the corresponding setting terminated with a better or worse objective than the “no LNS” setting, the shifted geometric mean of the running times, the shifted geometric mean of the number of branch-and-bound nodes, and the average primal integral $P(\mathrm{t}_{\max})$, see Chapter 3 and [Ber13]. For this experiments, all three settings led to the same set of 57 instances to be solved to optimality. The shifted geometric means of the times and nodes are taken over these 57 instances.

In Figure 6.1, we see the evolution of the primal gap for the three different settings. The dotted green line corresponds to the average (taken over all 92 instances) primal gap function, when running SCIP with LNS heuristics based on a full copy of the original problem. The green shaded area corresponds to the average primal integral. Accordingly, the orange dashed line and the yellow shaded (plus the green shaded) area represent a run of SCIP with LNS heuristics based on the LP relaxation; the solid red line and the red shaded (plus the green and yellow shaded) area stand for running the solver without any LNS heuristic. We see a typical picture that resembles the observations made in Chapter 3: In the beginning, there is a steep descent of $p(t)$, which converges to a certain level for every setting. Although there are some intersections of the three functions in the beginning, for most of the time it holds that the average primal gap of “sub-MINLP” is strictly

[24]This was also the maximum running time observed for any instance, i.e. the time limit was never hit. The increase in running time for this particular instance came from long separation times in several nodes of the subproblem.

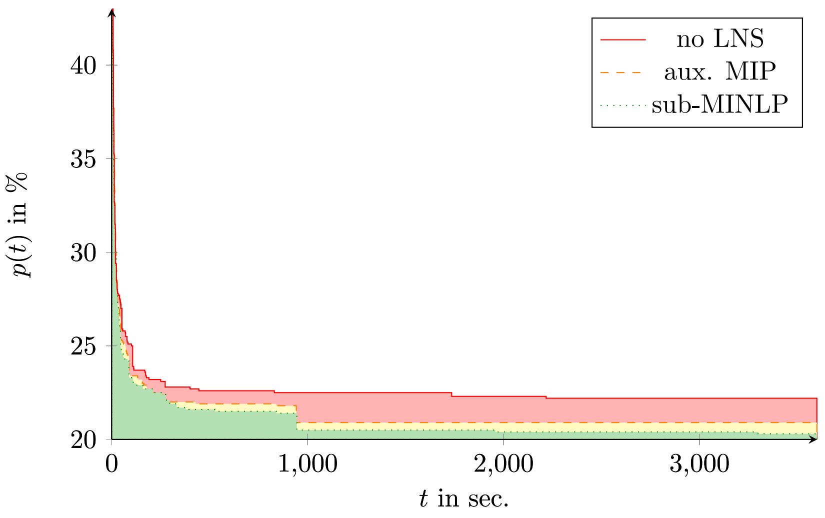


Figure 6.1.: Course of the primal gap when running SCIP with different settings for LNS heuristics

smaller than the one of “aux. MIP” which itself is strictly smaller than “no LNS”.

We observe that when using LNS heuristics in either setting, there is one more instance for which a feasible solution is found, and for those 35 instances which cannot be solved within one hour, LNS heuristics more often lead to an improvement in the primal bound at termination than to a deterioration. Concerning the running time for the solved instances, the setting that uses sub-MINLPs is performance-neutral whereas the LP-based version slightly decreases performance. Using LNS heuristics leads to a significant reduction in the number of branch-and-bound nodes that are needed to prove optimality, but this saving is complete equalized by the computational overhead which they produce. The primal integral reduces by 8 % when using LNS heuristics, which leads us to the conclusion that, although being performance-neutral on a first glimpse, they are a valuable component of SCIP. Our experiments further showed that using a sub-MINLP instead of an auxiliary MIP is better w.r.t. the overall running time as well as w.r.t. the quality of the primal bounds.

6.6. SCIP vs. LocalSolver

In 2011, the first version of LocalSolver was released. LocalSolver is a heuristic software for mixed integer optimization problems, it uses neighborhood search techniques to find feasible solutions. Since release 4.0, LocalSolver features full support of MIP, including handling of continuous variables. On the LocalSolver homepage, impressive results for 21 of the 164 challenge instances of Miplib 2010 can be found, accompanied by the statement that “For many instances, the conclusion is that none of the fastest MIP solver [sic!] is currently able to provide high-quality solutions quickly in short running times, as it is needed today in the practice of optimization and operations research.” [Loc]

We performed an additional experiment for which we ran SCIP 3.1 with SoPlex 2.0 as LP solver against LocalSolver 4.0 on the benchmark set of Miplib 2010.[25] We had to exclude five instances, namely `dfn-gwin-UUM`, `ex9`, `msc98-ip`, `n4-3`, and `rocII-4-11`, that could not be read after an automatic conversion of `*.lp` files into the `*.lsm` format. Note that unlike the rest of this chapter, the present section is purely concerned with mixed integer linear programs. The results were obtained on a 64bit Intel Core i7-3610QM 2.30 GHz with 256 KB cache and 4 GB main memory, running a Windows 8.1 operating system. Since we had to use Windows to run LocalSolver, the environment is different from the other experiments in this thesis.

We adopted the computational setup to match the setup used for the results shown by [Loc]. Firstly, we used a time limit of five minutes. Secondly, since LocalSolver is a pure heuristic solver that does not explicitly compute and strengthen dual bounds, we only focus on the primal side of the problem, namely on the development of the primal bound. Thirdly, since LocalSolver mainly uses neighborhood search techniques, we decided to use a setting for SCIP which has an emphasis on neighborhood search. We opted for the winning setting from the previous experiment, that applies different LNS heuristics throughout the branch-and-bound search of SCIP.

The outcome of this experiment is visualized in Figure 6.2, where we see the evolution of the primal gap for the two solvers. The dotted blue line corresponds to the average (taken over 82 instances) primal gap function of the SCIP run, the shaded blue area corresponds to the average primal integral, see Chapter 3. Accordingly, the red dashed line and the red shaded area (plus the blue shaded area) represent the LocalSolver performance.

We observe that w.r.t. the average quality of the incumbent solution, LocalSolver is superior to SCIP for the first ten seconds of this experiment; for the rest of the time, the incumbent solution of SCIP is better than the one given by LocalSolver. Furthermore, we see that the primal gap function of LocalSolver stalls after about one minute. Thus, five minutes is a reasonable choice for a time limit. SCIP, however, keeps improving w.r.t. solution quality, although like-wise its primal gap function “bends” after about one minute. In total, the average primal gap of SCIP on the Miplib 2010 benchmark set during the first five minutes of the solution process is 36.4 %, the average primal gap of LocalSolver is 65.8 %.

Looking at the results after the time limit of five minutes, there are six instances for which LocalSolver finds a feasible solution, but SCIP does not. For 23 instances, SCIP finds a feasible solution, but LocalSolver does not. There are 17 instances for which both solvers fail to find any feasible

[25]We are indebted to Gerwin Gamrath for his support with this particular experiment.

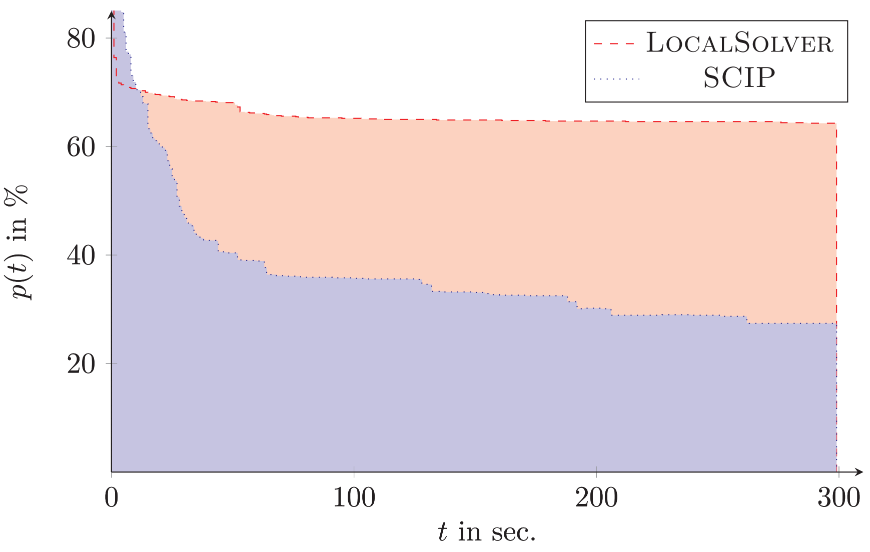


Figure 6.2.: Course of the primal gap when running SCIP with aggressive LNS heuristics and LocalSolver, time limit: five minutes

solution. This leaves 36 instances for which both solvers succeed in finding solutions. Out of these, both solvers end up with the same primal bound in six of the cases. In four of these cases, the final solution (after five minutes) reported by LocalSolver is better than the one by SCIP, in 26 cases, SCIP is superior. Moreover, for 20 out of 87 instances, SCIP did actually not need the full five minutes, but could prove optimality and therefore stop before the time limit, whereas LocalSolver would always exhaust the full time limit.

We conclude that on a test set of general MIPs, a MIP focused solver seems to be preferable over a pure neighborhood search in most of the cases, even when only looking at the quality of the primal bound within a quite restricted time interval. As always, averages do not tell the story for single instances and for certain problem classes, e.g., the `opm` instances from Miplib, a local search might perform significantly better than state-of-the-art MIP codes.

6.7. Conclusion

In this chapter, we provide an overview on large neighborhood search heuristics for MIP and existing approaches to extend them to MINLP. As a further contribution, we described a generic and straightforward way of generalizing large neighborhood search heuristics from mixed integer *linear* programming to mixed integer *nonlinear* programming, using MIQCP as a showcase. We implemented and tested MINLP versions of six LNS heuristics that are known from the literature. To the best of our knowledge, for Crossover and dins this is the first time that nonlinear variants have been described and tested.

Both showed promising results, DINS being the heuristic providing the most solutions in our root node experiment, Crossover the one with the highest success rate given by “solutions per call”.

Our computational results did not only show that the generalized LNS heuristics increased the quality of the best feasible solution after root node computation, but also that they improve the behavior of the solver for the global search. The improvement merely consisted of a faster convergence towards the optimal solution from the primal side, while not affecting the time needed to prove optimality. This aligns well with our overall computational experiments, see Chapter 11. The results further indicate that using an actual copy of the problem to solve a sub-MINLP is superior to using an auxiliary MIP based on the LP relaxation. This confirms earlier experiments with generic implementations of LNS heuristics for MIQCP, pseudo-Boolean optimization and resource-constrained project scheduling that we presented in [BHPV11]. In addition, we showed that on the Miplib 2010 benchmark set, SCIP outperforms a commercial solver that is based on neighborhood search w.r.t. the quality of primal solutions.

7. RENS: the optimal rounding

At the heart of many MIP improvement heuristics, such as Local Branching [FL03], RINS [DRP04], and DINS [Gho07], lies large neighborhood search (LNS), the paradigm of solving a small sub-MIP which promises to contain good solutions. Recently, these LNS improvement heuristics have been extended to the more general case of MINLP, cf. Chapter 6.

In this chapter, we introduce the *relaxation enforced neighborhood search* (RENS), a large neighborhood search algorithm for MINLP. It constructs a sub-MINLP of a given MINLP based on an optimal solution of a linear or nonlinear relaxation. RENS is designed to compute the optimal – w.r.t. the original objective function – rounding of a relaxation solution. Unlike the primal heuristics mentioned above, RENS is not an improvement algorithm: it does not require a feasible solution as a reference point. A slightly modified version of this chapter has been published in Mathematical Programming Computation [Ber14].

This chapter is organized as follows. Section 7.1 motivates our approach and Section 7.2 introduces the generic scheme of the RENS algorithm. In Section 7.3, we discuss the algorithmic design and describe implementation details, in particular for the application of RENS as a subsidiary method inside a global solver. The setup for the computational experiments is presented in Section 7.4. Section 7.5 provides detailed computational results and Section 7.6 contains our conclusions.

7.1. Introduction

Many LNS heuristics, diving and of course all rounding heuristics are based on the idea of fixing some of the variables that take an integral value in a relaxation solution. Therefore, the question of whether a given solution of a relaxation is roundable, i.e., all fractional variables can be shifted to integral values without losing feasibility for the constraint functions, is particularly important for the likelihood of many primal heuristics to succeed.

The RENS algorithm which is introduced in this chapter can be applied in two different ways: as a standalone algorithm to compute an optimal rounding of the given start solution and as a primal heuristic inside a global MINLP solver.

Following the former, we use RENS to analyze the roundability of instances from different classes of mathematical programs, and to demonstrate the computational impact of using different relaxations, namely an LP and an NLP relaxation. We evaluate the performance of several rounding heuristics, see Chapter 4, by a comparison against these results. Finally, we investigate the effectiveness of RENS applied as a start heuristic at the root node of a branch-and-cut solver. For these experiments, we use general, publicly available MIP, MIQCP and MINLP test sets obtained from the Miplib 3.0 [BCMS98], the Miplib 2003 [AKM06], the Miplib 2010 [KAA+11], the MinlpLib [BDM03] and the MIQCP test set compiled in [BGHV12].

7.2. A scheme for an LNS rounding heuristic

Given a mixed integer program, the paradigm of fixing a subset of the variables in order to obtain subproblems that are easier to solve has proven successful in many MIP improvement heuristics such as Crossover [Ber06, Rot07], RINS [DRP04], and DINS [Gho07]. These strategies can be directly extended to MINLP, see Chapter 6 and [BHPV11].

Before we formulate the RENS algorithm, let us formalize the notion of an optimal rounding. Recall Definition 2.9:

Definition (rounding). *Let $\bar{x} \in [l,u]$. The set*

$$\mathcal{R}(\bar{x}) \coloneqq \{x \in \mathbb{R}^n \mid x_j \in \{\lfloor \bar{x}_j \rfloor, \lceil \bar{x}_j \rceil\} \text{ for all } j \in \mathcal{I}, l_j \leqslant x_j \leqslant u_j \text{ for all } j \in \mathcal{N}\}$$

is called the set of roundings *of $\bar{x}$.*

In general, $\mathcal{R}(\bar{x})$ is a mixed integer set, a disjoint union of $2^{|\mathcal{F}|}$ polyhedra, with $\mathcal{F}$ being the set of fractional variables. Note that in the special case of pure integer problems, hence $\mathcal{I} = \mathcal{N}$, the set of roundings of $\bar{x}$ is a $2^{|\mathcal{F}|}$-elementary lattice, more precisely, the vertices of an $|\mathcal{F}|$-dimensional unit hypercube:

$$\mathcal{R}(\bar{x}) = \Big\{x \in \mathbb{Z}^n \mid x_{\mathcal{I}\setminus\mathcal{F}} = \bar{x}_{\mathcal{I}\setminus\mathcal{F}}, x_{\mathcal{F}} \in \bigtimes_{j\in\mathcal{F}} \{\lfloor \bar{x}_j \rfloor, \lceil \bar{x}_j \rceil\}\Big\} \subseteq \bigtimes_{j\in\mathcal{I}} \{l_j, \dots, u_j\}.$$

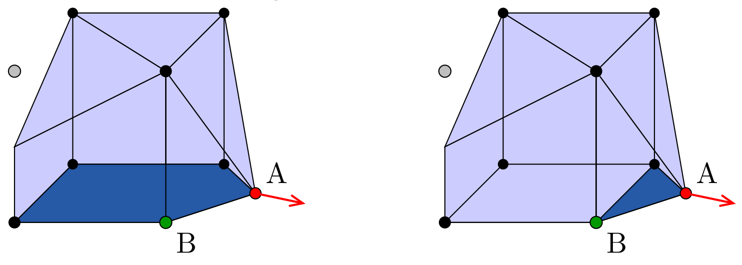
Here, $x_{\mathcal{F}}$ and $x_{\mathcal{I}\setminus\mathcal{F}}$ denote the projection of x to the space of fractional and integral variables, respectively.

Definition 7.1 (optimal rounding). *Let an MINLP P, $\bar{x} \in [l,u]$ and $\tilde{x} \in \mathcal{R}(\bar{x})$.*

1. *We call $\tilde{x}$ a* feasible rounding *of $\bar{x}$, if $g_i(\tilde{x}) \leqslant 0$ for all constraints $i \in \mathcal{M}$ of P.*
2. *We call $\tilde{x}$ an* optimal rounding *of $\bar{x}$, if $\tilde{x} \in \operatorname{argmin}\{c^\mathsf{T} x \mid g_i(x) \leqslant 0 \text{ for all } i \in \mathcal{M}, x \in \mathcal{R}(x)\}$.*
3. *We call $\bar{x}$* roundable *if it has a feasible rounding.*

The idea of our newly proposed LNS algorithm is to define a sub-MINLP that optimizes over the set of roundings of a relaxation optimum $\bar{x}$. This

Figure 7.1.: RENS for MIP: original MIP (light), sub-MIP received by fixing (dark, left) and 0-1 sub-MIP by additional bound reduction (dark, right). The red point A shows the optimum of the LP relaxation, the green point B is the optimal rounding of A.



is done by fixing all integer variables that take an integral value in $\bar{x}$. For the remaining integer variables, the bounds get tightened to the two nearest integral values, see Figure 7.1. Note that in the case of a problem for which all integer variables are *binary* and that has a *completely fractional* relaxation optimum, the subproblem would be identical to the original. We will therefore use a threshold for the percentage of integral variables, see next section.

If the sub-MINLP is solved by using a linear outer approximation, tightening the variable bounds to the nearest integers often improves the dual bound, since reduced domains give rise to a tighter linear relaxation. Technically, all integer variables can be easily transformed to binary variables, by substituting $x'_j = x_j - l_j$. Binary variables are preferable over general integers since many MIP-solving techniques such as domain propagation via probing [Sav94], knapsack cover cuts [Bal75, HJP75, Wol75], or the primal heuristic OCTANE [BCD$^+$01] are only used for binary variables.

As the sub-MINLP is completely defined by the relaxation solution $\bar{x}$, we call the procedure *relaxation enforced neighborhood search*, or shortly RENS. Note that unlike RINS [DRP04], RENS does not require a known feasible solution.

Figure 7.2 shows the basic algorithm, which by construction has some important properties:

Lemma 7.2. *Let the starting point $\bar{x}$ be feasible for the NLP relaxation.*

1. *A point $\tilde{x}$ is a feasible solution of the sub-MINLP if and only if it is a feasible rounding of $\bar{x}$, in particular:*

2. *a point $\tilde{x}$ is an optimal solution of the sub-MINLP if and only if it is an optimal rounding of $\bar{x}$, and*

3. *the sub-MINLP is infeasible, if and only if no feasible rounding of $\bar{x}$ exists.*

Two major features distinguish RENS from other MIP and MINLP primal heuristics known from the literature. Firstly, the RENS algorithm does not

Figure 7.2.: Generic RENS algorithm

```
Input  : MINLP $P$ as in (2.1)
Output: feasible solution $\tilde{x}$ for $P$ or $\emptyset$
/* compute optimal solution of the NLP relaxation of $P$    */
1 $\bar{x} \leftarrow \operatorname{argmin}\{c^\mathsf{T} x \mid g_i(x) \leqslant 0 \text{ for all } i \in \mathcal{M}, x \in [l,u]\}$;
2 forall the $j \in \mathcal{I}$ do
3     if $\bar{x}_j \in \mathbb{Z}$ then
4         fix $x_j$: $l_j \leftarrow \bar{x}_j, u_j \leftarrow \bar{x}_j$;
5     else
6         change to binary bounds: $l_j \leftarrow \lfloor \bar{x}_j \rfloor$, $u_j \leftarrow \lceil \bar{x}_j \rceil$;
/* solve the resulting sub-MINLP of $P$                       */
7 $\tilde{x} \leftarrow \operatorname{argmin}\{c^\mathsf{T} x \mid g_i(x) \leqslant 0 \text{ for all } i \in \mathcal{M}, x \in [l,u], x_j \in \mathbb{Z} \text{ for } j \in \mathcal{I}\}$;
8 return $\tilde{x}$;
```

require a known feasible solution, unlike other large neighborhood search heuristics that have been described for MIP, namely RINS [DRP04], Local Branching [FL03], Crossover [Ber06, Rot07], DINS [Gho07], or Proximity Search [FM13]. It is a start heuristic, not an improvement heuristic. The same holds for nonlinear variants of these heuristics [BHPV11, BG12, NBL08], see also Chapter 6.

Secondly, RENS solves a *single* sub-*MINLP*. In contrast, most primal heuristics for MINLP, in particular the various nonlinear feasibility pump versions [BCLM09, BG12, DFLL10, DFLL12] and Chapter 5, RECIPE [LNM10, LMN11] and Iterative Rounding [NB12], solve a *series* of auxiliary *MIPs*, often alternated with a sequence of NLPs, to produce a feasible start solution. The number of iterations is typically not fixed, but depends on the instance at hand.

7.3. Design and implementation details

In this section, we discuss the details of our RENS implementation. A particular focus is set on the application of RENS as a subsidiary method inside a complete branch-and-bound solver.

In principle, an arbitrary point may be used as starting point in line 1 of the RENS algorithm, see Figure 7.2. Most global solvers for MINLP are based on branch-and-bound and involve the solution of an NLP relaxation or a linear outer approximation. Their optima are natural choices as starting points. While the NLP optimum is supposed to be "closer" to the feasible region of the MINLP, the LP optimum can usually be computed faster and often gives rise to smaller subproblems. More precisely, the NLP fulfills all nonlinear constraints $g_i(x) \leq 0$, whereas the LP, if solved with the simplex algorithm, tends to fulfill more integrality constraints, which reduces the computational complexity of the RENS subproblem. Also, when using RENS inside a complete solver, one or both points (optima of the LP and the NLP relaxation) may already have been computed in advance by other components of the solver and they may therefore be used “for free”. Altogether, both relaxations have their pros and cons; which one proves better in practice will be investigated in our empirical studies, see Section 7.5.

When using a linear outer approximation (the LP relaxation in case of MIP), an important question is whether we should use the optimum of the initial LP relaxation or the LP solution after cutting planes have been applied. As before, cutting planes strengthen the formulation, but it is generally assumed that they tend to produce more fractional LP values. As before, in Section 7.5 we examine which relaxation works best.

If RENS is used as a primal heuristic embedded in a complete solver, further modifications are necessary to obtain a good overall performance. When primal heuristics are considered as standalone solving procedures, e.g., the Feasibility Pump [AB07, BFL07, BCLM09, DFLL10, DFLL12, FGL05, FS09], the algorithmic design typically aims at finding feasible solutions for as many instances as possible, even if this takes substantial running time. However, if they are used as supplementary procedures inside a complete solver, the overall performance of the solver is the main objective.

To this end, it is often worth sacrificing success on a small number of instances for a significant saving in average running time. The Stage 3 of the Feasibility Pump for MIPs[26] is a typical example of a component that is crucial for its impressive success rate as a standalone algorithm, but it will not be applied when the Feasibility Pump is used inside a complete solver, see [Ber06]. RENS principally is an expensive algorithm that solves an $\mathcal{NP}$-hard problem; therefore, the decision of when to call it should be made carefully to avoid slowing down the overall solving process. The remainder of this section describes some algorithmic enhancements, most of which are concerned with identifying which subproblems are the most promising for calling RENS and on which subproblems it should be skipped.

First, RENS should only be called if the resulting sub-MINLP seems to be substantially easier than the original one. This means that at least a specific ratio of all integer variables, say $r_1 \in (0, 1)$, or a specific ratio of all variables including the continuous ones, say $r_2 \in (0, 1)$, should be fixed. The first criterion limits the difficulty of the discrete part of the sub-MINLP itself, the second one limits the total size of the relaxations that will have to be solved. For example, think of a MIP which consists of 20 integer and 10 000 continuous variables. Even if one fixes 50 % of the integer variables, RENS would be a time-consuming heuristic since solving the LPs of the sub-MIP

[26]Stage 3 of the Feasibility Pump solves (a reformulation of) the original MIP with a new objective function. It minimizes the distance to an infeasible point gained from the pumping algorithm; more precisely to the one which was closest to the polyhedron associated to the LP relaxation. For details, see Chapter 5 or [FGL05].

would be nearly as expensive as solving the ones of the original MIP. Since by propagation, fixing integer variables might also lead to fixed continuous variables, e.g., for variable bound constraints, we check the latter criterion only after presolving the subproblem.

Second, the sub-MINLP does not have to be solved to proven optimality. Therefore, we decided to use limits on the solving nodes and the so-called *stalling nodes* of the sub-MINLP. The absolute solving node limit nl_1 is a hard limit on the maximum number of branch-and-bound nodes that should be processed. The stalling node limit nl_2 indicates how many nodes should at most be processed without an improvement to the incumbent solution of the sub-MINLP.

Third, partially solving the sub-MINLP aims at finding a good primal solution quickly. Hence, algorithmic components that mainly improve the dual bound, such as cutting plane separation, and that are very time-consuming, such as strong branching, can be disabled or reduced to a minimum. Further on this list are conflict analysis, pairwise presolving of constraints, probing and other LNS heuristics. As branching and node selection strategies we use Inference Branching and Best Estimate Search, see, e.g., [Ach07a].

RENS could be either used as a pure start heuristic, calling it exclusively at the root node, or frequently throughout the branch-and-bound search to find rounded solutions of local LP optima. In particular when the integrality of the root LP relaxation falls below the minimum fixing ratio r_1, it seems reasonable to employ RENS at deeper levels of the tree where the number of fractional variables tends to be smaller. For the case of repeated calls of RENS, we implemented a few strategies to determine the points at which RENS should be called.

How often RENS should be called mainly depends on two factors: how expensive is it for a particular instance and how successful has it been in previous calls for that instance? The first can be estimated by the sum n_{RENS} of branch-and-bound nodes RENS used in previous calls in comparison to n_{all}, the number of branch-and-bound nodes already searched in the master problem. The second can be measured by the success rate $s = \frac{n_{\text{sols}}+1}{n_{\text{calls}}+1}$, where n_{calls} denotes the number of times RENS has been called and n_{sols} denotes the number of times it contributed an improving solution, respectively. In our implementation, we computed the stalling nodes limit as

$$nl_2 = 0.3 n_{\text{all}} \cdot s - n_{\text{RENS}} + 500 - 100 n_{\text{calls}}.$$

The last term represents the setup costs for the subproblem which accrue even if subproblem solving terminates quickly. The offset of 500 nodes ensures that the limit is reasonable for the first few calls of RENS. We only start RENS if nl_2 is sufficiently large.

In an LP-based branch-and-bound search, consecutive nodes tend to have similar LP optima. This is due to the similarity of the solved problems as well as to the warm-starting technique of the simplex algorithm, which is typically used for this purpose. Since similar LP optima most likely lead to similar results for the quite involved RENS heuristic, it should not be called in consecutive nodes, but the calls should rather be spread equally over the tree. Therefore, we use a calling frequency f: RENS only gets called at every f-th depth of the tree.

7.4. Experimental setup

This section proposes three computational experiments that evaluate the potential of RENS to find optimal rounded solutions, compare RENS to other rounding heuristics, and demonstrate the impact of RENS inside a full-scale branch-and-bound solver. We conduct these experiments on three different test sets of MIPs, MIQCPs, and MINLPs in order to analyze RENS on different classes of mathematical programs. All test sets are compiled from publicly available libraries.

For all computational experiments, we used SCIP 2.1.1.1 compiled with SoPlex 1.6.0 [Wun96, Sop] as LP solver, Ipopt 3.10 [WB06, Ipo] as NLP solver, and CppAD 20110101 [Cpp] as expression interpreter for evaluating general nonlinear constraints. The results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20GHz with 12 MB cache and 48 GB main memory, running an openSuse 11.4 with a gcc 4.5.1 compiler. Hyperthreading and Turboboost were disabled. In all experiments, we ran only one job per node to avoid random noise in the measured running time that might be caused by delays if multiple processes share common resources, in particular the memory bus.

Test sets

We used all instances from Miplib3.0 [BCMS98], Miplib2003 [AKM06], and the Miplib2010 benchmark set [KAA$^+$11] as MIP test set. We excluded instances `air03`, `ex9`, `gen`, `manna81`, `p0033`, `vpm1`, for which the optimum of the LP relaxation (after SCIP presolving) is already integral, instance `stp3d`, for which SoPlex cannot solve the LP to optimality within the given time limit and instances `sp97ar`, `mine-166-5`, for which SoPlex 1.6.0 fails in computing an optimal LP solution. This leaves 159 instances. We will refer to this test set as MMM (as an abbreviation for three Miplibs).

For MIQCP, we used the test set described in [BGHV12] that comprises instances from several sources. We excluded instances `ex1263`, `ex1265`, `sep1`, `uflquad-30-100`, for which the LP optimum is already integral (but in none of the cases feasible for the quadratic constraints), instances `nuclear14`, `isqp1`, `nuclearva`, for which the LP relaxation is unbounded, the instance `200bar`, for which SoPlex produces an error, `108bar`, `isqp0`, for which SCIP's separation loop has not terminated within the time limit, and those 18 instances that are linear after SCIP presolving, see [BGHV12]. This test set contains 70 instances.

We further tested RENS on general MINLPs from MinlpLib [BDM03], excluding those that are MIQCPs, that are linear after SCIP presolving, or that contain expressions which cannot be handled by SCIP, e.g., trigonometric functions. We also excluded `4stufen`, `csched1a`, `st_e35`, `st_e36`, `waters`, for which the optimum of the LP relaxation is integral, and instances `csched2`, `minlphix`, `uselinear`, for which the LP relaxation is unbounded, leaving 105 instances. It remains to be said that this particular test set is not as heterogeneous as the others, since there are many instances of similar type.[27]

Analyzing roundability and computing optimal roundings

In a first test, we employ RENS to analyze the roundability of optimal relaxation solutions. For this, we ran RENS without any node limits or variable fixing thresholds on the test sets described above. A time limit of two hours, however, was set for solving the RENS subproblem.

We used an optimal solution of the LP relaxation as starting point for the MIP test. We compare the performance of RENS using the “original” LP optimum before the cutting plane separation loop versus the one after cuts. One question of interest here is how the integrality of the LP solution interacts with the feasibility of the sub-MIP. The desired situation is that the LP solution contains a lot of integral values, but still gives rise to a feasible RENS problem. For this and the following experiments, we restricted ourselves to analyzing the optimal solution which is reported by SoPlex; we did not investigate differences in performance when using alternative optimal solutions as it has been done for instance in [Ach10, Ach11] for cutting planes.

For the MIQCP and the MINLP test run, we further evaluate how different types of relaxations, namely the LP and the NLP relaxation, behave w.r.t. the roundability of their optima and the quality of the rounded solutions. The results shall give an insight into which solutions should be used as starting points for RENS and other primal heuristics. Here, the performance in terms of running time of the RENS heuristic has to be weighed up against the success rate and quality of solutions produced with different relaxations.

Evaluating the performance of rounding heuristics

In a second test, we use RENS for the analysis of several MIP rounding heuristics, see Chapter 4. The results shall give an insight into how often these heuristics find a feasible rounding and how good the quality of this solution is compared to the optimal rounding.

All considered rounding heuristics iteratively round all variables that take a fractional value in an optimal solution of the LP relaxation. One rounding is performed per iteration step, without resolving the relaxation. We

[27]This holds, to a certain extent, for all general MINLP test sets that the author is aware of.

recapitulate from Chapter 4 that

- ▹ *Simple Rounding* only performs roundings, that maintain feasibility;
- ▹ *ZI Round* conducts roundings, using row slacks to maintain primal feasibility;
- ▹ *Rounding* conducts roundings, that potentially violate some constraints and reduces existing violations by further roundings.

The main decision criterion for all these primal heuristics is the number of down- and uplocks, see Definition 2.10. ZI Round and Rounding both are extensions of Simple Rounding. Both are more powerful, but also more time-consuming. For details on these rounding heuristics and their implementation in SCIP, see [Ber06, Hen11, Wal10] and Chapter 4.

Note that rounding heuristics are quite defensive, in the sense that they often round opposite to the variable's objective function coefficient and thereby "sacrifice" optimality for feasibility. Hence, we do not expect them to often detect the optimal rounding computed by RENS. The question is rather for how many of the roundable instances these heuristics find any feasible solution and only as a second point how big the gap to the optimal rounding is.

Impact of RENS on the overall performance

In a third test, we evaluate the usefulness of RENS when applied as a primal heuristic inside a branch-and-bound solver. For comparison consider the RINS algorithm [DRP04], an LNS improvement heuristic which is applied in Cplex [DRP04] and Gurobi [BRG09b]. RINS uses two starting solutions, a relaxation optimum and the incumbent. It fixes variables which take identical values in both solutions, cf. Chapter 6.

The advantage of RENS in contrast to RINS is that it does not require a given primal solution and that it always fixes at least the same number of variables as RINS, if applied to the same relaxation solution. The advantage of RINS is that the RINS subproblem is guaranteed to contain at least one feasible solution, namely the given starting solution.

To assess RENS as a primal heuristic, we run SCIP with RENS applied exclusively as a root node heuristic and SCIP with RENS applied both at the root and throughout the search. For this experiment, we used a reduced version of RENS which requires a minimal percentage of variables to be fixed and which stops after a certain number of branch-and-bound nodes, see Section 7.3. For comparison, we ran SCIP with RENS deactivated.

The main criteria to be analyzed in this test are the impact of RENS on the quality of the primal bound early in the search and the impact of RENS on the overall performance. While we hope for improvements in the former, a major improvement in the latter is not to be expected. Different studies show that the impact of primal heuristics on time to optimality often is slim. Bixby et al. report a deterioration of only 9 % if deactivating all primal heuristics in Cplex 6.5, Achterberg [Ach07a] presents a performance loss of 14 % when performing a similar experiment with SCIP 0.90i, in [Ber06] differences of at most 5 % for deactivating single primal heuristics are given. Compare also Chapters 3 and 11. Therefore, a good result for this experiment would be an improvement on the primal bound side, coming with no deterioration to the overall performance.

7.5. Computational experiments

As a first test, we ran rens without node or variable fixing limits, to evaluate its potential to find optimal roundings of optimal LP and NLP solutions.

The results for MIP can be seen in Tables B.6 and B.7 in the appendix, those for MIQCP in Tables B.8 and B.9, those for MINLP in Tables B.10 and B.11; aggregated results can be found in Table 7.1. Each table presents the names of the instances, Int, the percentage of integer variables that were fixed by rens, All, the percentage of all variables that were fixed after presolving of the rens subproblem, TimeS, the time SCIP needed before rens was called, Time and Nodes, the running time and the number of branch-and-bound nodes needed to solve the subproblem to optimality, Solution, the best solution found in the rens subproblem, and Found At, the node in the subproblem's branch-and-bound tree at which it has been found. Note that these values are rounded, e.g., the 100.0% given in column Int of Table B.6 for nw04 represents a ratio of $^{87460}/_{87482}$.

If the subproblem was proven to be infeasible or no solution was found within the time limit, this is depicted by an "–" in the column Solution. When the time limit of two hours was hit in the rens subproblem, this is indicated by the term limit in the Time column. Hence, for all instances that do not hit the time limit, the column Solution depicts the proven optimal rounding of the relaxation solution and "–" indicates that it was proven that no feasible rounding exists. Instances for which the optimal rounding is an optimal solution of the original MINLP are marked by a star.

The correlation between the percentage of fixed variables and the success of rens is depicted in Figures 7.3–7.6. Each instance is represented by a cross, with the fixing rate being the x-coordinate, and 0 or 1 representing success or failure as y-coordinate. The dotted blue line shows a moving average taken over ten consecutive points and the dashed red line shows a moving average taken over 30 consecutive points. A thin gray line is placed at the average success rate taken over all instances of the corresponding test set.

If we have to average running times or number of branch-and-bound nodes, we use a shifted geometric mean, see Definition 2.12, with a shift of $s = 10$ for time and $s = 100$ for nodes. In the given mean numbers, instances hitting the time limit are accounted for with the time limit and the number of processed nodes at termination.

Unless otherwise noted, the term variables always refers to *integer* variables for the remainder of this section. Consequently, by *integral variables* we mean integer variables that take an integral value in a given relaxation solution – as contrasted with fractional variables.

Table 7.1 summarizes the results from Tables B.6–B.11. Each line of it represents one combination of a test set and a choice of a relaxation. Column ψ(int) shows the average percentage of integral variables, Columns >90 % and 0 % show for how many instances more then 90 % or none of the variables are integral, respectively. The Column round depicts the number of roundable instances in the test set and Column opt shows for how many instances the optimal rounding actually is an optimal solution of the original problem. Column bett/wor indicates for two consecutive table lines that refer to the same test set, how often the one or the other was better or worse w.r.t. to the objective function value of the optimal rounding. Finally, ϕ(nodes) and ϕ(time) provide us with the shifted geometric means of the branch-and-bound nodes and the overall running time required to solve the subproblem.

Table 7.1.: RENS results for computation of optimal roundings (aggregated results)

	integrality			success			comp. effort	
test set, relax	ψ(int)	>90 %	0 %	round	opt	bett/wor	ϕ(nodes)	ϕ(time)
MIP + cuts	71.7 %	55	3	95	34	38:7	814.4	22.6
MIP − cuts	73.6 %	62	3	80	20	7:38	719.9	21.7
MIQCP (LP)	59.9 %	9	10	49	9	1:27	627.7	30.9
MIQCP (NLP)	13.8 %	1	47	48	26	27:1	7078.8	168.1
MINLP (LP)	63.5 %	6	4	65	13	2:37	11175.6	83.0
MINLP (NLP)	15.0 %	1	68	73	47	37:2	93908.0	262.7

Computing optimal roundings: MIP

In Table B.6 in the appendix, we see that for roughly one third ($^{55}/_{159}$) of the instances, more than 90 % of the variables took an integral solution in the optimal LP solution. In contrast to that, there are only 22 instances for which the portion of integral solution values is less than 40 %. The average percentage of variables with integral LP solution value is 71.7 %. There are a few cases with many continuous variables for which fixing the majority of the integer variables did not result in a large ratio of all variables being fixed, see, e.g., `dsbmip` or `p5_34`. This is the reason that we will use two threshold values for later tests, see Section 7.3.

For 59.7 % ($^{95}/_{159}$) of the instances, RENS found a feasible rounding of the LP optimum. For 15 of these instances, the RENS subproblem hit the time limit, eleven of them are from MIPLIB 2010. For the remaining 80 instances, the solutions reported in Table B.6 are the optimal roundings of the given

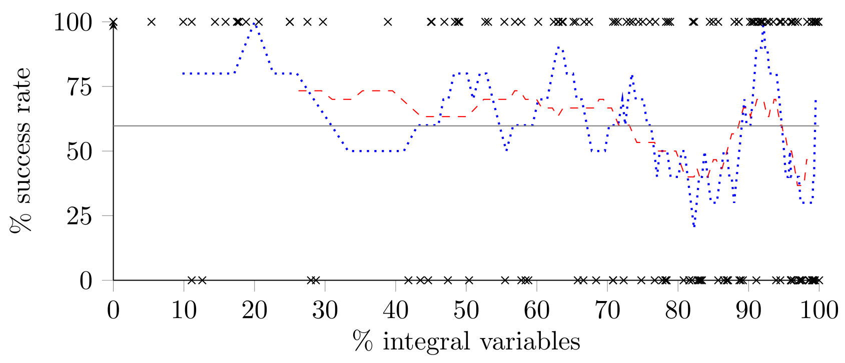


Figure 7.3.: Moving averages of success rate, MMM instances, after cuts

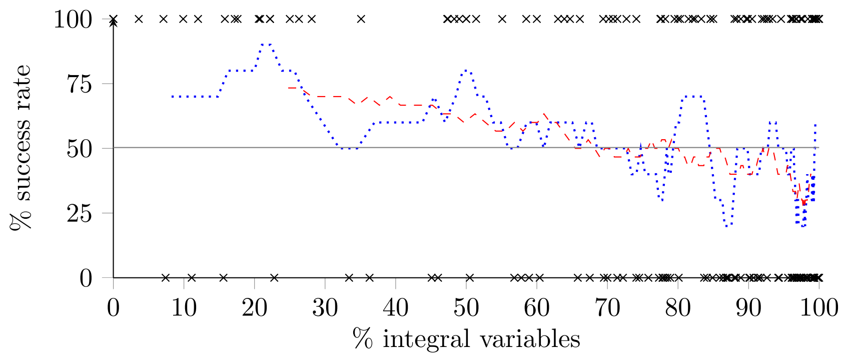


Figure 7.4.: Moving averages of success rate, MMM instances, before cuts

starting solutions. For 34 instances, the optimal rounding coincides with the global optimal solution.

We further observe that the success rate is only weakly correlated to the ratio of fixed variables. The success rate on the instances with more than 90 % fixed variables was nearly the same as on the whole test set, namely 58.2 %. This is an encouraging result for using RENS as a start heuristic inside a complete solver: very small subproblems contain feasible solutions.

The connection between the fixing rate and the success rate is also depicted in Figure 7.3. We see that the success rate decreases slightly, at about 75 % fixed variables, but the difference between low and high fixing rates is not huge.

We further observe that proving the non-existence of a feasible rounding is relatively easy in most cases. For 59 out of 64 infeasible rounding subproblems, infeasibility could be proven in presolving or while root node processing of the subproblem. There is only one instance, `pigeon-10`, for which proving infeasibility takes more than 600 nodes. Considering the running time, infeasibility could be proven in less than a second in 56 of 64 cases, with only one instance, Papp1-2, taking more than 15 seconds. The instance `neos-1601936` is the only one for which feasibility could not be decided within the given time limit; hence, it is the only instance for which we could not decide whether the optimal LP solution is roundable or not.

The results for using the LP optimum before cutting plane separation are shown in Table B.7 in the appendix. Even more instances, 62 compared to 55, have an integral LP solution for more than 90 % of the variables. However, there is one more (24 vs. 23) instance, for which the portion of integral solution values is less than 40 %. Contrary to what one might expect, the average percentage of variables with integral LP value is hardly affected by cutting plane separation: it is 73.6 % before separation and 71.7 % after.

The number of instances for which RENS found a solution, however, goes down: 80 instead of 95, which is only half of the test set. This is particularly due to those instances with many variables that take an integral value. Consequently, the success rate of RENS drops with an increase in the ratio of fixed variables. When RENS is called before cutting planes are added, fewer of the optimal roundings are optimal solutions to the original problem: 20 compared to 34, when called after cuts. Note that the better mean running time which is given in Table 7.1 mainly is due to not performing the separation loop.

We conclude that, although the fractionality is about the same, LP solutions before cuts are less likely to be roundable and the rounded solutions are often of inferior quality. In other words: before cutting planes, integral solution values are more likely to be misleading (in the sense that they cannot be extended to a good feasible solution). This is an important result for the design of primal heuristics in general and confirms the observation that primal heuristics work better after cutting plane separation, see, e.g., [FS09].

Computing optimal roundings: MIQCP

For MIQCP, we tested RENS with LP solutions, see Table B.8 in the appendix, and with NLP solutions, see Table B.9, as starting points. We also experimented with the LP solution before cuts; the results were much worse and are therefore not shown.[28]

The ratio of integral LP values is smaller compared to the MMM problems: there are only 9 out of 70 instances for which more than 90 % of the variables were integral, but there are 10 instances for which all variables were fractional. Note that this does not necessarily mean that the RENS sub-MIQCP is identical to the original MIQCP, e.g., when general integer variables are present. In this case, the RENS subproblem corresponds to the original problem intersected with the integral lattice-free hypercube around the starting

[28]Note that for MIQCP and MINLP, cutting plane separation is even more substantial than for MIP. In MIP all constraints are present in the LP relaxation from the beginning, whereas in MINLP cutting plane separation is used to add a local approximation of them to the LP relaxation. In the extreme case of an MINLP that only consists of general nonlinear constraints, the LP relaxation before cutting planes will only be the domain box of the variables. In particular, all integer variables will take integral or infinite values in an optimal LP solution.

solution. On average, 59.9 % of the variables took an integral value. The success rate of RENS is even better than for MIPs: In 49 out of 70 instances (70 %), RENS found a feasible rounding. Note that this is not due to the 10 instances for which all variables were fractional: three of them also fail. Moreover, the success rate appears not to depend on the percentage of fixed variables, see Figure 7.5.

Deciding feasibility, however, seems to be more difficult. Out of ten instances hitting the time limit, there were eight for which RENS did not find a feasible rounding. For 13 instances, infeasibility of the rounding problem was proven, mostly in presolving or within a few branch-and-bound nodes. Nine times, the optimal rounding was identical to the optimal solution of the MIQCP.

The next observation we made is that the NLP solution tends to be much less integral than the LP solution, on average only 13.8 % of the variables take an integral value, see Table B.9 and Figure 7.5. This is due to the fact that in our experiments the LP solution was computed with the simplex algorithm which tends to leave variables at their bounds, whereas the NLP solution was computed with an interior point algorithm that tends to choose values from the interior of the variables' domains.

Surprisingly, this did not enhance the roundability. For 48 instances, RENS found a feasible rounding of the NLP optimum, compared to 49 for the LP. Worth mentioning, this was nearly the same set of instances, and there were 46 on which both versions found a solution. The solution quality, however, was typically better when using an NLP solution: 27 times the NLP solution yielded a better rounding, only once the LP was superior. Since the LP relaxation itself is a relaxation of the NLP relaxation, this result could be expected. 26 times, the optimal rounding was even an optimal solution of the original MIQCP.

The higher fractionality of the NLP relaxation is expressed in a much larger search space. In shifted geometric mean, RENS processed 628 search nodes if starting from an LP solution, 7078 if starting from an NLP solution. The shifted geometric mean of the running time (Time) is roughly 5.5 times larger: 30.9 vs. 168.1 seconds.

We conclude that the same observation holds as in the MIP case: small subproblems (in case of using the LP) generate high-quality feasible solutions. Although the solution quality is improved by using an NLP relaxation, the computational overhead and the success rate are not encouraging to make this a standard setting if using RENS as a primal heuristics inside a complete solver.

Computing optimal roundings: MINLP

For MINLP, we again compared two versions of RENS: one using the LP solution and one using the NLP solution as starting point, see Tables B.10 and B.11 in the appendix, respectively. For the same reason as in the MIQCP

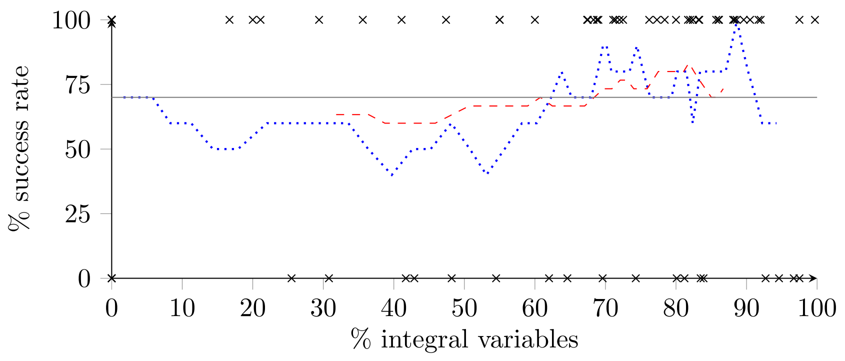


Figure 7.5.: Moving averages of success rate, MIQCP, LP sol., after cuts

case we omitted the results for the LP solution before cuts.

The integrality of the LP solutions is comparable to the MIQCP case. On average, 63.5 % of the variables take an integral value; there are 6 out of 105 instances for which more than 90 % of the variables are integral, and only four instances for which all variables are fractional. For this test set, we see a clearer connection between the ratio of fractional variables and the success rate of RENS. The more variables are integral, the lower the chance for RENS to succeed, see Figure 7.6.

For seven instances, the RENS subproblem hit the time limit of two hours, always without having found a feasible solution. Overall, 65 out of 105 (62 %) of the LP solutions proved to be roundable, which is similar to the MMM results. In all cases, RENS found the optimal rounding. Generally, RENS needs much more nodes to solve the rounding problem as compared to the other tests.

Using the NLP instead of the LP relaxation slightly increases the success rate: 73 times, RENS finds a feasible rounding. As for MIQCPs, the quality is typically better (37 vs. 2 times), which comes with a much lower integrality of 15 % on average, 68 instances having all variables fractional, and a huge increase in running time: a factor of more than three in shifted geometric mean.

Computing optimal roundings: summary

Interestingly, the integrality and roundability of LP solutions is very similar for MIPs, MIQCPs and MINLPs: on average, only 30 % to 40 % of the variables are fractional and for 60 % to 70 % of the instances RENS found a feasible rounding. We further observed that most often the RENS subproblem could be solved to proven optimality and that the success rate of RENS is only weakly correlated to the fractionality. These three insights are very encouraging for applying RENS as a start heuristic inside a complete solver, see below. A summary of the results on computing optimal roundings can be found in Table 7.1.

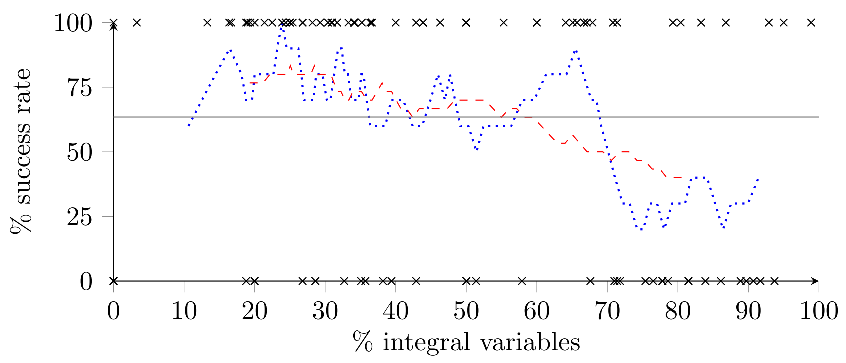


Figure 7.6.: Moving averages of success rate, MINLP, LP sol., after cuts

We further performed a McNemar test, see [McN47] or Section 11.1, to analyze the statistical significance of the results. As null hypothesis we assume that the LP and the NLP solution (or the LP before and after cuts) are equally likely to yield a feasible rounding. For the MMM test set, the null hypothesis gets rejected with a p-value of 0.0011 and for MINLP with 0.0114. For MIQCP, the p-value is 0.6547. This means that for MIP the LP solution after cuts is more likely to be roundable with very high probability, for MINLP the NLP solution is more promising with high probability, for MIQCP there is no statistically significant difference.

We conclude that the solutions found by RENS are usually better when it is applied after cutting plane separation and that using an NLP instead of an LP relaxation does not give a good trade-off between solution quality and running time: it might be better, but the computational overhead is huge.

Analyzing rounding heuristics

Our next experiment compares RENS applied to the LP solution after cuts with the three pure rounding heuristics that are implemented in SCIP, see Chapter 4. The results for the MMM instances are shown in Table B.12 in the appendix. Instances for which none of the compared methods could provide a solution are omitted in the presentation.

As implied by definition, the solutions found by RENS (if the subproblem has been solved to optimality) are always better or equal to the solutions produced by any rounding heuristic. As expected, the solution quality of Rounding and ZI Round is always better or equal to Simple Rounding, and ZI Round often is superior to Rounding. Since Simple Rounding, Rounding, and ZI Round all endeavor to feasibility and neglect optimality, it is not too surprising that there are only three instances, for which Simple Rounding and Rounding find an optimal rounding; four in the case of ZI Round.

A comparison of the number of solutions, however, shows that there is a big discrepancy between the number of instances which have a roundable LP optimum (95) and the number of instances for which these heuristics succeed

(37 for ZI Round, 36 for Rounding, and 27 for Simple Rounding). Of course, this has to be seen under the fact that these heuristics are much faster than RENS. The maximum running time was attained by Rounding on instance `opm2-z7-s2`; it was only 0.09 seconds.

For the MIQCP and MINLP test sets, the situation was even more extreme. The rounding heuristics were unable to produce a feasible solution for any of the instances – even though the previous experiments proved that 60 % to 70 % of the LP solutions are roundable. This is most likely due to the special design of these heuristics: they solely work on the LP relaxation. This result demonstrates the need for rounding heuristics that take the special requirements of nonlinear constraints into consideration. Note that in this context, we do not consider Iterative Rounding [NB12] to be a rounding heuristic, since it solves a series of MIPs and NLPs.

Table 7.2.: RENS as primal heuristic inside SCIP (aggregated results), numbers of instances for which RENS was called and succeeded at least once

	at root		in tree	
	called	found	called	found
MIP (of 160)	124	63	154	87
MIQCP (of 70)	45	31	60	42
MINLP (of 105)	45	9	99	39

Table 7.3.: RENS as primal heuristic inside SCIP (aggregated results), computational effort

	No RENS		Root RENS		Tree RENS	
test set	Nodes	Time	Nodes	Time	Nodes	Time
MIP: arithm. mean	1 446 078	2461.4	1 442 400	2427.0	1 443 404	2414.3
sh. geom. mean	11 248	377.2	10 390	366.3	10 346	365.8
MIQCP: arithm. mean	659 740	2872.3	677 123	2927.0	664 117	2888.6
sh. geom. mean	6 457	229.9	6 361	232.0	6 193	229.9
MINLP: arithm. mean	2 338 903	3274.5	2 324 208	3274.7	1 925 902	3168.7
sh. geom. mean	58 758	466.5	58 406	467.1	51 066	431.3

RENS as primal heuristic inside SCIP

Finally, we evaluate whether a reduced version of the full RENS algorithm is suited to serve as a primal heuristic applied inside a complete solver. Based on the results from our first experiment, considering the running times and the node numbers at which the RENS subproblems find their optimal solutions, we decided to use 50 % as a threshold value for r_1, the minimal fixing rate for integer variables, in this run. The minimal fixing rate for all variables r_2 was set to 25 %. We used an absolute node limit nl_1 of 5000 and computed the stalling node limit nl_2 as given in Section 7.3. Because of the long running times, we refrained from using an NLP relaxation, although it might produce better solutions. We always used the LP solution after cutting planes as a starting solution.

For this experiment, interactions of different primal heuristics among each other and with other solver components come into play. SCIP applies eleven primal heuristics at the root node, in particular the rounding heuristics against which we compared in the previous subsection. Of course, a primal heuristic called prior to RENS might already have found a solution which is better than the optimal rounding, or in an extreme case, the solution process might already terminate before RENS is called. Further, any solution found before RENS is called might change the solution path. It might trigger variable fixings by dual reductions, which lead to a different LP and hence to a different initial situation for RENS. Since RENS is relatively time-consuming, it is among the last heuristics that are performed. Only Undercover, see Chapter 8, has a lower priority.

The results are shown in Tables B.13–B.15 in the appendix. We compare SCIP without the RENS heuristic (No RENS) against SCIP with RENS applied at most once at the root node (Root RENS) and SCIP with RENS applied at every tenth depth of the branch-and-bound tree (Tree RENS). Columns Nodes and Time show the number of branch-and-bound nodes and the running time SCIP needs to solve an instance to proven optimality. If a limit was hit, this is indicated by the term limit in the time column and the node number at which the solution process stopped is preceded by a '>'-symbol. At the bottom of the table, the arithmetic means and the shifted geometric means of the number of branch-and-bound nodes and the running time are given.

A summary of the results is given in Tables 7.2 and 7.3. The Columns called and found in Table 7.2 show for how many instances RENS was called and found a feasible solution, respectively. Table 7.3 depicts the arithmetic means and the shifted geometric means of the number of branch-and-bound nodes and the running time for each combination of the three different settings and the three test sets.

First, let us consider the results for MIP, see Table B.13. Due to the a-priori limits, RENS was called at the root node for only 124 out of the 160 instances. Out of these, RENS found a feasible solution in 63 cases, which corresponds to a success rate of 50 %, compared to 59 % without any limits, see above. We conclude that the quite strong node limits for the subproblem do not lower the success rate much. In 61 cases, the solution found by RENS was the best solution found at the root node. Considering that there are ten other primal heuristics applied at the root node, this appears to be a very strong result. When RENS was additionally used during search, it was called on 154 instances, finding feasible solutions for 87 of them.

As is typical for primal heuristics, the impact on the overall performance is not huge. Nevertheless, we see that both versions, calling RENS only at the root and all over the tree, give slight decreases in the arithmetic and shifted geometric means of the node numbers and the running time. Both versions were about 3 % faster and took 8 % less nodes in shifted geometric mean. For the time-outed instances, Root RENS and Tree RENS provided a better primal bound than No RENS eight and nine times, respectively, whereas both were inferior in two cases.

For the MIQCP test set, RENS was called at the root for 45 out of 70 instances, finding a feasible solution in 31 cases. This was always the best solution SCIP found at the root node. The overall performance was about the same: the running time stayed constant for Tree RENS and was increased by less than one percent for Root RENS, whereas the number of branch-and-bound nodes was reduced by 7 % and 2 %, respectively. When RENS is called during search tree processing, there are four instances with a better primal bound at timeout, once it was worse. For calling RENS exclusively at the root, this ratio was 2:0. Also, there is one instance, namely `nuclear14a`, for which only Tree RENS provided a feasible solution.

For MINLP, the lower success rate for the root LPs with large ratios of integral variables is confirmed by this experiment. For 45 out of 105 instances, RENS was called, but in only 9 cases it could improve the incumbent solution. Interestingly, the version that calls RENS during the tree performs really well. There were 42 instances, for which RENS could improve the incumbent at least once during search, `ghg_3veh` being the front-runner with 27 improving solutions in 44 calls of RENS.

The overall performance reflects that situation. The Root RENS setting shows the same behavior as No RENS, the running time is nearly equal on average and in shifted geometric mean, the number of branch-and-bound nodes goes down by one percent, there are hardly any instances for which we see any change in performance. For Tree RENS, however, the shifted geometric mean of the running time and the number of branch-and-bound nodes goes down by 8 % and 13 %, respectively. One might argue that this is mainly because of `enpro48pb` and `fo8_ar4_1` which show a dramatic improvement in performance. But even if we excluded these two instances (and for fairness reasons also `enpro48` and `enpro56pb`, two outliers in the opposite direction), the mean performance gain is 3 % for running time and 8 % for number of branch-and-bound nodes.

We further performed a variant of the Wilcoxon signed rank test [Wil45] (see also Section 11.1) to analyze the statistical significance of the results, using the Stats package of the SciPy project [JOP$^+$]. We ranked the results by the running time factors per instance and calculated one rank sum from the improving instances and one from those which showed a degradation. Instances that showed no or hardly any performance difference (less than one second or less than 1 %) were excluded. As null hypothesis, we assume that a version of SCIP using RENS at the root or throughout the tree does perform equally w.r.t. running time as SCIP without RENS. For the MMM test set, the null hypothesis gets rejected with a p-value of 0.0236 (for Root RENS) and 0.0178 (for Tree RENS) which is below the standard threshold of 0.05 used as level of significance. Not surprisingly, the results for MIQCP indicate that there are no performance differences for this test set: the p-values are 0.6465 and 0.8753 for Root RENS and for Tree RENS, respectively. For MINLP, p-values of 0.3980 and 0.2862 are achieved. Although failing to reject the null hypothesis when a standard threshold is applied, at least the latter could be taken as an indicator that it is more likely that the results are not simply acquired by chance.

Altogether, these experiments show that RENS, in particular for MIP and MIQCP, helps to improve the primal bound at the root node, and hence the initial gap, before the branch-and-bound search starts. Applying RENS exclusively at the root node had a neutral to slightly positive effect on the overall performance, while giving a user the advantage of finding good solutions early. Applying RENS throughout the search was at least as good for all three test sets and showed a nice improvement in the case of MINLP – which was partly due to two outliers. Consequently, RENS is used in the default settings of SCIP.

7.6. Conclusion

In this chapter, we introduced RENS, a large neighborhood search algorithm that, given a MIP or an MINLP, solves a subproblem whose solution space is the set of feasible roundings of a relaxation solution. We demonstrated that most MIP, MIQCP, and MINLP instances have roundable LP and NLP optima and that in most cases, the optimal roundings can be computed efficiently. Surprisingly, the roundability seems not to be related to the fractionality of the starting solution. Knowing the optimal roundings provides us with a benchmark for rounding heuristics; we discovered that the rounding heuristics implemented in SCIP, see Chapter 4, often fail in finding a feasible solution, even though the provided starting point is roundable. They rarely find the optimal rounding.

We further investigated the impact of a reduced version of RENS if applied as a primal heuristic inside a complete solver. We showed that RENS directly helps to improve the primal bound known at the root node for 38 % of the MIP problems and 44 % of the MIQCP problems in the test sets. For MINLP, a version of SCIP that applies RENS frequently during search gave an improvement of 8 % in running time. The impact on the overall performance is minor but measurable, which is typical for primal heuristics.

Since version 0.90, a first version of RENS for MIPs [Ber06] has been part of the SCIP standard distribution and has ever since been employed by default. It constituted one of the first contributions that the author of this thesis made to the SCIP project. For the SCIP release 2.0.0, we added the capabilities to apply RENS to MIQCPs; SCIP 2.1.0 was the first version featuring solution techniques for general MINLPs, including rens for nonlinear problems. The implementation presented in this chapter is identical to the one released with SCIP 3.0.0 and can be accessed in source code at [Sci]. Furthermore, versions of rens have been recently integrated into Bonmin [Bon] and cbc [Cbc]; Cplex[29] and Gurobi feature similar heuristics [Ach, Bix].

[29]In Cplex, the heuristic is called Intinfeas and can be activated by using hidden parameters.

8. Undercover

In this chapter, we introduce *Undercover*, a primal LNS heuristic that explores a mixed integer *linear* subproblem of a given MINLP. This sub-MIP is constructed by fixing a minimal set of variables that suffices to linearize *all* nonlinear constraints. Although general in nature, the heuristic appears most promising for MIQCPs.

Most of this chapter results from joint work with Ambros M. Gleixner and has been published in the proceedings of the EWMINLP 2010 [BG10] and in Mathematical Programming A [BG14]. In [BGKV12], Bley, Gleixner, Koch, and Vigerske analyzed the performance of different solution techniques for open pit mine production scheduling problems and showed that general purpose MINLP solvers often are competitive to handcrafted algorithms. Their model involves nonconvex bilinear mixing constraints $a^p_t o^s_{t-1} = a^s_{t-1} o^p_t$ for nonnegative continuous variable vectors a^p, a^s, o^p and o^s which determine the amount of metal on the stockpile, of metal sent from the stockpile, of ore on the stockpile, and of ore sent from the stockpile, respectively, for each time period t. For this particular problem, fixing either the a or the o variables to values from a feasible solution of the NLP relaxation will always result in a feasible linear subproblem.

Motivated by the experiments shown in [BGKV12], Gleixner suggested a way to generalize this *covering* structure to generic MINLPs, see Section 8.3, and together we designed a primal heuristic that explores *covers of an MINLP*. The result of this joint research is presented in this chapter.

It is organized as follows. In Section 8.1, we give a brief introduction and explain how the Undercover idea differs from other MINLP heuristics. Section 8.2 introduces a first generic version of the Undercover algorithm. In Section 8.3, we describe how to find variables to fix such that the resulting subproblem is linear. Section 8.4 explains how to extract useful information, even if the sub-MIP proves to be infeasible. Finally, Section 8.6 provides computational results that show the effectiveness of Undercover. In Section 8.7, we briefly discuss further variants of the Undercover heuristic that we have experimented with, and Section 8.8 presents concluding remarks.

8.1. Introduction

Undercover is a large neighborhood search start heuristic that explores a sub-MIP of a given MINLP. Therefore, it solves a vertex covering problem to identify a smallest set of variables to fix, a so-called *cover*, such that each nonlinear constraint becomes linear in the remaining variables. Subsequently, these variables are fixed to values obtained from a reference point, e.g., an optimal solution of a linear relaxation. As typical for LNS heuristics, see Chapter 6, each feasible solution of the sub-MIP corresponds to a feasible solution of the original MINLP.

Naturally, fixing variables to values obtained from a relaxation might render the MINLP infeasible, cf. Chapter 7. We apply domain propagation to try to avoid infeasibilities, and conflict analysis to learn additional constraints from infeasibilities that are nonetheless encountered.

During the design of Undercover, our focus was its application as a start heuristic inside a global MINLP solver such as baron [Sah96, TS04], Bonmin [BBC$^+$08], Couenne [BLL$^+$09], Antigone [MF14], minotaur [Mio], or SCIP.

Two major features distinguish Undercover from other primal heuristics for MINLP that have been suggested in the literature. Firstly, unlike most of them [BHPV11, BCLM09, BG12, DFLL12, NBL08], Undercover is not an extension of an existing MIP heuristic towards a broader class of problems. Moreover, it does not even have a clear counterpart in mixed integer linear programming. Secondly, Undercover solves two auxiliary MIPs (one for finding a set of variables to be fixed plus the resulting sub-MIP), and at most two NLPs (possibly one to compute initial fixing values and one for postprocessing the sub-MIP solution). In contrast to Undercover, most large neighborhood search heuristics [BCLM09, DFLL12, LNM10, LMN11, NB12, NBL08] for MINLP solve an arbitrarily large series of MIPs, often alternated with a sequence of NLPs, to produce a feasible start solution. The number of iterations is typically not fixed, but depends on the instance at hand.

For this chapter, we assume w.l.o.g. that $l_i < u_i$ holds for all $i \in \mathcal{N}$ in Definition 2.1, i.e., that the interior of the variable domain box $[l, u]$ is nonempty. This assumption is valid since fixed variables can always be eliminated and replaced by a constant term.

8.2. A generic algorithm

The paradigm of fixing a subset of the variables of a given mixed integer program in order to obtain subproblems that are easier to solve has proven successful in many primal MIP heuristics, see Chapters 6, 7, and [DRP04, Gho07, Ber14]. The core difficulty in MIP solving is the presence of integrality constraints. Thus, in a MIP context, “easy to solve” usually means that there are few integer variables.

Actually, integrality is a special case of nonconvex nonlinearity, since it is possible to model the integrality of a bounded integer variable $x_i \in \{l_i, \dots, u_i\}$ by the nonconvex polynomial constraint $(x_i - u_i)(x_i - u_i + 1) \cdots (x_i - l_i) = 0$. This insight matches the practical experience that in MINLP, while integralities do contribute to the complexity of the problem, the specific difficulty is the presence of nonlinearities. Hence, “easy” in an MINLP context can be understood as having few nonlinear constraints.

Our heuristic is based on the simple observation that by fixing certain variables (to some value within their bounds) any given mixed integer *nonlinear* program can be reduced to a mixed integer *linear* subproblem (sub-MIP). Every feasible solution of this sub-MIP is then a feasible solution of the original MINLP.

Whereas in general it holds that many or even all of the variables might need to be fixed in order to arrive at a linear subproblem, our approach is motivated by the experience that for several practically relevant MINLPs, fixing only a comparatively small subset of the variables suffices to obtain a sub-MIP. The computational effort of solving this subproblem compared to solving the original problem, however, is usually greatly reduced since we can apply the full strength of state-of-the-art MIP solving. Before formulating a first generic algorithm for our heuristic, consider the following definition.

Definition 8.1. *Let P be an MINLP of form* (2.1) *and $\mathcal{C} \subseteq \mathcal{N}$ be a set of variable indices of P. We call $\mathcal{C}$ a* cover *of function g_k, $k \in \mathcal{M}$, if and only if for all $\bar{x} \in [l,u]$ the set*

$$\{(x, g_k(x)) : x \in [l,u], x_i = \bar{x}_i \text{ for all } i \in \mathcal{C}\} \tag{8.1}$$

is an affine set intersected with $[l,u] \times \mathbb{R}$. We call $\mathcal{C}$ a cover *of P if and only if $\mathcal{C}$ is a cover of all constraint functions $g_1, \dots, g_m$.*

Trivial examples of covers are the set of all variables or the set of all variables appearing in nonlinear terms. Many instances of practical interest, however, allow for significantly smaller covers, as we will show in Section 8.6.

Note that Definition 8.1 refers to the functions g_k in the problem formulation and not to the feasible set of an MINLP directly. It does not exploit, for example, that some of the functions might be redundant. This is motivated by taking the perspective of a solver, which initially obtains a problem as a list of individual constraints, and is a common practice in MINLP. Compare, for instance, how a *convex* MINLP is typically understood as all g_k being convex rather than the feasible region being convex because only the first ensures the general validity of gradient cuts.

The following example illustrates how covers of an MINLP are used to construct a sub-MIP for finding feasible solutions.

Example 8.2 (the Undercover sub-MIP)**.** *Consider the following convex MIQCP:*

$$\begin{aligned} \min \quad & -x_2 - x_3 \\ \text{s.t.} \quad & x_1 + x_2 + x_3^2 - 4 \leqslant 0, \\ & x_1, x_2, x_3 \geqslant 0, \\ & x_1, x_2 \in \mathbb{Z}. \end{aligned} \tag{8.2}$$

The only variable that appears in a nonlinear term is x_3, hence $\{3\}$ is a (smallest) cover of (8.2) w.r.t. the above definition. The (unique) optimal

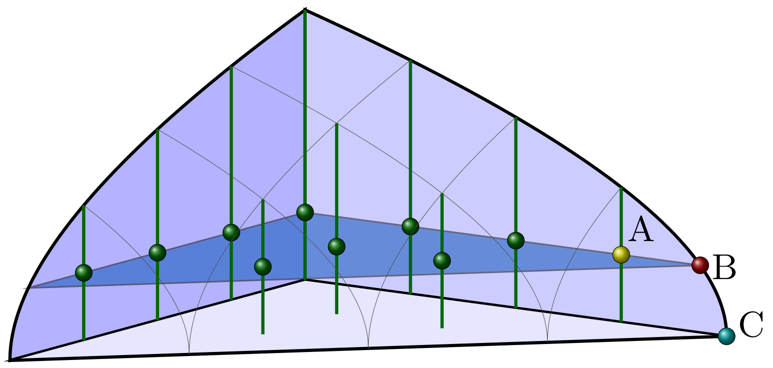


Figure 8.1.: A convex MIQCP and the Undercover sub-MIP induced by the NLP relaxation.

solution of its nonlinear relaxation is $(0, 3.75, 0.5)$ *with objective function value* -4.25*.*

Taking that relaxation, the idea of Undercover is to fix x_3 *to* 0.5*, which renders (8.2) an integer linear program. The (unique) optimal solution of this is* $(0, 3, 0.5)$ *with an objective function value of* -3.5*, which is necessarily a feasible solution for the MIQCP (8.2). Taking the NLP as dual and the Undercover solution as primal bound, this gives an optimality gap of roughly* 20%*. The actual (unique) optimal solution of (8.2) is* $(0, 4, 0)$*.*

This example is illustrated in Figure 8.1. The lightly shaded region shows the solid corresponding to the NLP relaxation; the parallel lines show the mixed integer set of feasible solutions of (8.2). The darkly shaded area shows the polytope associated with the Undercover sub-MIP. The red point B is the optimum of the NLP, the yellow point A is the optimum of the Undercover sub-MIP, the light blue point C is the optimum of the MIQCP. The green points indicate further feasible solutions of the Undercover sub-MIP.

A first generic algorithm for our heuristic is given in Figure 8.2. The crucial step of the algorithm is found in line 4: finding a suitable cover of the given MINLP. Section 8.3 elaborates on this aspect of the algorithm in detail.

To obtain suitable fixing values for the selected variables, an approximation or relaxation of the original MINLP is used. For integer variables, the approximate values are rounded. Most complete solvers for MINLP are based on branch-and-bound [LD60]. If the heuristic is embedded within a branch-and-bound solver, using its (linear or nonlinear) relaxation appears as a natural choice for obtaining approximate variable values.

Large neighborhood search heuristics that rely on fixing variables typically have to trade off between eliminating many variables in order to make the sub-MIP tractable and leaving enough degrees of freedom such that the sub-MIP is still feasible and contains good solutions. Often their implementation

Figure 8.2.: Generic Undercover algorithm

```
Input   : MINLP $P$ as in (2.1)
Output: feasible solution $\tilde{x}$ (on success)
1 begin
2     compute a solution $\bar{x}$ of an approximation or relaxation of $P$;
3     round $\bar{x}_i$ for $i \in \mathcal{I}$;
4     determine a cover $\mathcal{C}$ of $P$;
5     solve the sub-MIP of $P$ given by fixing $x_i = \bar{x}_i$ for all $i \in \mathcal{C}$;
```

inside a MIP solver demands that a sufficiently large percentage of variables be fixed to arrive at an easy to solve sub-MIP, see Chapters 6, 7, and [Ber06, Ber14, DRP04, Gho07].

For our heuristic, the situation is different since we do not aim to eliminate integrality constraints, but nonlinearities. While it still holds that fixing variables, even only few, results in a smaller search space, the main benefit is that we arrive at a MIP.

In a nutshell: instead of solving an easier problem of the same class, we solve a smaller problem of an easier class.

In order to linearize a given MINLP, in general we may be forced to fix integer and continuous variables. The fixing of continuous variables, especially, in an MINLP can introduce a significant restriction, even rendering the subproblem infeasible. Thus our heuristic will aim at fixing as *few* variables as possible to obtain as *large* a linear subproblem as possible, through the utilization of minimum covers.

8.3. Finding minimum covers

Minimum covers, i.e., minimal subsets of variables to fix in order to linearize each constraint of an MINLP, can be computed by solving vertex covering problems. To show the connection between vertex covers and covers of an MINLP, we consider the following definition, which is a generalization of Hansen and Jaumard's notion of a *co-occurrence graph* for quadratically constrained quadratic programs [HJ92].

Definition 8.3 (co-occurrence graph)**.** *Let P be an MINLP of form* (2.1) *with $g_1, \dots, g_m$ twice continuously differentiable on the interior of $[l, u]$. We call $G_P = (V, E)$ the* co-occurrence graph *of P with node set $V_P = \mathcal{N}$ given by the variable indices of P and edge set*

$$E_P = \{(i,j) \mid i,j \in V, \exists k \in \mathcal{M} : \frac{\partial^2}{\partial x_i \partial x_j} g_k(x) \not\equiv 0\},$$

i.e., an edge connects nodes i and j if and only if the Hessian matrix of some constraint has a structurally nonzero entry (i, j).

Note that Definition 8.3 relies on the standard assumption that the nonlinear functions involved are twice continuously differentiable, whereas this is not required for Definition 8.1.

Remark 8.4. *Since the Hessian of a twice continuously differentiable function is symmetric, G_P is a well-defined, undirected graph. It may contain loops, e.g., if square terms x_i^2 are present. Trivially, the co-occurrence graph of a bilinear program is bipartite; the co-occurrence graph of a MIP is an edge-free graph.*

The Undercover algorithm rests on the observation that for an MINLP a set $\mathcal{C} \subseteq \mathcal{N}$ is a cover of P if and only if it is a vertex cover of the co-occurrence graph G_P.

Note that any undirected graph $G = (V, E)$ is the co-occurrence graph of the QCP $\min\{0 : x_i x_j \leqslant 0 \text{ for all } (i,j) \in E\}$. Hence, the problem of computing a minimum vertex cover problem can be transformed to computing a minimum cover of an MINLP. Since minimum vertex cover is $\mathcal{NP}$-hard [GJ79], it follows directly that computing a minimum cover of an MINLP is $\mathcal{NP}$-hard, even when restricted to quadratic constraints.

Nevertheless, we aim at computing minimum covers exactly. To do so, we decided to model the covering problem as a simple binary programming formulation. For an MINLP of form (2.1), define auxiliary binary variables α_i, $i \in \mathcal{N}$, equal to 1 if and only if the original variable x_i is fixed. Then

$$\mathcal{C}(\alpha) := \{i \in \mathcal{N} : \alpha_i = 1\}$$

forms a cover of P if and only if $\alpha_i + \alpha_j \geqslant 1$ for all $(i,j) \in E_P$. For an MIQCP, for example, this requires all square terms and at least one variable in each bilinear term to be fixed.

Definition 8.5 (covering problem)**.** *Let a general MINLP in form* (2.1) *with co-occurrence graph $(\mathcal{N}, E_P)$ and auxiliary variables α_i, $i \in \mathcal{N}$, be given. We call the binary program*

$$\min\Big\{\sum_{i=1}^{n} \alpha_i : \alpha_i + \alpha_j \geqslant 1 \text{ for all } (i,j) \in E_P, \alpha \in \{0,1\}^n\Big\} \tag{8.3}$$

the covering problem of (2.1).

Problem (8.3) is solved by the Undercover algorithm to obtain a set of fixing variables that yield as large a linear subproblem as possible.

In our experiments, the binary program (8.3) could always be solved by a standard MIP solver within a fraction of a second. In all cases, optimality was proven at the root node, hence without enumeration, despite the problem being $\mathcal{NP}$-hard in general, as argued above.

8.4. Fix-and-propagate and conflict learning

Fixing a variable can have a great impact on the original problem and the approximation we use. An important detail, that is crucial for the success rate of Undercover, is not to fix the variables in the cover simultaneously, but sequentially one by one. This section describes how we use domain propagation, backtracking, and conflict analysis to avoid and handle infeasibilities during this process.

Fix-and-propagate

The task of *domain propagation* is to analyze the structure of individual constraints w.r.t. the current domains of the variables in order to infer additional domain reductions, thereby tightening the search space. For an overview of domain propagation techniques applied in MIP and MINLP solvers, see [Ach09] and [Vig12], respectively. A brief summary of domain propagation techniques for MIP is also given in Section 4.3.

To prevent obvious infeasibilities, we fix the variables in the cover one after the other and apply domain propagation after each fixing in order to further tighten the bounds, in particular those of the yet unfixed cover variables. During this process, it might happen that the value a variable takes in the reference solution is no longer contained in its reduced domain. In this case, we fix the variable to the closest bound instead.[30] This fix-and-propagate procedure resembles a method described by Fischetti and Salvagnin [FS09]. Additionally, we apply it for continuous variables.

In the above scheme, the fixed values of variables depend on the fixing order. Different variable orderings may lead to different propagations, thereby to different subproblems and different solutions being found.

Of course, it might also happen that a variable domain becomes empty. Then the subproblem with the currently chosen fixing values is proven to be infeasible without even having started its solution procedure.

In this case, we apply a one-level backtracking, i.e., we undo the last bound change and try alternative fixing values, see Section 8.5 for details. Note that if we cannot resolve the infeasibility by one-level backtracking, Undercover will terminate. This is a "fail fast" strategy (see Chapter 4.2): if we cannot easily resolve the infeasibility, we abort at an early stage of the algorithm without wasting running time.[31]

Even if fix-and-propagate runs into an infeasibility, we can extract useful

[30]Alternatively, we could recompute the reference solution to obtain values within the current bounds.

[31]If we want to apply Undercover more aggressively, we can try to recover from infeasibility by reordering the fixing sequence, e.g., such that the variable for which the fixing failed will be the first one in the reordered sequence. This is a simple version of a restarting mechanism. Restarting techniques are commonly used in solving SAT problems [MMZ+01].

information for the global solution process. Adding so-called conflict constraints prevents us from reaching the same deadlock again.

Conflict analysis in MIP

Conflict learning is a technique that analyzes infeasible subproblems encountered during a branch-and-bound search. Whenever a subproblem is infeasible, conflict analysis can be used to learn one (or more) reasons for this infeasibility. This gives rise to so called conflict constraints that can be exploited in the remainder of the search to prune other parts of the tree.

Carefully engineered conflict analysis has led to a substantial increase in the size of problems modern SAT solvers can solve [MMZ$^+$01]. It has recently been generalized to MIP [Ach07a, Ach07b]. One main difference between MIP and SAT solving in the context of conflict analysis is that the variables of a MIP do not need to be of binary type. Achterberg [Ach07a] has shown how the concept of a conflict graph can be extended to MIPs with general integer and continuous variables.

The most successful SAT learning approaches use so-called *first unique implication point (1UIP)* learning, which captures a conflict that is “close” to the infeasibility and can infer new information. Solvers for MIP or MINLP typically have a longer processing time per node compared to SAT or CP solvers and they do not restart during search. As a consequence, the overhead for further exploring the conflict graph is often negligible compared to the potential savings. That is why MIP solvers with conflict learning such as SCIP potentially generate several conflicts for each infeasibility. See also Section 9.2 for a more detailed comparison of conflict analysis in CP and MIP and a description of its implementation in SCIP.

Conflict analysis for Undercover

The fix-and-propagate strategy can be seen as a simulation of a depth-first search in the branch-and-bound tree, applying one-level backtracking when a fixing results in an infeasible subproblem. Hence, using conflict analysis for these partially fixed, infeasible subproblems enables us to learn constraints that are valid for the global search of the original MINLP. This is done by building up the conflict graph that is implied by the variable fixings and the propagated bound changes. Therefore, the reason for each propagation, i.e., the bounds of other variables that implied the domain reduction, needs to be stored or reconstructed later on.

Note that the generated conflict constraints will not be limited to the variables in the cover since the conflict graph also contains all variables that have changed their bounds due to domain propagation in the fix-and-propagate procedure.

Valid constraints can be learned even after fix-and-propagate. If the subsequent sub-MIP solution process proves infeasibility and all variables in the cover are integer, we may forbid the assignment made to the cover variables for the global solution process. The same constraint can be learned if the Undercover sub-MIP can be solved to proven optimality, since the search space that is implied by these fixings has been fully explored. In both cases, this is particularly useful for small covers.

8.5. The complete algorithm

This section outlines the details of the complete Undercover algorithm, see Figure 8.3. In the first step, we construct the covering problem (8.3) by collecting the edges of the co-occurrence graph, see Section 8.3. For constraints of simple form such as quadratic ones, the sparsity pattern of the Hessian matrix can be read directly from the description of the constraint function. For general nonlinearities, we use *algorithmic differentiation* to automatically compute the sparsity pattern of the Hessian (see, e.g., Griewank and Walther [GW08]).

This assumes that the computational graphs of the constraint functions are readily available and not given implicitly by an oracle. Furthermore, the sparsity pattern computed by an algorithmic differentiation code depends on the formulation of the function. Consider, for example, the linear expression $x^3 + 3x^2 - (x+1)^3$ for which an algorithmic differentiation code might return unnecessary structural nonzeros in the Hessian. Therefore, the constraint functions should be reformulated and simplified in advance. In our implementation, this happens during the preprocessing phase of SCIP. Although SCIP's preprocessing does not explicitly take cover sizes into account, it helps to avoid simple cases of redundant variables in the cover. For instance, it replaces square terms of binary variables by the binary variable itself.

We solve the covering problem as a MIP. This is in the spirit of the *MIP-ping* [FLS10] approach, that propagates to transfer crucial decisions during the solution process to (partially) solving auxiliary MIPs. In our computational experiments, SCIP never took more than a fraction of a second to find an optimal cover. Nevertheless, since the covering problem is $\mathcal{NP}$-hard, solving it to optimality may be time-consuming, in general. To safeguard against this, we only solve the root node and proceed with the best solution found. This is valid since for covering problems SCIP will always find an incumbent solution during root node processing: the Trivial and the 1-Opt heuristic together serve as a greedy algorithm, see also Section 4.2. Subsequently, we fix the variables in the computed cover as described in Section 8.4.[32]

[32]If we want to apply Undercover aggressively and allow for solving the covering problem multiple times, the following two strategies can be used. First, during the fix-and-propagate routine, variables outside the precomputed cover may be fixed simultaneously. In this case, the fixing of some of the yet unfixed variables in the cover might become redundant. Recomputing the cover with $\alpha_i = 1$ for all i with local bounds $\hat{l}_i = \hat{u}_i$ may yield a smaller number of remaining variable fixings. Second, if no feasible fixings for

Figure 8.3.: Complete Undercover algorithm

```
Input   : MINLP as in (2.1), reference point $\bar{x} \in [l,u]$,
          $n_i \geqslant 0$ alternative fixing values $y^*_{i,1},\dots,y^*_{i,n_i} \in [l_i,u_i]$ for all $i \in \mathcal{N}$
Output  : feasible solution $\tilde{x}$ (on success)
/* Step 1: create covering problem */
1  $E \leftarrow \emptyset$;                                   /* edge set of co-occurrence graph */
2  foreach $k \in \mathcal{M}$ do
3      $S_k \leftarrow \{i \in \mathcal{N} : g_k$ depends on $x_i\}$;         /* variables in $g_k(x) \leqslant 0$ */
4      foreach $i \in S_k$ do
5          if $\frac{\partial^2}{\partial x_i^2} g_k(x) \not\equiv 0$ then $E \leftarrow E \cup \{(i,i)\}$;      /* have to fix $x_i$ */
6          else
7              foreach $j \in S_k, j > i, \frac{\partial^2}{\partial x_i \partial x_j} g_k(x) \not\equiv 0$ do
8                  $E \leftarrow E \cup \{(i,j)\}$;                  /* have to fix $x_i$ or $x_j$ */
/* Step 2: solve covering problem (8.3) */
9  $\alpha^* \leftarrow \arg\min \left\{ \sum_{i=1}^n \alpha_i : \alpha_i + \alpha_j \geqslant 1 \text{ for all } (i,j) \in E, \alpha \in \{0,1\}^n \right\}$;
10 $\mathcal{C} \leftarrow \{i \in \mathcal{N} : \alpha^*_i = 1\}$;
/* Step 3: fix-and-propagate loop */
11 $\hat{l} \leftarrow l$, $\hat{u} \leftarrow u$;                              /* local bounds */
12 foreach $i \in \mathcal{C}$ do
13     $\hat{l}^0 \leftarrow \hat{l}$, $\hat{u}^0 \leftarrow \hat{u}$, $p \leftarrow 0$;            /* store bounds for backtracking */
14     $\mathcal{X} \leftarrow \emptyset$, $success \leftarrow$ false;             /* set of failed fixing values */
15     while $\neg success$ and $p \leqslant n_i$ do
16         if $p = 0$ then $X_i \leftarrow \bar{x}_i$;
17         else $X_i \leftarrow y^*_{i,p}$;
18         if $i \in \mathcal{I}$ then $X_i \leftarrow [X_i]$;                /* round if variable integer */
19         $X_i \leftarrow \min\{\max\{X_i, \hat{l}_i\}, \hat{u}_i\}$;          /* project to bounds if outside */
20         if $X_i \in \mathcal{X}$ then
21             $p \leftarrow p + 1$;                        /* skip fixing values tried before */
22         else
23             $\hat{l}_i \leftarrow X_i$, $\hat{u}_i \leftarrow X_i$;                                   /* fix */
24             call domain propagation on $[\hat{l},\hat{u}]$;                         /* propagate */
25             if $[\hat{l},\hat{u}] \neq \emptyset$ then
26                 $success \leftarrow$ true;            /* accept fixing, go to next variable */
27             else
28                 call conflict analysis;
29                 $\hat{l} \leftarrow \hat{l}^0$, $\hat{u} \leftarrow \hat{u}^0$;                  /* infeasible: backtrack */
30                 $\mathcal{X} \leftarrow \mathcal{X} \cup \{X_i\}$, $p \leftarrow p + 1$;          /* try next fixing value */
31     if $\neg success$ then return;          /* no feasible fixing found: terminate */
/* Step 4: solve sub-MIP */
32 solve sub-MIP $\min \left\{ c^\mathsf{T} x : g_k(x) \leqslant 0 \text{ for all } k \in \mathcal{M},\ x \in [\hat{l},\hat{u}],\ x_i \in \mathbb{Z} \text{ for all } i \in \mathcal{I} \right\}$;
33 if sub-MIP solved to optimality or proven infeasible and $\mathcal{C} \subseteq \mathcal{I}$ then
34     add conflict constraint $\bigvee_{i \in \mathcal{C}} (x_i \neq X_i)$ to original problem;
/* Step 5: solve sub-NLP */
35 if feasible sub-MIP solution found then
36     $\tilde{x} \leftarrow$ best sub-MIP solution;
37     if sub-MIP not solved to optimality or $\mathcal{C} \nsubseteq \mathcal{I}$ then
           /* restore global bounds, fix integers, solve locally */
38         solve $\min \left\{ c^\mathsf{T} x : g_k(x) \leqslant 0 \text{ for all } k \in \mathcal{M},\ x \in [l,u],\ x_i = \tilde{x}_i \text{ for all } i \in \mathcal{I} \right\}$;
39         $\tilde{x} \leftarrow$ sub-NLP solution;                  /* update sub-MIP solution */
40     return $\tilde{x}$;
```

As motivated in the beginning of this chapter, we designed Undercover to be applied within a complete solver. During fix-and-propagate, we call two routines provided by the solver: domain propagation in line 24 and conflict analysis in line 28. If the former detects infeasibility, we call the latter to learn conflict constraints for the global solution process, see Section 8.4.

If domain propagation detects infeasibility after fixing variable x_i, $i \in \mathcal{C}$, to the (rounded and projected) value X_i in the reference solution, we try to recover by one-level backtracking. The following alternatives will be tried: for binary variables the value $1-X_i$; for nonbinary variables the lower bound l_i and, if this is also infeasible, the upper bound u_i. In the case of infinite bounds, l_i and u_i are replaced by $X_i - |X_i|$ and $X_i + |X_i|$, respectively. If $X_i = 0$, then -1 and $+1$ will be used instead. Of course, if fixing values accidentally coincide, each value is tested only once.

Typically, the sub-MIP solved in the next step incurs the highest computational effort and is controlled by work limits on the number of nodes, LP iterations, etc. (see Section 8.6 for details). Since by construction the sub-MIP should be significantly easier than the original MINLP, we expect that it can often be solved to optimality or proven infeasible. As described in Section 8.4, we may then forbid the assignment of fixing values to the cover variables if the latter are all integer, as stated in line 34. Note that the two learning steps described in lines 28 and 34 are only relevant for the overall solution process of the complete solver within which Undercover is called. They do not alter the behavior of the Undercover algorithm itself.

If Undercover finds a feasible sub-MIP solution $\tilde{x}$, we try to improve it further by fixing all integer variables to their values in $\tilde{x}$ and solving the resulting NLP to local optimality. Clearly, if all cover variables are integer and $\tilde{x}$ is optimal for the sub-MIP, this step can be skipped. Otherwise, we reoptimize over the continuous variables in the cover and may obtain a better objective value.

For several design decisions, we considered different alternatives which have been ruled out in preliminary experiments. We tried using alternative objectives for the covering problem, an NLP solution instead of an LP solution for the fixing values, different variable orders in the fix-and-propagate loop, and so on. Details can be found in Section 8.7. These variants either altered the performance only slightly or they were inferior in the sense that they failed on a significant number of instances for which the default settings succeeded but did not typically succeed on any instance for which the default failed. All these choices have been made user parameters of SCIP.

Note that at most four optimization problems have to be solved for running the algorithm given in Figure 8.3: one for computing the reference solution $\bar{x}$ as input, one for solving the covering problem in line 10, one for solving

the cover variables in $\mathcal{C}$ are found, we can solve the covering problem again with an additional cutoff constraint $\sum_{i\in\mathcal{C}}(1-\alpha_i) + \sum_{i\notin\mathcal{C}} \alpha_i \geqslant 1$ and try once more. However, both techniques appear to be computationally too expensive for the standard setting that we explored in our computational experiments.

the sub-MIP in line 32 and possibly one for polishing the solution by solving a sub-NLP in line 38.

8.6. Computational experiments

This section describes the experimental setup and computational results for using Undercover as a root node start heuristic. We conducted these experiments on two test sets of MIQCPs and MINLPs which have been compiled from publicly available libraries.

We implemented the algorithm given in Figure 8.3 within SCIP and used SCIP's LP solution as reference point $\bar{x}$. To perform the fix-and-propagate procedure, we called the standard domain propagation engine of SCIP. Secondary SCIP instances were used to solve both the covering problem (8.3) and the Undercover sub-MIP.

Experimental setup

The goal of our computational experiments was to analyze the performance of Undercover as a start heuristic for MINLPs, applied at the root node. To this end, we benchmarked against state-of-the-art solvers and measured its impact on the overall performance of an MINLP solver. We evaluated the sizes of the actual covers found, the success rate of Undercover, and the distribution of running time among different components of the algorithm.

We controlled the computational effort for solving the sub-MIP in two ways. First, we imposed a hard limit of 500 nodes and a dynamic stall node limit between 1 and 500 nodes. With a stall node limit, we terminate if no improving solutions are found within a certain number of branch-and-bound nodes since the discovery of the current incumbent. Second, we adjusted the SCIP settings to find feasible solutions fast: we disabled time-consuming presolving techniques and used the "primal heuristics emphasis aggressive" and the "emphasis feasibility" settings. Furthermore, if the sub-MIP is infeasible, this is often detected already when solving the root relaxation, hence we deactivated expensive pre-root heuristics so as to not lose time on such instances. Components using sub-MIPs themselves are switched off altogether. For details, please refer to the source code at [Sci].

We performed two main experiments to evaluate the Undercover algorithm. In order to investigate how Undercover can enhance the root node performance of complete solvers, we compare UC with the root heuristics of four different MINLP solvers. SCIP, for instance, applies eleven primal heuristics at the root node: three rounding heuristics, three propagation heuristics, a trivial one, a feasibility pump, a local search, a repair heuristic and an improvement heuristic. Our second experiment measures the impact of using Undercover as a subroutine inside a complete solver on the overall performance.

In our first experiment, we ran SCIP with all heuristics other than Undercover switched off and cut generation deactivated. We used SCIP 2.1.1 with CPLEX 12.3 [IBM] as LP solver, IPOPT 3.10 [WB06] as NLP solver for the postprocessing, and CPPAD 20100101.4 [Cpp] as expression interpreter for evaluating general nonlinear constraints. We refer to this Undercover standalone configuration as UC.

We tested against three state-of-the-art solvers for nonconvex MINLPs: BARON 9.3.1 [Sah96, TS04], COUENNE 0.3 [BLL$^+$09], and SCIP 2.1.1 with Undercover disabled. Additionally, we included BONMIN 1.6 [Bon], which is a solver for convex MINLP, but can be used as a heuristic for nonconvex problems. All solvers were run in their default configuration. In particular, the algorithm "B-BB" was used for BONMIN. We compare the primal bound obtained after the solution of the root node. Therefore, all solvers, including UC, were started with a node limit of one. We further imposed a large time limit of six hours to enforce termination and a memory limit of 8 GB.

Our test set is based on the 172 MIQCPs from the test suite of Misener and Floudas [MF13, Glo], a broad selection of publicly available MIQCP and QCP instances. From this test set we removed all instances for which we knew a-priori that Undercover would never be called. These were seven very easy instances, mainly of the `st_test` type, that are solved by SCIP presolving or the solution of the root LP, i.e., before Undercover would be executed, and eleven instances that have an unbounded root LP, all of the `nuclear` type. For the first experiment, we further excluded three of the `LeeCrudeOil` instances for which BONMIN did not terminate within nine hours (given a time limit of six hours). This left 147 instances.

We further tested Undercover on general MINLPs from MINLPLIB, excluding those which are MIQCPs, linear after SCIP presolving, or contain expressions that cannot be handled by SCIP, e.g., sin and cos. Additionally, three more instances with unbounded root LP relaxation were removed, leaving 110 instances. We used the same settings and solvers as described above.

Our second experiment analyzes the impact of Undercover on the overall solution process of a complete solver. Therefore, we ran SCIP in its default settings, with and without Undercover, using a time limit of one hour and a memory limit of 40 GB. For this test, we included the three `LeeCrudeOil` instances, and excluded `ruiz_flowbased_pw4`, for which SCIP terminates with an error. This gives a test set of 149 instances.

The results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20GHz with 12 MB cache and 48 GB main memory, running an openSUSE 11.4 with a GCC 4.5.1 compiler. Hyperthreading and Turboboost were disabled. For the latter experiment, we ran only one job per node to avoid random noise in the measured running time that might be caused by delays if multiple processes share common resources, in particular the memory bus.

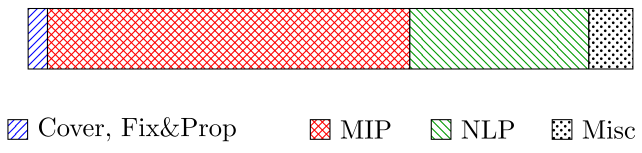


Figure 8.4.: Distribution of running time among different components of Undercover heuristic.

Results for MIQCP

The results for the experiments on MIQCPs are shown in Table B.16 in the appendix. In Columns % cov and % nlcov, we report the relative size of the cover used by UC as a percentage of the total number of variables and of the number of variables that appear in at least one nonlinear term, respectively. A value of 100 % in the % nlcov column means that the trivial cover consisting of all variables appearing in nonlinear terms is the minimum cover. For all other instances, the solution of the covering problem gives rise to a smaller cover, hence a larger sub-MIP and potentially more solutions for the MINLP. All numbers are calculated w.r.t. the numbers of variables after preprocessing.

Column UC shows the objective value of the best solution found by Undercover. For all other solvers, we provide the objective value of the best solution found during root node processing.

The computational results for MIQCPs seem to confirm our expectation that a low fixing rate often suffices to obtain a linear subproblem: 25 of the instances in our test set allow a cover of at most 5 % of the variables, a further 40 instances of at most 10 % and 46 instances of at most 25 %. Eighteen instances were in a medium range of 25 % to 50 %; for another 18, a minimum cover contained more than half of the variables.

UC found a feasible solution for 76 test instances. Interestingly, it worked similarly well with small and large covers. For 15 out of 25 instances with a cover of at most 5 % of the variables, UC found a solution, but also for 30 out of the 36 instances with a cover of at least 25 %. In comparison, BARON found a feasible solution in 65 cases, Couenne in 55, SCIP and Bonmin in 98 each.

There were 32 instances for which UC found a better solution than BARON, 20 for which it was better than SCIP, 36 for Couenne, and 32 for Bonmin. We note that for six instances UC found the single best solution compared to *all* other solvers and for 27 further instances it produced the same solution quality as the best of the other solvers.

Out of 147 instances, the time for applying Undercover was less than 0.1 seconds in 131 cases, 14 times it was between 0.1 and 0.5 seconds; the two outliers are `waste` (1.31 seconds) and `Sarawak_Scenario81` (2.53 seconds). Figure 8.4 shows the average distribution of running time for solving the covering problem, processing the fix-and-propagate loop, solving the sub-MIP, polishing the solution with an NLP solver and for the remaining parts such as allocating and freeing data structures, constructing the auxiliary problems, computing conflict constraints, and so on. This average has been taken over all instances for which Undercover found a feasible solution, hence all main parts of the algorithm were executed. The major amount of time, namely 65 %, is spent in solving the sub-MIP. Solving the covering problem plus performing fix-and-propagate took only about 3 % of the actual running time.

Although the polytope described by the constraints of BP (8.3) is not integral, the covering problem could always be solved to optimality at the root node by SCIP's default heuristics and cutting plane algorithms. In 89 out of 147 cases, the minimum cover was nontrivial, with cover sizes of 8 % to 60 % of the nonlinear variables.

We note that in 55 out of the 70 cases for which the resulting sub-MIP was infeasible, the infeasibility was detected during the fix-and-propagate stage and in ten of the remaining fifteen cases during root node processing of the sub-MIP.[33] Thus in most cases, no time was wasted trying to find a solution for an infeasible subproblem, since the most time-consuming part (see Figure 8.4) can be skipped. This confirms that Undercover follows a "fast fail" strategy, a beneficial property of primal heuristics applied within complete solvers, as argued in Chapter 4.2. Also, all except one feasible sub-MIP could be solved to optimality within the imposed node limit of 500. This indicates that – with a state-of-the-art MIP solver at hand – the generated subproblems are indeed significantly easier than the full MIQCP, as can also be seen when compared to the running times and the number of nodes in Table B.17, see below.

For 31 out of 76 successful runs, all cover variables were integral. For the remaining 35 instances, NLP postprocessing was applied; 21 times, it further improved the Undercover solution.

Recall that an arbitrary point $\bar{x} \in [l, u]$ can serve as a reference solution for Undercover. A natural alternative to the LP solution is a (locally) optimal solution of the NLP relaxation. An additional experiment showed that, using an NLP solution, Undercover only succeeded in finding a feasible solution for 52 instances of the MIQCP test set, instead of 76. If both versions found a solution, the quality of the one based on the NLP solution was better in 23 cases, worse in eleven. Our interpretation for the lower success rate is that the advantage of the NLP solution, namely being feasible for all nonlinear constraints, is dominated by the fact that an NLP solution typically has a higher fractionality, which leads to a higher chance that infeasibility is introduced in line 18 of the Undercover algorithm in Figure 8.3. We also tried using an NLP relaxation for those eleven instances that were excluded because of an unbounded root LP: in no case was a feasible solution found.

[33]For one instance, the feasibility status had not been decided within 500 nodes.

Results for MINLP

As expected, Undercover is much less powerful for general MINLPs compared to MIQCPs. UC produced feasible solutions for only six out of more than a hundred test problems from MinlpLib: `mbtd`, `nvs09`, `nvs20`, `stockcycle`, `synthes1`, and `johnall`. During root node processing, baron found feasible solutions for 39 instances, Couenne for 23, SCIP for 35. Although it is clearly outperformed by the other solvers w.r.t. the number of solutions found, we would like to mention that for each other solver there is at least one instance for which UC succeeded, but the solver did not.

Nevertheless, the experiments showed that fixing a small fraction of the variables would often have sufficed to obtain a linear subproblem: for 77 out of the 110 test instances, the minimum cover contained at most 25 % of the variables, similar to the MIQCP case, but only five MINLPs allowed for a cover size below 5 %. The extreme values were 0.18 % for `mbtd` and 96.97 % for `nvs20`.

Hence, compared to the MIQCP test set, cover sizes are on average larger and very small covers occur rarely, but this alone does not explain the lower success rate. It simply appears to be more difficult to find feasible fixing values due to the higher complexity of the nonlinear constraints, even if we use the solution of an NLP relaxation as the reference point $\bar{x}$. Surprisingly, Undercover produced feasible solutions for the two instances with the smallest and the two instances with the largest minimum covers.

Undercover inside a complete solver

The previous experiments showed that for general MIQCP instances, Undercover nicely complements the existing root node heuristics of baron, Bonmin, Couenne and SCIP. The question remains as to whether this is beneficial for the overall solution process.

For this experiment, interactions of different primal heuristics with each other and with other solver components come into play. Obviously, a primal heuristic called prior to Undercover might already have found a solution which is better than the optimal solution of the Undercover sub-MIP, or in an extreme case, the solution process might have terminated before Undercover is called. Further, any solution found before Undercover is called might change the solution path. It might trigger variable fixings by dual reductions, which lead to a different LP and hence to a different initial situation for Undercover. Because of this, Undercover might succeed for problems where it failed in the first experiment and vice versa. Note that even in the case of failure, Undercover might produce conflict clauses (see Section 8.4) and thereby be beneficial for the overall solution process.

In this experiment, our main criteria for measuring performance are the running time and the number of branch-and-bound nodes needed to prove optimality. To average values over all instances of the test set, we use a the shifted geometric mean, see Definition 2.12.

Table 8.1.: Comparison of SCIP with and without Undercover (aggregated results).

	both solved (117)		time > 10 s (31)	
	nodes	time [s]	nodes	time [s]
SCIP − UC	702	10.8	21 097	98.2
SCIP + UC	610	9.4	15 619	74.5
shifted geom. mean	−15 %	−15 %	−35 %	−32 %

The results of running SCIP once with and once without Undercover are shown in Table B.17 in the appendix, and a summary can be found in Table 8.1. In Table B.17, Columns nodes and time [s] show the number of branch-and-bound nodes and the running time SCIP needs to solve an instance to proven optimality, pb root depicts the primal bound at the root node. In Table 8.1, Columns nodes and time [s] refer to the shifted geometric means taken over all instances in the test set.

Both versions of SCIP solved nearly the same set of instances within the given time limit; there was only one instance, `SLay10H`, which needed more than an hour when using SCIP with Undercover, but solved within 35 minutes and about 150 000 nodes otherwise. However, for those 117 instances which could be solved by both variants, the SCIP version that included Undercover needed 15 % fewer nodes and 15 % less running time in shifted geometric mean. If we ignore very easy instances that both versions solved in less than ten seconds, the improvement is even more significant: for the 31 instances which fall into this category, SCIP without Undercover is 32 % slower and needs 35 % more nodes in shifted geometric mean.

Even though the running times for Undercover are moderate (see above) for instances that solve within a fraction of a second, it sometimes consumes a significant amount of the running time. However, when we consider the 33 problems that need between one and ten seconds of running time, on average, Undercover only requires 1.5 % of the overall running time; for the 58 instances that need more than ten seconds the ratio is 0.04 %.

Further experiments

We experimented with the following extensions of Undercover: reordering the fixing sequence if fix-and-propagate fails, see Footnote 31; re-solving the covering problem if the sub-MIP is infeasible, see Footnote 32; using a weighted version of the covering problem, see Section 8.7. None of those performed significantly better than our default strategy.

In a complete solver, primal heuristics are applied in concert, hence a feasible solution may be already at hand when starting Undercover. In our implementation, this is exploited in two ways. First, we use values from the incumbent solution as fixing alternatives during fix-and-propagate. Fixing the variables to values in the incumbent has the advantage that the resulting sub-MIP is guaranteed to be feasible, compare, e.g., Danna et al. [DRP04]. Second, we add a primal cutoff to the sub-MIP to only look for improving solutions.[34] On four instances of the MIQCP test set, SCIP 2.1.1 with default heuristics including Undercover produced a primal solution that was significantly better than the best solution found by either SCIP or Undercover alone; a worse solution was produced only for one instance.

8.7. Variants

We experimented with a few more variants of the Undercover heuristic. Some of them proved beneficial for specific problem classes. For the standard setting presented in our computational results, however, they showed no significant impact. As they might prove useful for future applications of Undercover, we will provide a brief description.

Our initial motivation for using a minimum cardinality cover was to minimize the impact on the original MINLP. Instead of measuring the impact of fixing variables uniformly, we could solve a weighted version of the covering problem (8.3). To better reflect the problem structure, the objective coefficients of the auxiliary variables α_i could be computed from characteristics of the original variables x_i such as the domain size, variable type (integer or continuous), or appearance in nonlinear terms or in constraints violated by the reference solution.

Instead of fixing the variables in a cover, we could also merely reduce their domains to a small neighborhood around the reference solution. Especially for continuous variables this leaves more freedom for the exploration of the subproblem and can lead to better solutions found. Of course, the difficulty of solving the subproblem is increased. Nevertheless, small domains may allow for s sufficiently tight relaxation for an MINLP solver to tackle the subproblem.

The main idea of Undercover is to reduce the computational effort by switching to a problem class that is easier to address. While we have focused on exploring a linear subproblem, for nonconvex MINLPs, convex nonlinear subproblems may provide a larger neighborhood to be searched and still be sufficiently easy to solve.

8.8. Conclusion

In this chapter, we have introduced Undercover, a primal MINLP heuristic exploring large linear subproblems induced by a minimum vertex cover. It differs from other recently proposed MINLP heuristics in that it is neither

[34]A primal cutoff is an upper bound on the objective function that results in branch-and-bound nodes with worse dual bound not being explored.

an extension of an existing MIP heuristic, nor solves an entire sequence of MIPs.

We defined the notion of a minimum cover of an MINLP and proved that it can be computed by solving a vertex covering problem on the co-occurrence graph induced by the sparsity patterns of the Hessians of the nonlinear constraint functions. Although $\mathcal{NP}$-hard, covering problems were solved rapidly in our experiments. Several extensions and algorithmic details have been discussed.

Undercover exploits the fact that small covers correspond to large sub-MIPs. We showed that the majority of MIQCPs from the GloMIQO test set [Glo] and MINLPs from the MINLPLib [BDM03] allow for covers consisting of at most 25 % of the variables.

For MIQCPs, Undercover proved to be a fast start heuristic, that often produces feasible solutions of reasonable quality. The computational results indicate that it nicely complements existing root node heuristics in different solvers.

We further showed that for MIQCPs, applying Undercover at the root node significantly improved the overall performance of SCIP, in particular for hard instances. Undercover is now one of the default heuristics applied in SCIP.

A minimum cover of an MINLP is an abstract structure that can obviously be studied and employed beyond the application that has been presented here. In [BG13], Berthold and Gleixner present a branching strategy that exploits minimum covers to drive the created subproblems towards linearity.

9. Rapid Learning

So far, we have been mainly concerned with primal heuristics, i.e., heuristic methods that aim to find good feasible solutions. This chapter, however, deals with a heuristic algorithm to rapidly learn valid conflict constraints for a given mixed integer (linear) program.

Conflict learning is a technique that analyzes infeasible subproblems encountered during a tree search algorithm. Each subproblem can be identified by its local variable bounds, i.e., by local bound changes that come from branching decisions and domain propagation at the current node and its ancestors. If domain propagation detects infeasibility, conflict learning will traverse this chain of decisions and deductions in a reverse fashion, reconstructing which bound changes led to which other bound changes, and will thereby identify explanations for the infeasibility. If it can be shown that a small subset of the bound changes suffices to prove infeasibility, a so-called conflict constraint is generated that can be exploited in the remainder of the search to prune parts of the tree.

In this chapter, we suggest a heuristic algorithm that searches for valid conflict constraints rather than for primal solutions. Our approach is based on the observation that a CP solver can typically perform a partial search on a few hundred or thousand nodes in a fraction of the time that a MIP solver needs for processing the root node. This is mainly due to the substantial computational effort needed to solve the initial LP relaxation from scratch (and partly due to LP resolves during cutting plane generation and strong branching). The idea of our newly proposed heuristic is to apply a fast CP branch-and-bound search for a few hundred or thousand nodes, generating and collecting valid conflict constraints at the root node of a MIP solver. By this, the MIP solver is already equipped with the valuable information of which bound changes will lead to an infeasibility, and can avoid them by propagating the derived constraints. We refer to this quick search for conflicts as *Rapid Learning*.

This chapter is based on joint work with Peter J. Stuckey and Thibaut Feydy which was conducted during a four month research stay of the author as a NICTA Visiting Researcher at the University of Melbourne. A preliminary version of Rapid Learning for binary programs has been published in the proceedings of CPAIOR2010 [BFS10].

The remainder of this chapter is organized as follows: in Section 9.1, we give a short introduction; in Section 9.2, we recap conflict analysis for mixed integer programming. In Section 9.3, we describe the details of Rapid Learning and discuss its implementation. Section 9.4 provides computational experiments, and in Section 9.5 we give our conclusions.

9.1. Introduction

Conflict analysis techniques have been developed and furthered by the artificial intelligence research community [SS77] and, later, the SAT community [MMZ$^+$01]; they led to a substantial increase in the size of problems modern SAT solvers can handle [MSS99, MMZ$^+$01, ZMMM01]. The most successful SAT learning approaches use so called *one-level first unique implication point (1-UIP)* [ZMMM01] learning which in some sense captures the conflict constraint "closest" to the infeasibility. Conflict analysis also is successfully used in the CP community [JB00, KB05, OSC09] (who typically refer to it as nogood learning) and the MIP world [Ach07a, DBS02, SS06].

Constraint programming and mixed integer programming are two complementary ways of tackling discrete optimization problems. Hybrid combinations of the two approaches have been used for more than a decade [Ach07b, ABE$^+$02, AHY04, BK98, RWH99, YAH10]. As a notable example, the software SCIP [Ach09], which is used for computational experiments throughout this thesis (except Chapter 5), is based on the idea of *constraint integer programming* (CIP) [Ach07b, ABKW08]. CIP is a generalization of MIP that supports the notion of general constraints as in CP. SCIP itself follows the idea of a very low-level integration of CP, SAT, MIP, and MINLP techniques. All involved algorithms operate on a single search tree and share information and statistics through global storage of, e.g., solutions, variable domains, cuts, conflicts, the LP relaxation and so on. This allows for a very close interaction amongst CP and MIP (and other) techniques.

The approach that we take in the present chapter, though implemented within SCIP, is different. We split the search in two phases: an incomplete CP search which feeds the global information storage and a subsequent MIP search which starts from scratch (the tree created during the CP search will be deleted), hopefully profiting from the information, in particular the conflict constraints, that the rapid CP search gathered. The suggested algorithm is most promising for pure BPs and IPs.

9.2. Conflict learning in MIP

Conflict learning has not played a role in the integer programming community until recently (although see [DBS02]). MIP solvers used to rule out a subproblem as soon as infeasibility is detected or bounding can be applied, without caring for the reasons of infeasibility; some solvers still do so.

In 2007, Achterberg [Ach07a] generalized the SAT techniques for infeasibility analysis to MIP. A key ingredient for this is a fast heuristic to derive small conflict constraints from infeasible LPs by constructing a dual ray with minimal nonzero elements. In [Ach07a], it is shown that conflict learning for general mixed integer problems can result in an average speedup of 10 %. Kılınc Karzan et al. [KKNS09] suggest restarting the MIP solver and using a branching rule that selects variables which appear in small conflict constraints for the second run. Achterberg and Berthold [AB09] propose a hybrid branching scheme that incorporates conflict-based SAT and impact-based CP style search heuristics as dynamic tie-breakers. For branching strategies based on conflict information, see also Chapter 10.2 of this thesis.

Conflict constraints can further be derived as "by-products" of primal heuristics (see Section 8.4) when sub-MIPs or auxiliary LPs are solved that only made deductions which would also be valid for the original MIP. The remainder of this section shall serve as a brief review of how the analysis of conflicts that have been detected by domain propagation works in SCIP. We keep close to the description given in [Ach07a, Ach07b] and refer the interested reader to these two references for detailed information.

There are two main differences between MIP and SAT solving in the context of conflict analysis: In MIP, the problem variables are not necessarily binary, and infeasibility is often detected by the LP relaxation rather than by domain propagation. To deal with the first issue, we have to extend the concept of the *conflict graph*. A conflict graph gets constructed whenever infeasibility is detected in a local search node; it represents the logic of how the set of branching decisions led to the detection of infeasibility.

More precisely, the conflict graph is a directed acyclic graph in which the vertices[35] represent bound changes of variables and the arcs (u, v) correspond to bound changes implied by propagation, i.e., the bound change corresponding to v is based (besides others) on the bound change represented by u. In general, a vertex will have multiple ingoing arcs which represent all bound changes that have been used to propagate the corresponding bound change. In addition to the inner vertices which represent the bound changes from domain propagation, the graph features source vertices for the bound changes that correspond to branching decisions and an artificial target vertex representing the infeasibility. Then, each cut that separates the branching decisions from the artificial infeasibility vertex gives rise to a valid conflict constraint. A *unique implication point (UIP)* is an (inner) vertex of the conflict graph which is traversed by all paths from the branching vertices to the conflict vertex. Or, how Zhang et al. [ZMMM01] describe it: "Intuitively, a UIP is the *single* reason that implies the conflict at [the] current decision level." UIPs are natural candidates for finding small cuts in the conflict graph.

For integer programs, conflict constraints can be expressed as so-called *bound disjunction* constraints:

Definition 9.1. *For an IP, let $\mathcal{L} \subseteq \mathcal{I}, \mathcal{U} \subseteq \mathcal{I}$ be index sets of variables, let $\lambda \in \mathbb{Z}^{\mathcal{L}}$ with $l_i \leqslant \lambda_i \leqslant u_i$ for all $i \in \mathcal{L}$, and $\mu \in \mathbb{Z}^{\mathcal{U}}$ with $l_i \leqslant \mu_i \leqslant u_i$ for all*

[35]For disambiguation, we will use the term *vertex* for elements of the conflict graph, as opposed to *nodes* of the search tree.

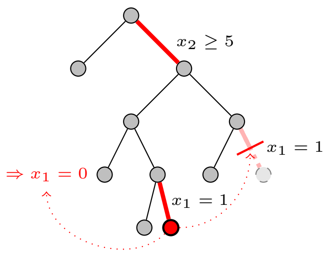


Figure 9.1.: Conflict analysis helps to reduce the search tree. Assume that the subproblem at the thick, red node was detected to be infeasible and the proof of infeasibility could be reduced to the two decisions corresponding to the thick, red edges. Then, $(x_2 \leq 4) \vee (x_1 = 0)$ would be a valid conflict constraint (given that x_1 is binary and x_2 general integer). It can be used to deduce new variable bounds (left dotted arc) and prune nodes that have been created beforehand and violate the conflict constraint (right dotted arc).

$i \in \mathcal{U}$. *Then, a constraint of the form*

$$\bigvee_{i \in \mathcal{L}} (x_i \geq \lambda_i) \vee \bigvee_{i \in \mathcal{U}} (x_i \leq \mu_i)$$

is called a bound disjunction *constraint.*

For details on bound disjunction constraints and for the case of continuous variables, see Achterberg [Ach07a]. An graphical example on how conflict constraints can help to reduce the size of the search tree is given in Figure 9.1.

As before-mentioned, the infeasibility of a subproblem in a MIP search tree often has its reason in the LP relaxation of the subproblem. In this case, there is no single conflict-detecting constraint as in SAT or CP solving. One way of coping with this situation is to analyze the infeasibility certificate of the LP, which is an unbounded ray in the dual LP. The goal is to identify a subset of all bound changes that suffices to render the LP infeasible or bound-exceeding. This is done by searching for a dual ray for which a large number of coefficients that belong to variable bounds are zero. Intuitively, this means that from all possible infeasibility certificates, one is chosen that is primarily derived from problem constraints and requires only few local variable bounds. In SCIP this is done by a greedy heuristic approach[36], see [Ach07a]. Compare also the generation of *irreducible infeasible subsystems* [GM91, APT03].

After having analyzed the LP, the algorithm works in the same fashion as conflict analysis for SAT or CP: it constructs a conflict graph, chooses one or

[36]Here, we witness again the "omnipresence" of heuristic algorithms in MIP and MINLP solver software: The generation of conflict constraints in SCIP is based on a heuristic to adjust the dual ray.

more cuts in this graph which are identified by UIPs, and produces a conflict constraint which consists of the bound changes along the frontier of this cut. However, the Rapid Learning procedure which we introduce in the remainder of this chapter performs a CP-like search: it does not use the LP relaxation. Thereby, all conflicts will be derived from infeasibilities detected by domain propagation.

The power of conflict learning arises because often branch-and-bound based algorithms implicitly repeat the same search in a slightly different context in another part of the tree. Conflict constraints help to handle and exploit such situations and avoid redundant work. As a consequence, the more search is performed by a solver and the earlier conflicts are detected, the greater the chance for conflict learning to be beneficial.

Although the conflict analysis methods of SAT, CP, and MIP approaches are effectively the same, one should note that because of differences in the amount of work per node each solver undertakes there are different design tradeoffs in each implementation. A MIP solver will most likely spend more time processing each node than a CP solver and much more compared to a SAT solver. For that reason SAT and CP systems typically use 1-UIP learning and frequent restarts to tackle problems, while this is not the case for MIP. On the contrary, SCIP only restarts, if at all, at the root node, and potentially generates several conflict constraints for each infeasibility, using *All-UIP* [ZMMM01] by default (see [Ach07a]).

9.3. Rapid Learning for integer programs

The idea of Rapid Learning is based on the fact that a CP solver can typically perform a partial search on a few hundred or thousand nodes in a fraction of the time that an MIP solver needs for processing the root node of the search tree. Rapid Learning applies a fast CP search[37] for a few hundred or thousand nodes, before starting the MIP search. By this, conflict constraints can be gained beforehand, and not only during (the MIP) search. Very loosely speaking: The aim of conflict learning is to avoid making mistakes a second time, whereas Rapid Learning tries to avoid making them the first time.

Rapid Learning is related to the concept of large neighborhood search heuristics, see Chapter 6. But rather than doing a partial search on a sub-problem using the same (MIP search) algorithm, Rapid Learning performs a partial search on the same problem using a much faster algorithm. Rapid Learning also differs from most of the primal heuristics described in this thesis in that it merely aims at improving the dual bound by collecting information on infeasibility rather than searching for feasible solutions.

Each piece of information collected in a rapid CP search can be used to guide the MIP search or even deduce further reductions during root node

[37]By CP search we mean that no LP relaxation is solved and that the tree is traversed in a depth-first manner.

processing. Since the CP solver is solving the same problem as the MIP solver

- ▷ each generated conflict constraint is valid for the MIP search,
- ▷ each global bound change can be applied at the MIP root node,
- ▷ each feasible solution can be added to the MIP solver's solution pool,
- ▷ the branching statistics can initialize a hybrid MIP branching rule, see Chapter 10 and [AB09], and
- ▷ if the CP solver completely solves the problem, the MIP solver can abort.

All five types of information are supposed to be beneficial for a MIP solver, and are potentially generated by our algorithm which we now describe more formally.

The Rapid Learning algorithm is outlined in Figure 9.2. Here, $l(\tilde{P})$ and $u(\tilde{P})$ shall denote the local lower and upper bound vectors, respectively, of a subproblem $\tilde{P}$ of an integer program P, the symbol C refers to a single globally valid conflict constraint explaining the infeasibility of the current subproblem. Rapid Learning is an incomplete CP search: a branch-and-bound algorithm which traverses the search space in a depth-first manner (Line 4), using domain propagation (Line 5) and conflict analysis (Line 8), but no LP relaxation. Instead, the *pseudo solution* [Ach07b], i.e., an optimal solution of a relaxation consisting only of the variable bounds (Line 6), is used for the bounding step. Variable and value selection takes place in Line 14; Inference Branching (see Chapter 10.2) is used as branching rule – for details and the implementation of Inference Branching in SCIP, see the PhD theses of Achterberg and Heinz [Ach07a, Hei]).

We assume that the domain propagation routines in Line 5 might also deduce global bound changes and modify the global bound vectors $l(P)$ and $u(P)$, and that single-clause conflicts are automatically upgraded to global bound changes. Note that it suffices to check constraint feasibility in Line 11, since the pseudo solution $\bar{x}$ (see Line 6) will always take the value of one of the (integral) bounds for each variable.

Our implementation of the Rapid Learning heuristic uses a secondary SCIP instance to perform the CP search. Only a few parameters need to be altered from their default values to turn SCIP into a CP solver, an overview is given in Table 9.1. Most importantly, we disabled the LP relaxation and use a pure depth-first search with Inference Branching (but without any additional tie breakers). Further, we switch from All-UIP to 1-UIP in order to generate only one conflict per infeasibility. Expensive feasibility checks and propagation of the objective function as a constraint are also avoided. We aim to generate short conflict constraints, since such are most likely to frequently trigger propagations in the upcoming MIP search. Thus, we only

Table 9.1.: Settings for Rapid Learning sub-SCIP.

parameter name	value	effect
lp/solvefreq	-1	disable LP
conflict/fuiplevels	1	use 1-UIP
nodeselection/dfs/stdpriority	INT_MAX/4	use DFS
branching/inference/useweightedsum	FALSE	pure inference, no VSIDS
constraints/disableenfops	TRUE	no extra checks
propagating/pseudoobj/freq	-1	no objective propagation
conflict/maxvarsfac	0.05	only short conflicts
history/valuebased	TRUE	extensive branch. statistics

collect conflicts that contain at most 5 % of the problem variables. Finally, we adapt the collection of branching statistics such that history information on general integer variables are collected per value in the domain rather than having one counter for down- and one for up-branches regardless of the value on which was branched. This can be essential for performing an efficient CP search on general integer variables, see [Hei], and was a building block that enabled us to use Rapid Learning on IPs rather than solely on BPs, as in [BFS10]. In addition to the particular parameters listed in 9.1, we set the emphasis[38] for presolving to "fast". For technical reasons of SCIP, Rapid Learning has been implemented as a separator plugin.

9.4. Computational results

For our computational experiments, we used SCIP 3.0.1.4 compiled with SoPlex 1.7.1 as underlying LP solver. The results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20 GHz with 12 MB cache and 48 GB main memory, running an openSuse 12.3 with a gcc 4.7.2 compiler. Hyperthreading and Turboboost were disabled. We ran only one job per node to reduce random noise in the measured running time that might be caused by delays if multiple processes share common resources, in particular the memory bus. We compared SCIP with default settings to a version of SCIP that runs Rapid Learning once at the root node, after cutting plane separation. We set a time limit of two hours.

For performing the Rapid Learning search in a CP-like fashion, we used a secondary SCIP instance (see previous section). As node limit we used $\max(500, \min(niter, 5000))$, with *niter* being the number of simplex iterations used for solving the root LP in the main instance. We further aborted the CP search prematurely if no information that potentially restricts the search space (i.e., no nogoods, no bound changes, no primal solution) was gained after 20 % of the node limit.

[38]In SCIP, emphasis settings correspond to a group of individual parameters being changed.

We chose two different test sets of pure BPs and IPs; one of the test sets is focused on optimization instances, the other on problems for which feasibility is the driving factor. The first test set, which we call MMM-IP, consists of all BP and IP instances from MIPLIB3.0 [BCMS98], MIPLIB2003 [AKM06], and the benchmark set of MIPLIB2010 [KAA+11]. The second, feasibility-focused, test set contains all BPs and IPs from the Infeasible and the Primal set of MIPLIB2010; we refer to it as the FEASIBILITY test set. These are instances which are either known to be infeasible or for which the root dual bound is equivalent to the optimal solution, i.e., the instance is solved as soon as an optimal solution is found, the proof of optimality comes "for free". Both cases constitute complements of the "standard" MIP case that finding a good feasible solution is much easier than proving optimality. Our hope is that a nogood-based search that ignores the LP optimality bound will be particularly helpful on these instances.

We excluded instances that had more than 10 000 variables or constraints after SCIP's default presolving (motivated by an additional experiment, see below) or that were solved to optimality before Rapid Learning was called. This leaves 71 instances in the MMM-IP test set and 32 in the FEASIBILITY test set.

Our computational experiments extend those in [BFS10] in several aspects. Firstly, we used a newer SCIP version 3.0.1.4 compared to 1.2.0.5. In between these two versions, 33 new default plugins have been added to SCIP, among them five new propagators. Secondly, we also give results for IPs with general integer variables, whereas [BFS10] used Rapid Learning solely for BPs. Third, we present results on a larger set of instances that we split by their type (benchmark sets versus feasibility-driven IPs).

Tables B.18 and B.19 in the appendix compare the performance of SCIP with and without Rapid Learning applied at the root node (Columns SCIP def and SCIP RL). For both branching variants, Nodes and Time give the number of branch-and-bound nodes and the computation time needed to prove optimality, the term limit in the Time column indicates that the time limit of two hours was reached before proving optimality. The four Rapid Learning columns provide detailed information on the performance of Rapid Learning. Time gives the running time of Rapid Learning, Ngds and Bds present the number of applied nogoods and global bound changes, respectively, and a checkmark in Column Sol indicates that a new incumbent solution was found.

Table 9.2 shows aggregated results. Columns Nodes and Time present shifted geometric means for the number of branch-and-bound nodes and the computation times, taken over all instances that could be solved to proven optimality by both versions of SCIP. Column Solved shows how many instances each of the versions could solve and Column Obj indicates for how many of the unsolved instances either version provided a better primal bound at termination. The three Columns Ngds, Bds, and Sol show for how many of the instances in the test sets Rapid Learning learned valid nogoods, global variable bound or a new incumbent solution, respectively.

Table 9.2.: Summary of Rapid Learning results

Test Set/Param	Nodes	Time	Solved	Obj	Ngds	Bds	Sol
MMM-IP							
SCIP def	2444	66.4	58	2			
SCIP RL	2226	66.1	59	4	40/71	8/71	12/71
Feasibility							
SCIP def	3493	264.7	24	0			
SCIP RL	2131	199.0	24	1	23/32	7/32	0/32

Note first that Rapid Learning is indeed rapid, there is only one instance of the mmm-ip test set and six of the Feasibility test set, for which Rapid Learning takes more than five seconds. Among those are only two instances, `neos-859770` and `neos-957389`, for which Rapid Learning consumes more than 10 % of the total running time; consequently, the overall performance deteriorates in both cases. We observe a high variability in the results: in both test sets, there are many instances for which the overall performance improves, as well as several for which it deteriorates. Having a look at the shifted geometric means, we see that for the optimality-focused mmm-ip test set, Rapid Learning is performance neutral w.r.t. running time; for the Feasibility test set, the running time improves by 25 %. The number of nodes decreases significantly in both cases: for the mmm-ip set, the SCIP version with Rapid Learning needs only 91 % of the nodes compared to SCIP default, for the Feasibility set, only 61 % of the nodes are required. These numbers are accompanied by the fact that SCIP with Rapid Learning can solve one more instance to optimality and tends to give better primal bounds at termination, see Table 9.2.

We performed an additional experiment to evaluate the performance of Rapid Learning on large instances with more than 10 000 variables or constraints. For feasibility instances, it improved the average performance; this, however, did not hold for large instances from the Miplibs. For the 21 pure BPs and IPs with more than 10 000 variables or constraints, Rapid Learning slowed down SCIP by 12 % in shifted geometric mean, solving one instance less. This led to the decision to introduce a parameter for the maximum size of instances on which Rapid Learning should run (and thereby excluding those instances from the test set, as stated above).

Learning nogoods constitutes the main contribution of Rapid Learning. For more than half of the instances in either test set, nogoods are generated and added to the original SCIP instance. In Tables B.18 and B.19 it can be seen that if nogoods are generated, then there are typically plenty of them: a few hundreds or thousands. As a comparison, global variable bounds are only learned for 10 % and 20 % of the instances in mmm-ip and Feasibility, respectively, and hardly ever more than 20 variable bounds improve. For all instances, branching statistics were learned for some variables. Interestingly, there are twelve MMM-IP instances for which Rapid Learning found a new incumbent solution, but none in the FEASIBILITY test set. Despite the fact that 40 % of the FEASIBILITY instances are infeasible, this indicates that for the other 60 %, finding good feasible solutions is indeed hard. Nevertheless, excluding infeasible paths in the tree via nogoods seems to help the overall search. Finally, unlike in [BFS10], there was no instance for which the Rapid Learning CP search solved the problem to proven optimality. There were, however, three instances for which the components presolver [GKM$^+$13] of SCIP solved some of its subproblems via Rapid Learning.

We conclude that Rapid Learning is particularly useful for instances in which feasibility plays a major role, while not deteriorating (on average) the performance for "standard" MIPs.

9.5. Conclusion

In this chapter, we introduced a heuristic algorithm that searches for valid conflict constraints by performing a rapid, incomplete CP search at the root node of a MIP solver. This can be seen as a new way to combine CP and IP technologies into a single hybrid framework: a subsidiary CP solver performs a quick, tentative "run-ahead" search on the (sub-)problem that the "master" IP solver is about to explore, learning conflict constraints, variable bounds, and branching statistics that help to guide the IP search. One way of future research could be to perform truly parallel CP and IP searches, which continually communicate conflict constraints and bound changes amongst each other.

So far, our computational experiments indicate that performing a CP search for valid conflicts once at the root node is particularly beneficial for infeasible instances and IPs for which the primal part of the search is the hard one. On general MIPLIB instances, Rapid Learning performed neutral w.r.t. running time while reducing the number of branch-and-bound nodes by about 10 %. We conclude that Rapid Learning extends the MIP solving "bag of tricks" [Fou03] for two classes of problems that are often dismissed as exceptional cases.

Figure 9.2.: Rapid Learning algorithm

```
Input  : IP $P$ as in (2.3) (with $\mathcal{R} = \emptyset$),
         node limit $\text{lim}_{\text{node}}$,
         primal bound $\bar{c}$ for $P$ (might be $\infty$)
Output: set of valid conflict constraints $\mathcal{L}_C$ for $P$,
         valid global domain box $[l, u]$ for $P$,
         feasible solution $\tilde{x}$ for $P$ or $\emptyset$
1  $\mathcal{L} \leftarrow \{P\}$, $\text{n}_{\text{node}} \leftarrow 0$, $\mathcal{L}_C \leftarrow \emptyset$, $\tilde{x} \leftarrow \emptyset$;
2  while $\text{n}_{\text{node}} < \text{lim}_{\text{node}}$ do
3      if $\mathcal{L} = \emptyset$ then goto line 17 ;
4      $\tilde{P} \leftarrow$ select_dfs($\mathcal{L}$), $\mathcal{L} \leftarrow \mathcal{L} \setminus \tilde{P}$, $\text{n}_{\text{node}} \leftarrow \text{n}_{\text{node}} + 1$;
5      $[l(\tilde{P}), u(\tilde{P})] \leftarrow$ propagate($[l(\tilde{P}), u(\tilde{P})]$);
6      $\bar{x} \leftarrow \text{argmin}\{c^{\mathsf{T}}x \mid x \in [l(\tilde{P}), u(\tilde{P})]\}$;

       /* analyze infeasible subproblem, potentially store
          globally valid conflict constraint                    */
7      if $[l(\tilde{P}), u(\tilde{P})] = \emptyset$ or $c(\bar{x}) \geq \bar{c}$ then
8          $C \leftarrow$ analyze($\tilde{P}$);
9          if $C \neq \emptyset$ then $\mathcal{L}_C \leftarrow \mathcal{L}_C \cup \{C\}$;
10         goto line 2;

       /* check for new incumbent solution                      */
11     if $A\bar{x} \leqslant b$ and $c^T\bar{x} < \bar{c}$ then
12         $\tilde{x} \leftarrow \bar{x}$, $\bar{c} \leftarrow c^T\bar{x}$;
13         goto line 2;

14     $(x_i, v) \leftarrow$ select_infer($\tilde{P}$, $\bar{x}$);
15     $\tilde{P}_l \leftarrow \tilde{P} \cup \{x_i \leqslant v\}$, $\tilde{P}_r \leftarrow \tilde{P} \cup \{x_i \geqslant v\}$;
16     $\mathcal{L} \leftarrow \mathcal{L} \cup \{\tilde{P}_l, \tilde{P}_r\}$;
17 return $(\mathcal{L}_C, [l(P), u(P)], \tilde{x})$;
```

10. Cloud Branching

After five chapters dedicated to primal heuristics and the previous chapter on a heuristic algorithm for conflict learning, we now discuss branching heuristics.[39]

Branch-and-bound methods for mixed integer linear programming are traditionally based on solving a linear programming relaxation and branching on a variable which takes a fractional value in the (single) computed relaxation optimum. In this chapter, we study branching strategies for mixed integer programs that exploit the knowledge of *multiple* alternative optimal solutions of the current LP relaxation, a concept which we refer to as *Cloud Branching*. These strategies naturally extend state-of-the-art methods like Full Strong Branching [ABCC95], Pseudocost Branching [BGG$^+$71, GR77], and their hybrids.

We show that by exploiting dual degeneracy, and thus multiple alternative optimal solutions, it is possible to enhance traditional methods. We present computational results, applying the newly proposed strategy to Full Strong Branching, which is known to result in small search tress [AKM05, Ach07b]. We observe that Cloud Branching yields search trees of similar size, but reduces the mean running time by up to 30% on standard test sets.

The idea and implementation of Cloud Branching is joint work with Domenico Salvagnin from the Università degli Studi di Padova; most of this chapter has been published in [BS13]. The description of Hybrid Branching in Section 10.2 is based on [AB09].

This chapter is organized as follows. After a brief introduction in Section 10.1, Section 10.2 gives an overview of existing branching strategies from the literature. In Section 10.3 we discuss how to generate alternative optimal solutions (a *cloud* of solutions), and in Section 10.4 we argue how to exploit this information to enhance pseudocost-based branching strategies. In Section 10.5 we give more details on the technique applied to Full Strong Branching, while in Section 10.6 we report a computational evaluation of the proposed method. Conclusions are finally drawn in Section 10.7.

10.1. Introduction

Good branching strategies are crucial for any branch-and-bound based MIP solver (see, e.g., [AW13]). Unsurprisingly, the topic has been subject of constant and active research since the very beginning of computational mixed

[39]The terms branching heuristic, branching strategy, and branching rule are used synonymously in the literature.

integer programming, see, e.g., [BGG$^+$71, GR77, FLRKS78]. We refer to [LS99, AKM05, Ach07b] for comprehensive studies of branching strategies.

In mixed integer programming, the most common methodology for branching is to split the domain of a single variable into two disjoint intervals. For binary variables this is equivalent to fixing the variable. The ultimate goal is to find a dichotomy which reduces the computational complexity the most. Since this is very hard to determine[40], the rules for making branching decisions are of a very heuristic nature. In constraint programming, the variable and value selection are often referred to as "the heuristic", emphasizing at the same time that branching is *the* central component of a branch-and-bound search and that this central component depends on somewhat arbitrary decisions.

In this chapter, we will address the key problem of how to select such a variable for an LP-based binary branching scheme. Let $\bar{x}$ be an optimal solution of the LP relaxation at the current node of the branch-and-bound tree and let $\mathcal{F} = \{j \in \mathcal{I} : \bar{x}_j \notin \mathbb{Z}\}$ denote the set of fractional variables. A general scheme for branching strategies consists of computing a score s_j for each fractional variable $j \in \mathcal{F}$, and then choosing the variable with maximum score for branching. Different branching rules then correspond to different ways of computing this score.

The contribution of this chapter is twofold. First, we introduce for the first time, to the best of our knowledge, a branching strategy that makes use of multiple relaxation solutions and show how it can be naturally integrated into existing branching rules. Second, we evaluate one particular implementation of it in the context of Full Strong Branching – the branching rule commonly known to be most efficient w.r.t. the number of branch-and-bound nodes [Ach07b, AKM06]. We demonstrate that it leads to significant savings in computation time while not increasing the number of nodes.

10.2. Branching heuristics for MIP

This section gives an overview on branching strategies for mixed integer linear programming that have been suggested in the literature. We concentrate on the problem of selecting a branching dichotomy; for an introduction to node and child selection rules see, e.g., Achterberg [Ach07b]. We further restrict this literature overview to publications that address MIP in general and do not consider customized branching rules for particular applications of MIP. Section 10.2.1 is concerned with strategies that employ binary branching[41] on variables, mostly taking the LP relaxation into consideration. The filtering algorithm which we introduce in Section 10.3 is also designed for branching on variables. Section 10.2.2 discusses branching strategies that take into

[40]Liberatore [Lib00] proved that for SAT, it is $\mathcal{NP}$-hard to decide whether branching on a certain binary variable gives rise to a search tree with a minimum number of nodes.

[41]Branching with more than two children has been successfully used for special applications, e.g., [BFM98], but is not commonly used for general MIP solving.

account general (typically linear) disjunctions, a methodology that has been widely discussed in recent literature. See Chapter 2.2 for a brief introduction to the LP-based branch-and-bound algorithm.

10.2.1. Branching on variables

Branching on variables means to select a single unfixed variable x_j, $j \in \mathcal{I}$, and to split the current problem into two subproblems based on the disjunction $x_j \leq \alpha \bigvee x_j \geq \alpha + 1$ for some value $l_j \leq \alpha < u_j$. Typically, one chooses $j \in \mathcal{F}$ and $\alpha = \lfloor \bar{x}_j \rfloor$, with $\bar{x}$ being an optimal solution of the LP relaxation.

A straight-forward idea for selecting the branching variable is *Most Fractional Branching*, i.e., to branch on the variable x_j whose fractional part $\bar{x}_j - \lfloor \bar{x}_j \rfloor$ is as close as possible to 0.5. However, this is known to perform poorly in practice [BFG$^+$00, Ach07b]. A much more computationally effective approach is to consider the impact of the branching decision on the dual bounds (i.e., the objective of the relaxation) of the child nodes. Already in one of the first papers that suggested to solve MIPs by branch-and-bound, a branching rule was used that took into account the increase of the objective function when performing one dual simplex pivot after adding a bound change [Dak65]. These values are known as Driebeek penalties, referring to [Dri66]. This method was later refined by Tomlin [Tom71].

A more involved branching strategy is *Pseudocost Branching* [BGG$^+$71, GR77], which consists of keeping a history of how much the dual bound improved when branching on a given variable in previous nodes, and then using these statistics to estimate how the dual bound will improve when branching on that variable at the current node. For technical details on Pseudocost Branching, please see Section 10.4. Pseudocost Branching is computationally cheap since no additional LPs need to be solved. It performs reasonably well in practice [Ach07b]. Yet at the very beginning, when the most crucial branching decisions are taken, there is no reliable historic information to build upon.

An important of many branching rule is *strong branching*, which originated from the TSP community [ABCC95, ABCC07]. The basic idea consists in simulating branching on the variables in $\mathcal{F}$ and then choosing the actual branching variable as the one that gave the best progress in the dual bound. Put simply: strong branching computes the values which Pseudocost Branching estimates. Interestingly, this greedy local method empirically proves to be the best variable-based branching rule w.r.t. the number of nodes of the resulting branch-and-bound tree [Ach07b, AKM05], but it introduces a large overhead in terms of computation time, since $2 \cdot |\mathcal{F}|$ auxiliary LPs need to be solved at each node. Many techniques have been studied to speed up the computational burden of strong branching, in particular by heuristically restricting the list of branching candidates and by imposing simplex iteration limits on the strong branching LPs [LS99]. Implementations of strong branching in MIP solvers will typically use a couple of such heuristic limits to accelerate the branching decision. For disambiguation, we refer to a branching rule that solves all $2 \cdot |\mathcal{F}|$ strong branching LPs to optimality as *Full Strong Branching*. Recent improvements for strong branching include the application of domain propagation techniques to obtain better bounds [Gam14, Gam13] and neglection of inferior candidates during the strong branching process to speed up the decision [FM12]. However, according to computational studies, a pure strong branching rule is still too slow for practical purposes.

Already in 1977, Gauthier and Ribière [GR77] suggested a version of what today would be called *Pseudocost Branching with Strong Branching initialization*. Branching rules such as *Reliability Branching* [AKM05] or *Hybrid Branching* [AB09] refine the idea of combining mechanisms from Pseudocost Branching with strong branching. Reliability Branching performs strong branching on a branching candidate whenever its pseudocost history is considered *unreliable*, i.e., whenever it has not been updated often enough. Achterberg et al. [AKM05, Ach07b] suggest to use a threshold value of 8 updates, before considering a variable reliable, with a dynamic adjustment of this threshold which depends on the number of simplex iterations needed to solve the strong branching LPs. If those are cheap, the algorithm allows for more strong branching calls before switching to pseudocosts.

For constraint satisfaction problems, where no objective function is available, one may better estimate the impact of a branching by taking the number of implied reductions of other variable domains into account [LA97]. In analogy to the pseudocosts, we call the estimated numbers of implied reductions the *inference values* of a variable.

In pure SAT solvers, learning short, valid conflict clauses from the analysis of infeasible subproblems is one of the key ingredients [MSS99]. The *variable state independent decaying sum (VSIDS)* [MMZ$^+$01] branching strategy, which is a common rule in SAT solving, prefers variables that have been used to create recent conflict clauses. The idea to use the average lengths of the conflict clauses in which a variable appears, for branching in MIP was suggested by Kılınç Karzan et al. [KKNS09]. We call these the *conflict lengths* of a variable.

Hybrid Branching was originally introduced by Achterberg [Ach07b] and with some modifications published in [AB09]. It combines pseudocosts, inference values, VSIDS and conflict lengths into a single variable selection criterion which additionally includes a score based on the number of subproblems that could be pruned due to branching on this variable (the *cutoff values*). Therefore, Hybrid Branching first normalizes all the five individual values by mapping them into the interval $[0, 1)$. Afterwards, it takes a weighted sum of them, putting a high weight on the pseudocosts, a medium weight on the conflict values and lengths, and a low weight on the inference and cutoff values. To the best of our knowledge, most MIP solvers currently employ such a branching rule or other variants of Reliability Branching as default branching rule.

For hard MIP problems, a few extensions of strong branching have been proposed. Glankwamdee and Linderoth [GL06] studied the impact of performing a two-level strong branching on pairs of variables, and showed that this often helps to save nodes but is performance-neutral time-wise when compared to Full Strong Branching. Gilpin and Sandholm [GS11] suggested a variant of strong branching, called *Entropy Branching*, in which they consider the sum of (logarithms of) the fractionalities in the strong branching subproblems to compute a branching score, rather than the change in the LP objective. This brings us to the topic of criteria for variable-based branching that are different from the objective gain.

The *Active Constraint* method of Patel and Chinneck [PC07] is based on the impact of variables on those linear constraints which are fulfilled with equality by the current LP solution. The computational experiments of [PC07] concentrated on minimizing the computational effort needed to find a first feasible solution; hence, this branching rule should also be considered in the context of diving heuristics, see Chapter 2.3. It is similar to Vectorlength Diving [Ber06, Ber08], which uses the impact of a branching decision on the activity of all linear constraints (not only the active ones). From seven tested variants, the most successful ones in [PC07] were “Scheme A”, which simply counts the number of active constraints, and “Scheme B”, which totals up the inverse of the coefficient sums of all active constraints in which a variable appears.

Pryor and Chinneck [PC11] study algorithms that make a joint selection of the branching variable and its branching direction (i.e., the child selection), extending methods of Pesant and Quimper [PQ08] (see also [PQZ12]) for constraint programming. The central idea is to branch into a direction where a constraint is very likely to become unsatisfiable, thereby either being able to prune the resulting subtree quickly or triggering a huge amount of propagations. Pryor and Chinneck further suggest a child selection strategy that chooses the next node by the “most violating votes”, i.e., in our notation, the sum of locks of active constraints into the direction of infeasibility. The default child selection rule in SCIP combines the inference values (see above) of a variable with the direction in which the LP values evolved since the root node (see [Mar99]). In a comparison to the methods from [PC07], it turns out that branching on active constraints is superior when there are only inequalities in the MIP formulation; when equations are present, *Probability-Based Branching* performs better. Again, the used measure is the time needed to find a first feasible solution, which can be taken as a motivation to re-consider the suggested methods in the context of diving heuristics.

Kılınç Karzan et al. [KKNS09] introduced a branching scheme that is guided by the information gained from collecting valid conflict clauses. They divide the branch-and-bound search in two phases, conducting a single restart (see, e.g., [Ach07b] for restarts in MIP) after a certain number of nodes have been fathomed. To obtain minimum cardinality conflict constraints from pruned nodes, an auxiliary MIP is (partially) solved, as opposed to applying a greedy heuristic approach that has been suggested in [Ach07a]. After the restart, the branching criterion is to select variables that appear in short conflict constraints to enforce an early pruning of nodes. The idea of performing a partial search to gain information for the main branch-and-bound search is similar to Rapid Learning, see Chapter 9 and [BFS10]; the idea of using the length of conflict clauses for variable selection has been incorporated into Hybrid Branching, see above. In [KKNS09], improvements for medium hard[42] MBPs are reported. We performed a similar experiment by setting the SCIP parameters "conflict/restartnum" to 200 and "conflict/restartfac" to a large value. In a first shot, this yielded a neutral impact on the overall performance, indicating that with some additional implementation and tuning efforts, this one-time restarting mechanism might also prove beneficial for SCIP.

Multiple restarts are applied for *Backdoor Branching*, which has been suggested by Fischetti and Monaci [FM11]. The concept of backdoors comes from artificial intelligence [WGS03] for solving SAT problems and refers to, loosely speaking, a subset of the discrete variables which, once fixed to any combination of values, enforces feasibility or infeasibility for the remaining problem. In MIP context, this means that it is sufficient to branch on variables in a backdoor[43], the integrality of the other discrete variables (or the infeasibility of the relaxation) will be achieved implicitly, e.g., by propagation or because all optimal corner solutions of the remaining LP relaxation will be integral. Dilkina et al. [DGM+09] have shown that many Miplib instances have small backdoors (which are, however, hard to detect). Backdoor Branching aims at finding a good approximation of a backdoor. After each restart, the approximated backdoor is computed by solving a set covering problem. Branching is exclusively performed on backdoor variables until all of them are fixed. The idea of solving set covering models for finding branching decisions that lead to structurally easier subproblems is related to *Undercover Branching* for MINLP [BG13], cf. Chapter 8.

10.2.2. Branching on general disjunctions

All methods described so far considered branching on variables, i.e., branching on the simple disjunction $x_j \leq \lfloor \bar{x}_j \rfloor \bigvee x_j \geq \lceil \bar{x}_j \rceil$ for some variable x_j with $j \in \mathcal{F}$. Clearly, branching can also be performed on more general disjunctions. An early application of branching on general disjunctions has been presented by Ryan and Foster for solving scheduling problems [RF81]. Their *Constraint Branching* enforces a *sum* of binary decision variables to be either zero or greater equal one. Another example is branching on *special ordered sets* (SOSs) of variables, for which one SOS constraint gets split into two SOSs of half the length [BT70]. The variable-based branching in

[42]Here: instances for which Cplex 11.1 needs more than one minute but less than two hours to prove optimality.

[43]The backdoor definition used in [FM11] is called a *strong* backdoor to optimality in [DGM+09].

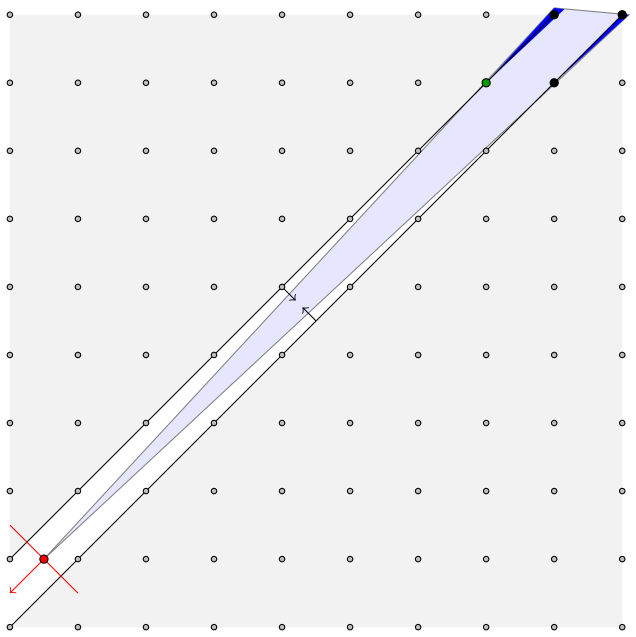


Figure 10.1.: An integer program for which branching on the general disjunction $x_2 - x_1 \geq 1 \bigvee x_2 - x_1 \leq 0$ is preferable to variable-based branching.

Lenstra's algorithm for integer programming in finite dimensions [LJ83] that takes place after performing a basis transformation of $\mathbb{R}^n$ can be interpreted as branching on general disjunctions w.r.t. the Cartesian basis. An overview on recent developments for branching on constraints can be found in the PhD thesis of Mahajan [Mah09].

A typical result when branching on general disjunctions in MIP, as we will see below, is that the generated branching trees are smaller on average, but the performance deteriorates or is neutral w.r.t. running time. One major reason for this computational overhead is that branching on variables *decreases* the size of the LP relaxation for the subproblems by (at least) one column per branching, whereas branching on general disjunctions potentially *increases* the LP's size by one row. Particularly, this means that the dimension of the LP basis increases, therefore the basis matrix will have to be refactorized, causing additional computational overhead.

Most MIP literature for branching on general disjunctions considers so-called split disjunctions [CKS90]:

$$\sum_{i \in \mathcal{I}} \pi_i x_i \leq \pi_0 \bigvee \sum_{i \in \mathcal{I}} \pi_i x_i \geq \pi_0 + 1 \qquad (10.1)$$

with $\pi_i \in \mathbb{Z}$ for all $i \in \mathcal{I} \cup \{0\}$. This methodology is synonymously referred to as *branching on hyperplanes* and *branching on constraints*. For an illustration see Figure 10.1. The core question for using general disjunctions to solve

MIPs is how to identify “good” disjunctions – and to find a suitable notion of “good”.

Owen and Mehrotra [OM01] provide a *disjunctive branch-and-bound* procedure and prove that their algorithm is finite, if all variables have finite bounds and the size of the coefficients in the used disjunctions is bounded, i.e., there is an $M \in \mathbb{Z}_{>0}$ such that in every branching step it holds that $|\pi_i| \leq M$ for all i. Their algorithm determines the branching disjunction via a neighborhood search heuristic (compare Chapter 6) that aims at maximizing the child nodes’ objective gains which are computed in a strong branching fashion. Therefore, they restrict the search to coefficients $\pi_i \in \{-1, 0, 1\}$. The disjunction shown in Figure 10.1 falls into this category.

Karamanov and Cornuéjols [KC11] introduce a different heuristic algorithm for selecting a branching disjunction. They consider the set of disjunctions which correspond to Gomory mixed integer cuts (GMICs) [Gom60]. Therefore, they use that GMICs can be viewed as intersection cuts [Bal71] which are derived from split disjunctions of type (10.1). They filter the GMICs to only keep the ten deepest cuts, and apply a strong-branching-like[44] procedure on the corresponding (at most) ten candidate disjunctions. In extensive computational experiments, the authors observe that their branching rule produces smaller search trees on solved instances and a smaller optimality gap on unsolved instance, with comparable running time when tested against standard variable-based rules. An extension of [KC11] is proposed by Cornuéjols et al. [CLN11] who not only consider GMICs on tableau rows, but also on linear combinations of the tableau rows.

Mahajan and Ralphs [MR09] put the question of finding a good branching decision to the extreme. In their paper, they formulate the problems of finding a disjunction (10.1) that maximizes the objective gain (here: the minimum of the LP optima of the two child nodes) or minimizes the width of the LP-polyhedron in direction π as a bilinear program or a mixed integer program, respectively. The disjunction shown in Figure 10.1 is optimal w.r.t. both criteria. The same team of authors proved that the problems of finding a general disjunction with maximal objective gain or minimal width are $\mathcal{NP}$-hard [MR10], even when the integer variables are all binary and several restrictions on the disjunction (10.1) are assumed. In particular, they prove that $\mathcal{NP}$-hardness still holds when demanding that all coefficients π_i be in $\{-1, 0, 1\}$, as Owen and Mehrotra did. Note that finding a *variable disjunction* that maximizes the objective gain can be achieved in polynomial time by solving at most $2 \cdot |\mathcal{F}|$ LPs; this is what Full Strong Branching does.

Combining ideas from [OM01] and [PC07], Mahmoud and Chinneck [MC13] choose a constraint that is active for the current LP optimum, and construct a general disjunction with coefficients in $\{-1, 0, 1\}$ that is as perpendicular or as parallel as possible to the chosen active constraint. They showed that a

[44]That means, they tentatively add the candidate disjunctions to the LP, solve it and use the dual bound change as a score to choose a branching dichotomy.

combined branching strategy of branching on variables and constraints leads to a reduction in the time to find a first feasible solution (and to better solution qualities) on hard models.

Interdiction Branching for binary programs by Lodi et al. [LRRS11] is at the borderline of variable-based and constraint-based branching. For a suitable subset $\{i_1, \dots, i_{n'}\} =: \mathcal{B}' \subseteq \mathcal{B}$, of the variables, it generates a branching disjunction

$$x_{i_1} = 1 \bigvee x_{i_2} = 1 \bigvee \cdots \bigvee x_{i_{n'}} = 1 \bigvee \sum_{i \in \mathcal{B}'} x_i = 0. \qquad (10.2)$$

This is a $(n'+1)$-ary branching, of which in n' children a variable is fixed and only for one child n variables are fixed. A version of Interdiction Branching that heuristically determines $\mathcal{B}'$ to consist of variables that lead to improving solutions could be successfully applied in [LRRS11] to solve knapsack and stable set instances. The idea of Interdiction Branching is closely connected to *Orbital Branching* [OLRS11] and *Constraint Orbital Branching* [OLRS08] which are general branching frameworks for highly symmetric MIP problems. Here, all variables from one orbit are considered to create a disjunction as in (10.2).

Finally, Local Branching by Fischetti and Lodi [FL03] was originally published as a strategy to interleave variable-based branching with branching on general $\{-1, 0, 1\}$-disjunctions. These disjunctions measure the distance to the incumbent solution. Nowadays, Local Branching is mainly used as a large neighborhood search heuristic that explores the smaller of the two subtrees, cf. Chapter 6.

Although branching on general disjunctions has been very successfully applied to solve some special cases of MIP[45], it is not employed by default by SCIP or Cplex [Ach] for MIP solving. Since version 7.0, Xpress has used branching on split disjunctions for problems that feature general integer variables [Per11].

A note on branching and heuristics

All the described procedures make branching decisions by heuristic criteria; the impact on the tree size is never clear a priori. The main reason for this is that for $\mathcal{NP}$-complete problems, finding a branching that leads to a minimal search tree is at least as hard as solving the underlying problem (compare [Lib00]) itself, potentially even harder. In a recent study, Le Bodic and Nemhauser show how the default branching rules of several state-of-the-art MIP solvers can be tricked into producing humongous search trees for a class of MIP problems where small search trees are easy to construct when knowing the problem structure at hand [LBN14].

[45]Notably, hard market split problems have been solved by an algorithm that uses lattice basis reductions to determine good branching directions [ABH$^+$00]

The relation between branching rules and heuristics, however, goes further than just branching being a heuristic process. We saw several interconnections between branching and primal heuristics on the last few pages. The branching rule of Owen and Mehrotra employs a local search heuristic to greedily add variables to the branching disjunction. Local Branching [FL03] has been originally designed as a branching rule and is merely used as primal heuristic nowadays. The other way around, the notion of minimal covers of MINLPs has been introduced in a primal heuristic context [BG14], but can be used for efficient branching as well [BG13]. The publications of Chinneck et al. [MC13, PC07, PC11] that focus on branching rules for finding a first feasible solution can equally well be considered as contributions to diving heuristics.

We conclude that there is an intersection, or rather a transition, between the topics of primal heuristics and branching rules.

10.3. A cloud of solutions

All branching strategies described in the previous section are naturally designed to deal with only one optimal fractional solution. History-based rules use the statistics collected in the process to compute the score of a variable starting from the current fractional solution. Even with strong branching, the list of branching candidates is defined according to the current fractional solution $\bar{x}$.

However, LP relaxations of MIP instances are well-known for often being massively degenerate [BFZ10, EMDS11]; multiple equivalent optimal solutions are the rule rather than the exception. Thus branching rules that consider only one optimal solution risk taking arbitrary branching decisions, thereby contributing to performance variability, see [KAA$^+$11]. In the following two sections we discuss the extension of some branching strategies to exploit the knowledge of multiple optimal solutions of the current LP relaxations.

In order to extend standard branching strategies to deal with multiple LP optima at the same time, we need to solve two problems:

1. How to generate efficiently multiple optimal solutions of the current LP relaxation?

2. How to make use of the additional information provided by these solutions?

The first problem can be effectively solved by restricting the search to the optimal face of the LP relaxation polyhedron. On this face, an auxiliary objective function can be used to move to different bases. From a computational point of view, fixing to the optimal face can be easily and safely implemented by fixing all variables whose reduced costs are non-zero, using the reduced costs associated to the starting optimal basis. As far as the choice of the

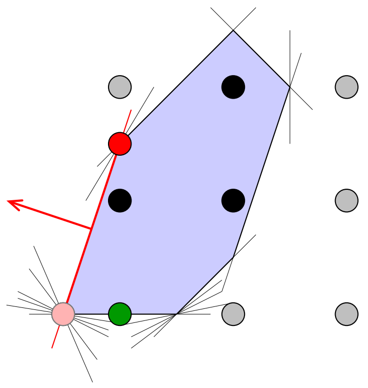


Figure 10.2.: An IP with a primal (over-determined corner solutions) and dual (multiple optimal solutions) degenerate LP relaxation.

second level objective function(s) is concerned, different strategies might be used. One option is to try to minimize and maximize each variable which is not yet fixed: this is what optimality-based bound tightening techniques do (see, e.g., [ZG99, CL10]), with the additional constraint of staying on the optimal face. Another option is to use a feasibility pump [FGL05] like objective function (see also Chapter 5, in which the current LP point is rounded and a Hamming distance function is generated to move to a different point (more details will be given in the next section): this is related to the PumpReduce procedure that CPLEX performs to achieve more integral LP optima [Ach10, Ach11]. Finally, a random objective function might be used.

Suppose now that we have constructed, in one way or another, a *cloud* $\mathcal{S} = \{\bar{x}^1, \dots, \bar{x}^k\}$ of alternative optimal solutions to the current LP relaxation. We assume that the initial fractional solution $\bar{x} \in \mathcal{S}$. Given $\mathcal{S}$, we can define our initial set of branching candidates $\mathcal{F}(\mathcal{S})$ as

$$\mathcal{F}(\mathcal{S}) = \{j \in \mathcal{I} \mid \exists \bar{x}^i \in \mathcal{S} : \ \bar{x}^i_j \notin \mathbb{Z}\},$$

i.e., $\mathcal{F}(\mathcal{S})$ contains all the variables that are fractional in at least one solution of the cloud. For each variable in $\mathcal{F}(\mathcal{S})$ it is then possible to calculate its *cloud interval* $I_j = [l_j, u_j]$, where:

$$l_j = \min\{\bar{x}^i_j \mid \bar{x}^i \in \mathcal{S}\}$$

$$u_j = \max\{\bar{x}^i_j \mid \bar{x}^i \in \mathcal{S}\}$$

Given the cloud interval for each branching candidate, we partition the set $\mathcal{F}(\mathcal{S})$ into three subsets, depending on the relative intersection between each interval I_j and the *branching interval* $B_j = [\lfloor \bar{x}_j \rfloor, \lceil \bar{x}_j \rceil]$. In particular, we define:

$$\mathcal{F}_2 \coloneqq \{j \in \mathcal{F}(\mathcal{S}) \mid \lfloor \bar{x}_j \rfloor < l_j \ \wedge \ u_j < \lceil \bar{x}_j \rceil\}$$

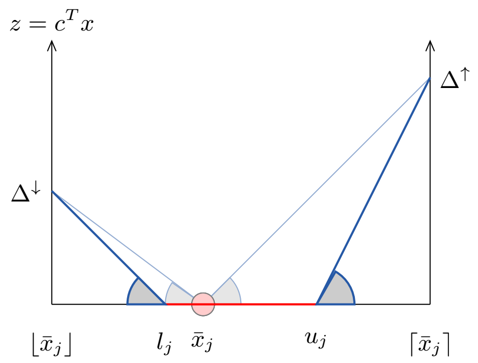


Figure 10.3.: Graphical representation of pseudocosts update, with and without using cloud intervals.

$$\mathcal{F}_0 := \{j \in \mathcal{F}(\mathcal{S}) \mid l_j \leqslant \lfloor \bar{x}_j \rfloor \;\wedge\; \lceil \bar{x}_j \rceil \leqslant u_j\}$$

$$\mathcal{F}_1 := \mathcal{F}(\mathcal{S}) \setminus (\mathcal{F}_2 \cup \mathcal{F}_0)$$

In particular for binary variables, $\mathcal{F}_2$ contains exactly those variables which are fractional for all $\bar{x}^i \in \mathcal{S}$, or differently spoken: $\mathcal{F}(\mathcal{S})$ is the union (taken over $\mathcal{S}$) of all branching candidates, $\mathcal{F}_2$ is the intersection. If $\mathcal{S}$ contained all vertices of the optimal face, then $\mathcal{F}_2$ would be exactly the set of variables that are guaranteed to improve the dual bound in both child nodes. The hope is that also with a limited set of sample points in $\mathcal{S}$, the set $\mathcal{F}_2$ will still be a good approximation to that set.

A variable being contained in the set $\mathcal{F}_0$ is a certificate that branching on it will not improve the dual bound on either side since alternative optima exist which respect the bounds after branching. For the same reasoning, variables in $\mathcal{F}_1$ are those for which the objective function will stay constant for one child, but hopefully not for the other.

The details about how branching rules can be extended to deal with this additional information, namely the three-way partition of the branching candidates $(\mathcal{F}_2, \mathcal{F}_1, \mathcal{F}_0)$ and the set of cloud intervals I_j, of course depends on the particular strategy. For example, a rule based on strong branching can safely skip variables in $\mathcal{F}_0$, thereby saving some LP solves. More details on how to extend a Full Strong Branching policy to the cloud will be given in Section 10.5. In the following section, we will describe how Pseudocost Branching can be modified to exploit cloud information.

10.4. Pseudocost Branching with a cloud

Pseudocost Branching consists mainly of two operations: (i) updating the pseudocosts after an actual branching has been performed and the LP relaxations of the child nodes have been solved and (ii) computing the score

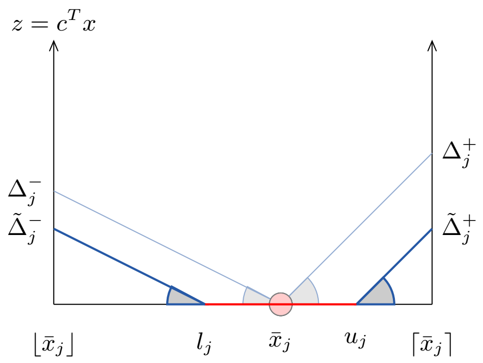


Figure 10.4.: Graphical representation of pseudocosts deployment: estimating the objective gain with and without using cloud intervals.

of a variable using the current pseudocosts when deciding for a branching candidate. When updating the pseudocosts, the objective gains ς_j^+ and ς_j^- per unit change in variable x_j are computed, that is:

$$\varsigma_j^+ = \frac{\Delta^{\uparrow}}{\lceil \bar{x}_j \rceil - \bar{x}_j} \quad \text{and} \quad \varsigma_j^- = \frac{\Delta^{\downarrow}}{\bar{x}_j - \lfloor \bar{x}_j \rfloor} \tag{10.3}$$

where $\Delta^{\uparrow}$ and $\Delta^{\downarrow}$ are the differences between the optimal LP objectives of the corresponding child nodes and the current LP value.

These gains are then used to update the current pseudocosts Ψ_j^+ and Ψ_j^- which are the averages of the objective gains (per unit step length) that have been observed for that particular variable so far. The thin, light blue line in Figure 10.3 illustrates the operation. These estimation formulas are based on the assumption that the objective increases linearly in both directions (hence the resulting triangle). This, however, may be a too crude approximation of the real shape of the projection on the split domain of x_j. In the case of dual degeneracy, there might be many optimal LP solutions with different values for x_j. Which of these values $\bar{x}_j$ takes is more or less arbitrary, but crucial for the current – and by that also for future – branching decisions.

By using the interval I_j, we replace this approximation with another model which is intended to be more precise (thick, dark blue line in Figure 10.3). The corresponding way to compute gains is then:

$$\tilde{\varsigma}_j^+ = \frac{\Delta^{\uparrow}}{\lceil \bar{x}_j \rceil - u_j} \quad \text{and} \quad \tilde{\varsigma}_j^- = \frac{\Delta^{\downarrow}}{l_j - \lfloor \bar{x}_j \rfloor} \tag{10.4}$$

Given a fixed LP relaxation, the values for ς^+ and ς^- may vary by chance when alternative LP solutions were used for their computation. The values $\tilde{\varsigma}^+$ and $\tilde{\varsigma}^-$ will be constant, when the set of all corners of the optimal face is used as a cloud.

As far as the computation of the score s_j is concerned, the standard formulas to predict the objective gains when branching on variable x_j are

$$\Delta_j^+ = \Psi_j^+(\lceil \bar{x}_j \rceil - \bar{x}_j) \quad \text{and} \quad \Delta_j^- = \Psi_j^-(\bar{x}_j - \lfloor \bar{x}_j \rfloor) \tag{10.5}$$

Again, the underlying linear model may give a too optimistic estimate on the dual bound improvements. We suggest to incorporate the interval I_j into the estimation, aiming for a more accurate prediction:

$$\tilde{\Delta}_j^+ = \Psi_j^+(\lceil \bar{x}_j \rceil - l_j) \quad \text{and} \quad \tilde{\Delta}_j^- = \Psi_j^-(u_j - \lfloor \bar{x}_j \rfloor) \tag{10.6}$$

A graphical representation is depicted in Figure 10.4. The thin, light blue lines indicate which values Δ^- and Δ^+ would be computed with the standard model, the thick, dark blue lines show the values for the interval-based scheme. More generally, the following observation holds:

Lemma 10.1. *Let $\bar{x}$ be an optimal solution of the LP relaxation at a given branch-and-bound node and $\lfloor \bar{x}_j \rfloor \leqslant l_j \leqslant \bar{x}_j \leqslant u_j \leqslant \lceil \bar{x}_j \rceil$. Then*

1. *for fixed $\Delta^\uparrow$ and $\Delta^\downarrow$, it holds that $\tilde{\varsigma}_j^+ \geq \varsigma_j^+$ and $\tilde{\varsigma}_j^- \geq \varsigma_j^-$, respectively;*
2. *for fixed Ψ_j^+ and Ψ_j^-, it holds that $\tilde{\Delta}_j^+ \leqslant \Delta_j^+$ and $\tilde{\Delta}_j^- \leqslant \Delta_j^-$, respectively.*

Proof. Follows directly from Equations (10.3)–(10.6). □

Thus, under the same preconditions, the standard pseudocosts will be an underestimation of the pseudocosts based on the cloud intervals, whereas the objective gain, on which the branching decision is made, will be an overestimation. Of course these quantities interact directly which each other: as soon as one of it gets altered, this will have an impact on all upcoming branching decisions and pseudocost computations. The effects of continuous over- and underestimation are likely to amplify each other. The hope is that Cloud Branching helps to make better, more reliable predictions and thereby leads to better branching decisions.

10.5. Full strong branching with a cloud

In the present section we detail the extension of a full strong branching strategy to the cloud. The first problem is again how to generate a cloud of optimal LP solutions $\mathcal{S}$. Following preliminary computational results, we opted for a feasibility pump like objective function, minimizing the distance to the nearest integral point. More precisely, given a fractional solution $\bar{x}$, we define the objective function coefficient c_j of variable x_j as

$$c_j = \begin{cases} 1 & \text{if } \quad 0 \;\; < \bar{x}_j - \lfloor \bar{x}_j \rfloor < 0.5 \\ -1 & \text{if } \quad 0.5 \leqslant \bar{x}_j - \lfloor \bar{x}_j \rfloor < 1 \\ 0 & \text{otherwise} \end{cases}$$

where $\bar{x}_j - \lfloor \bar{x}_j \rfloor$ is the fractional part of $\bar{x}_j$. Using the primal simplex, we re-solve the LP (fixed to the optimal face) with this new objective function. We update the interval bound vectors l_j and u_j, and iterate, using the new optimum as $\bar{x}$. If, at a given iteration, the update did not yield a new integral interval bound, we stop.

As far as the three-way partition $(\mathcal{F}_2, \mathcal{F}_1, \mathcal{F}_0)$ is concerned, we perform Full Strong Branching on all variables in the set $\mathcal{F}_2$. If we even find a variable in this set with a strictly improved dual bound in both child nodes, we stop after $\mathcal{F}_2$ and pick the best variable within this set, completely ignoring sets $\mathcal{F}_1$ and $\mathcal{F}_0$. In state-of-the-art solvers such as CPLEX or SCIP the score of a variable is computed as the *product* of the objective gains in both directions (maybe using a minimum value of some epsilon close to zero for each factor). By this, the score of all variables in $\mathcal{F}_1 \cap \mathcal{F}_0$ will be (nearly) zero and therefore none of them will have maximum score.

Note that in this case cloud information is used essentially to filter out variables and solve a smaller number of LPs. If no variable with strictly improving bounds on both sides is found, different strategies can be devised, depending on how we deal with the remaining variables. One option is to proceed with performing strong branching on the variables in set $\mathcal{F}_1$, but solving only one LP per variable (because by definition we already know that in one direction the dual bound change is zero). Note that variables in $\mathcal{F}_1$ are not necessarily a subset of the fractional variables in $\bar{x}$: as such, while we may still have some speedup because we only solve one LP per variable, the number of variables may indeed be higher than what standard Full Strong Branching would have done. If we can find at least one variable in $\mathcal{F}_2 \cap \mathcal{F}_1$ with a strictly improved dual bound in one direction, then we can stop and ignore set $\mathcal{F}_0$ for the same reason as before. If this is not the case, then we know that for all variables in $\mathcal{F}(\mathcal{S})$ no improvement can be obtained in any child node as far as the dual bound is concerned, and so the branching variable should be chosen with some other criterion.

Another, less time-consuming, option is to always ignore variables in $\mathcal{F}_1$ and stick to the variables in $\mathcal{F}_2$. Apart from the obvious computational savings, this choice can be justified by the following argument: if there is a variable in $\mathcal{F}_2$ with a strictly improved dual bound in both children, we will not consider $\mathcal{F}_1 \cap \mathcal{F}_0$ anyway. If there is none, this proves that the global dual bound will not improve independent of the branching decision: at least one of the two children will have the same dual bound as the current node. Therefore, we take the current set of points $\mathcal{S}$ as evidence that variables in $\mathcal{F}_2$ are less likely to become integral than variables in $\mathcal{F}_1$, and so should be given precedence as branching candidates.

Note that using additional points to filter out strong branching candidates is similar in spirit to a strategy called *Nonchimerical Branching* proposed in [FM12], where the optimal solutions of the strong branching LPs (which might have a different objective function value) were used for this purpose. The two strategies have complementary strengths: Nonchimerical Branching does not need to solve any additional LP w.r.t. strong branching, but needs the strong branching LPs to be solved to optimality, because of the usage of the dual simplex. Cloud Branching on the other hand needs additional LPs, but these are in principle simpler (we are fixed to the optimal face), need not be solved to optimality (the primal simplex is used), and do not impose any requirements to the solution of the final strong branching LPs. As such, the two techniques can be easily combined together and might synergize. Moreover, Cloud Branching can be used independent of strong branching, as argued in Section 10.4.

10.6. Computational experiments

For our computational experiments, we used SCIP 3.0.0.1 [Ach09] compiled with SoPlex 1.7.0 [Wun96] as LP solver. The results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20GHz with 12 MB cache and 48 GB main memory, running an openSuse 12.1 with a gcc 4.6.2 compiler. Hyperthreading and Turboboost were disabled. We ran only one job per node to reduce random noise in the measured running time that might be caused by delays if multiple processes share common resources, in particular the memory bus.

We used two test sets of general, publicly available MIP instances: the cor@l test set [Cor10], which mainly contains instances that users worldwide submitted to the neos server [CMM98] and the mmm test set which contains all instances from Miplib3.0 [BCMS98], Miplib2003 [AKM06], and Miplib2010 [KAA$^+$11]. We compare the performance of SCIP when using full strong branching versus a Cloud Branching version of full strong branching as described in the previous section. In particular, we compare to the Cloud Branching variant that only considers variables in $\mathcal{F}_2$ as possible branching candidates. Since we want to explicitly measure the impact of using the cloud for variable selection, we did not exploit the alternative LP optima by any other means, e.g., for cutting plane generation, primal heuristics, reduced cost domain propagation, etc. Results by Achterberg [Ach10, Ach11] indicate that this would be likely to give further improvements on the overall performance. Moreover, we used the default implementation of full strong branching in SCIP, which does not employ the methods suggested in [FM12] (yet). We used a time limit of one hour per instance. All other parameters were left at their default values.

For the mmm test set both, SCIP with Cloud Branching and with Full Strong Branching, solved the same number of instances; for the cor@l test set, one more instance was solved within the time limit when using Cloud Branching. Tables B.20 and B.21 in the appendix show results for all instances which the two variants could solve within the time limit, excluding those which were directly solved at the root node (hence no branching was performed). This leaves 68 instances for mmm and 104 instances for cor@l.

Table 10.1.: comparison of Cloud Branching and Full Strong Branching on MMM and COR@L instances, averages of success rate, cloud points, saved LPs per node, and rate of saved LPs; shifted geometric means of branch-and-bound nodes and running time in seconds

	cloud statistics				cloud branch		strong branch	
Test set	%Succ	Pts	LPs	%Sav	Nodes	Time (s)	Nodes	Time (s)
MMM	12.2	2.19	74.34	21.7	661	68.2	691	72.0
COR@L	40.8	2.71	70.97	51.8	569	118.3	593	157.3

Column %Succ shows the ratio of nodes on which Cloud Branching was run successfully, hence at least one additional cloud point was used. Considering those nodes, columns Pts and LPs depict of how many points the cloud consisted on average and how many strong branching LPs were saved on average per node, i.e., how many integral interval bounds could be found. The Column %Sav shows how many percent of all strong branching LPs could be saved for that instance. When the success rate is zero, these three columns show a dash. For both branching variants, Nodes and Time (s) give the number of branch-and-bound nodes and the computation time (in seconds) needed to prove optimality.

Table 10.1 shows aggregated results. It gives averages over the corresponding numbers (the success rates, the used points, the saved LPs per node and the percentage of overall saved LPs) from Tables B.20 and B.21. Shifted geometric means are shown for the number of branch-and-bound nodes and the computation times.

The results for the MMM test set show a slight improvement of 6 % w.r.t. mean running time and 5 % w.r.t. the mean number of nodes when using Cloud Branching. For COR@L, the mean number of nodes again is slightly larger, about 4 %, when using Full Strong Branching instead of Cloud Branching. The result when comparing computation times is much more explicit: the shifted geometric means differ by about 33 %. As can be seen in Table 10.1, the success rate of Cloud Branching is much better on the COR@L test set than it is on MMM; and even further, on the successful instances, the average ratio of saved LPs is much larger. Taking these observations together explains why the improvement is much more significant for the COR@L test set.

MIP solvers are known to be prone for an effect called performance variability, see also Section 11.1. Loosely speaking, the term performance variability comprises unexpected changes in performance which are triggered by seemingly performance-neutral changes in the environment or the input format. Besides others, performance variability is caused by imperfect tie breaking [KAA^{+}11]. This results in small numerical differences caused by the use of floating point arithmetic which may lead to different decisions being taken during the solution process. A branch-and-bound search often amplifies these effects, which can be similarly observed for all major MIP (and also other optimization) softwares. As a consequence, small changes in performance might in fact be random noise rather than a real improvement or deterioration. This can, e.g., be seen for instance `cap6000` from MMM: Although Cloud Branching was never successful, the number of branch-and-bound nodes alters.[46]. Then again, improvements brought by single components of a MIP solver typically lie in the range of 5 % to 10 %, see, e.g., [Ach07b]. In addition, even if MIP solvers did not exhibit performance variability, we would have the issue of assessing whether the measured difference in performance is statistically significant, a problem common to all empirical studies.

We performed two additional experiments to validate our computational results. First, we ran identical tests on four more copies of the test sets, with perturbed models that were generated by permuting columns and rows of the original problem formulation. This has been introduced in [KAA+11] as a good variability generator that affects all types of problems and all components of a typical MIP solver. Another benefit of this experiment is that it counters overtuning since the evaluation testbed is no longer identical to the development test bed.

As can be expected, the results differ in detail from the default permutation run. For MMM, the improvements w.r.t. computation time were 3 %, 4 %, 4 % and 7 %, and w.r.t. branch-and-bound nodes −3 %, 0 %, 1 % and 2 %. On COR@L, the improvements w.r.t. time were 25 %, 29 %, 32 %, and 42 % and w.r.t. branch-and-bound nodes 3 %, 5 %, 8 %, 14 %. We conclude that Cloud Branching was faster in all five times two experiments (including the original ones) and also consistently reduced the number of branch-and-bound nodes on the COR@L test set. For MMM, it can be argued that the changes are performance neutral w.r.t. the number of branch-and-bound nodes.

As far as the statistical significance of these differences is concerned, we performed randomized tests [Coh95] on the detailed results. Randomized tests are standard non-parametric statistics that do not make any assumptions on the underlying population distributions. According to these tests, the performance difference, both w.r.t. time and nodes, measured on the MMM is *not* statistically significant. As far as COR@L is concerned, the difference in branch-and-bound nodes is again not significant, while the difference in running times is. Note that on heterogeneous test sets such as MMM and COR@L, it is rather difficult to pass statistical significance tests when testing single MIP solver components, because the improvements are almost always in the single digit range and standard test sets are relatively small. In other words, one method might indeed be better than the other, but not by enough to pass the statistical test. We also applied these randomized tests to the

[46]This can be explained by the intermediate cloud LPs being solved – after this, the original LP basis gets installed again and a resolve without simplex iterations is performed. However, solution values, reduced costs etc. might be slightly different than before.

other four copies of the test sets, with consistent results.

Having a closer look at Tables B.20 and B.21, it can be seen that the success rate of Cloud Branching is negligible, i.e., close to zero, for a much higher ratio of the MMM test set than for the COR@L test set. This is also reflected by the much smaller average success rate shown in Table 10.1. This partially explains why the differences on COR@L are much more significant than on MMM: there are simply more instances on which degenerate LP solutions are detected in the PumpReduce step of our algorithm. A reason for this might be that Miplib instances contain more industry-based models with real, perturbed data whereas COR@L has more combinatorial models which often contain symmetries and are prone for degeneracy.

Our interpretation of the given results therefore is that Cloud Branching does not hurt a test set where only few degeneracy is detected but is clearly superior (w.r.t. computing time) on a test set which contains many highly degenerated problems.

10.7. Conclusion

In this chapter, we introduced branching strategies for mixed integer programs that exploit the knowledge of a set of alternative optimal LP solutions. We discussed extensions of Full Strong Branching and Pseudocost Branching that incorporate this idea. Our computational experiments showed that a version of Full Strong Branching that uses cloud intervals is about 30 % faster than default Full Strong Branching on a standard test set with high dual degeneracy. Even the mean number of branch-and-bound nodes could be reduced, though not significantly. We conclude that the presented implementation of Cloud Branching acts as an efficient filtering algorithm for strong branching. Since its first presentation at CPAIOR2013, the idea of Cloud Branching has already been implemented and tested within two commercial MIP solvers [Jen13, Per], namely FICO Xpress and Sulum.

This chapter comprises the most recent work from this thesis. The presented results are very encouraging for further research on Cloud Branching. From the implementation point of view, it could be further exploited that the cloud LPs are solved by the primal simplex algorithm, hence also intermediate solutions will be feasible and could be used as cloud points. Also, procedures like optimality-based bound tightening (see, e.g., [ZG99, CL10]) on general integer variables might produce alternative optimal solutions that could be used as cloud points. A natural next step would be to implement the described modifications on Pseudocost Branching and a development of hybrid strategies such as Reliability Branching that make use of alternative optimal relaxation solutions.

In this chapter, we used multiple optima from a single relaxation as cloud set. In particular in the context of MINLP, employing optima from *multiple, alternative relaxations* seems promising. Whereas for MIP using an

LP relaxation is the “gold standard”, MINLP solvers might be based on LP relaxations (like SCIP), on NLP relaxations (like LINDO API), or on QP relaxations (see [MLK12]). Cloud Branching offers a way to combine information from optimal solutions of these three (or more) relaxations, aiming at branching decisions that are beneficial for all considered relaxations.

Finally, two other improvements of strong branching were suggested recently: Nonchimerical Branching by Fischetti and Monaci [FM12] and a work of Gamrath [Gam14, Gam13] on using domain propagation in strong branching. It will be interesting to see how these ideas combine and whether it will even be possible to make Full Strong Branching competitive to state-of-the-art hybrid branching rules w.r.t. mean running time.

11. Computational results

The leitmotiv of this thesis are heuristic procedures. This second-to-last chapter deals with a topic that is intrinsically heuristic: benchmarking. Whenever we conduct empirical studies, we make heuristic decisions beforehand:

▷ we choose a test set with a limited number of instances,

▷ we choose a few settings from a big parameter space,

▷ we choose thresholds, limits (time, memory) and numerical tolerances,

▷ we choose a way of summarizing our observations, usually by providing some aggregated statistics,

and we aim at making all these choices representative so as to draw general conclusions from the results.

The present chapter constitutes the main computational study of this thesis. We investigate in which respect primal heuristics have an impact on the performance of a MIP and MINLP solver. Therefore, we present computational results on multiple test sets, which we analyze with respect to multiple performance measures and doublecheck the results by statistical tests and control runs on permuted instances.

This chapter is organized as follows. In Section 11.1, we describe how we addressed the above four points of choosing test sets, settings, limits, and performance measures. In Section 11.2, we present computational results for three heterogeneous, academic benchmarks sets of MIPs, MIQCPs, and MINLPs. In Section 11.3, we present computational results for three homogeneous, industry-related test sets of MIPs, MIQCPs, and MINLPs. In Section 11.4, we re-consider the results of the two previous sections, now grouping the test instances by their computational complexity. A summary of our findings is given in Section 11.5.

11.1. Test sets and experimental setup

We chose six different test sets, two of them consisting of MIP instances, two of them of MIQCPs, and two of MINLPs. Three of the test sets are well-established academic benchmark sets: For MIP, this is the MMM test set, which comprises all instances from the three latest MIPLIBs: MIPLIB3.0 [BCMS98], MIPLIB2003 [AKM06], and the benchmark set of MIPLIB2010 [KAA$^+$11]. This test set has also been used in Chapters 4, 7, and 10 of this thesis. For MIQCP, we use the GloMIQO [Glo] test set, see also Chapter 8. For MINLP, we chose the MinlpLib [BDM03], which has also been used in Chapters 7 and 8. Miplib and MinlpLib are long-standing standard benchmark sets for MIP and MINLP, respectively, and probably the most widely used test sets to compare performance of algorithms for the respective problem classes. The GloMIQO test set has been recently introduced [MF13] and represents a compilation of publicly available convex and nonconvex MIQCP instances. It is the largest general benchmark set for MIQCP that we are aware of.

Heterogeneous benchmark sets, such as the mentioned ones, are an important tool to evaluate the average performance of algorithms on a diverse set of problems and to draw conclusions about the general qualities of the tested methods. Naturally, such tests tell you little about the performance on *homogeneous* test sets consisting of instances that all arise from one particular model. This is, however, the standard application for a practitioner who is faced with one single optimization problem, that she or he wants to solve for a set of different input data, e.g., varying per day. Just as researchers and software vendors want to benchmark their code on general test sets such as the ones mentioned above, a practitioner wants to evaluate optimization software for a particular set of problems.

On this account, we decided to perform computational experiments on three more test sets with industrial background. As opposed to the previously described academic benchmark sets, those are very homogeneous, containing one single type of instances per test set.

Industry-related test sets

During his time at Zuse Institute Berlin, the author of this thesis has been an associated member of three research projects that are settled at the interface of industry and academia. The Siemens/ZIB Cooperation[47] is a long term collaboration that addresses the development of general solver software for LP, MIP, and lately also MINLP, with a particular focus on applications that arise in the department "Corporate Technology" of Siemens. "Advanced Solver Technology for SCM"[48] is a joint project of ZIB, the University of Erlangen-Nürnberg and the SAP Germany AG & Co. KG which investigates methods to solve large-scale, numerically challenging supply chain management problems. Finally, Matheon project B20 "Optimization of Gas

[47]`http://www.zib.de/en/projects/current-projects/project-details/article/siemens.html`

[48]`http://www.zib.de/en/optimization/mip/projects/projectdetails/article/neue-technologien-zur-loesung-von-scm-problemen.html`

Transport"[49] is part of the ForNe research cluster[50] (*For*schungskooperation *Ne*tzoptimierung, German for: research cooperation network optimization). ForNe is a joint project of five German universities, two research institutes, and the Open Grid Europe GmbH (OGE), Germany's biggest gas transport system operator; it deals with the complex task of planning and operating nation-wide gas networks. OGE was formerly known as e.on Gastransport.

There is a one-to-one coverage of the three problem classes (MIP, MIQCP, and MINLP) by the three test sets that we compiled from the applications of our three industry cooperation projects. The SAP test set consists of 40 MIPs that come from real-world supply chain management instances. Those instances have been provided by our industry partner and have been used before in [GKM+13] to evaluate the effectiveness of presolving techniques in mixed integer programming.

For MIQCP, we compiled the Siemens test set which contains 27 instances that originate from an internal Siemens project on "power management for smart buildings". The goal of the underlying optimization problem is to generate a best possible schedule for the operation of large office buildings, given forecasts for the base load, the power generation of the building's facilities, the temporal distribution of electric vehicles to be charged, and the pricing scheme for electricity procurement. The nonlinearity is introduced by a representation of the active charging power through nonconvex quadratic constraints. More precisely, these equations model the reactive power which emerges from loading operations that do not use the full capacity of the involved inverters.

For MINLP, we use the ForNe test set of 430 instances that model a nomination validation problem for two different networks of gas pipelines, under many different scenarios of the networks' load. For details, see [KBE+14, PFl+12]. In contrast to the other five test sets, ForNe consists of feasibility problems, not of optimization problems. About 28 % of the instances are infeasible. The model formulation uses indicator constraints and nonconvex absolute power constraints. An *indicator constraint* is given by a binary variable y and an inequality $\alpha^\mathsf{T} x \leqslant \beta$ with $\alpha \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$. It states that $y = 1 \to \alpha^\mathsf{T} x \leqslant \beta$. For the implementation in SCIP and a computational study that solves maximum feasible subsystem problems by models that include indicator constraints, see [Pfe08]. An *absolute power constraint*[51] is, loosely speaking, a power function whose symmetry is interchanged: $x_i|x_i|^{\nu-1} = \gamma x_j$ with $\nu \geq 2$ and $\gamma \in \mathbb{R}_{>0}$. For example if $\nu = 2$ (the relevant case for the gas transport optimization problems) and $\gamma = 1$, the

[49] `http://www.zib.de/en/optimization/mip/projects-long/matheon-b20-optimization-of-gas-transport/article/matheon-b20-optimierung-von-gastransport.html`

[50] `http://www.zib.de/en/projects/current-projects/project-details/article/forne.html`

[51] Absolute power constraints are also referred to as *signed power* constraints or *signpower* constraints since they can be equivalently formulated as $\gamma x_j = \operatorname{sgn}(x_i) x_i^\nu$.

graph of $x_j = x_i|x_i|$ looks like $-x_i^2$ for $x_i \leq 0$ and like $+x_i^2$ for $x_i \geq 0$, i.e., it is an "odd quadratic function". In the considered models, those constraints are used to approximate the differential equations which describe the laws of physics for the pressure loss of gas along the pipes. See Vigerske [Vig12] for details on the implementation of absolute power constraints in SCIP. See Arnold et al. [ABH$^+$14] for an overview on solving MINLPs from energy optimization applications with SCIP.

Computational environment

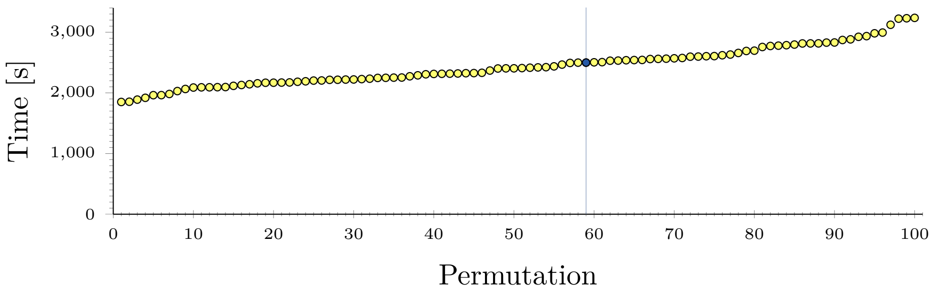
From all test sets, we excluded instances for which SCIP 3.0.2 failed or produced inconsistent results when running with different settings. Further, we removed instances from the MINLPLIB which contain nonlinear expressions that cannot be handled by SCIP, i.e., trigonometric and error functions. Altogether, the six test sets contain 1071 instances. Running the two proposed settings on each of the instances took 761 CPU hours ($\approx$ one month) in total, plus five months for runs on permuted instances (see below) plus many more weeks for additional experiments, e.g., to determine optimal solutions or good bounds for the industrial test sets.

In [AW13], Achterberg and Wunderling emphasized the size of a test set as a central factor for the conclusiveness of experimental results in computational mixed integer programming, using diverse test sets of around 100 to more than 3000 instances in their article.[52] With sizes between 167 and 430 instances, our test sets would be located rather at the lower end of this scale. However, even the smallest of these test sets is still about double the size of the MIPLIB 2010 benchmark set, which has turned out to be *the* default test set to compare the performance of MIP solvers [Mit]. To partly overcome this issue, we conducted experiments on permuted versions of all instances, see the paragraph on performance variability below.

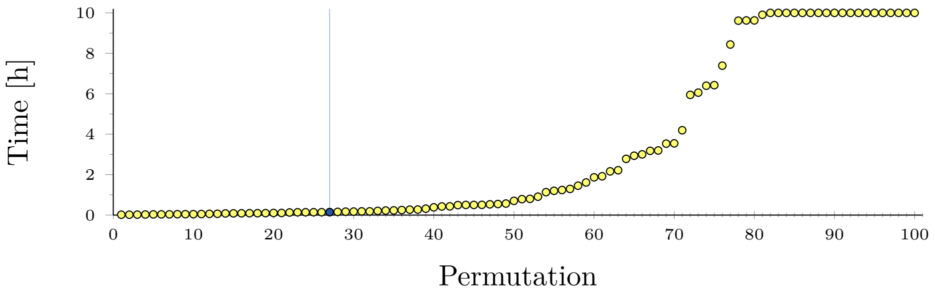
These final experiments compare the performance of SCIP when running with and without primal heuristics for each of the six test sets. We SCIP version 3.0.2, compiled with SOPLEX 1.7.1 [Wun96, Sop] as LP solver, IPOPT 3.11 [WB06, Ipo] as NLP solver, and CPPAD 20120101.3 [Cpp] as expression interpreter for evaluating general nonlinear constraints. Thus, we exclusively used academic software which is available in source code.

The results were obtained on a cluster of 64bit Intel Xeon X5672 CPUs at 3.20 GHz with 12 MB cache and 48 GB main memory, running an OPENSUSE 12.3 with a GCC 4.7.2 compiler. Turboboost was disabled. In all experiments, we ran only one job per node to reduce fluctuations in the measured running times that might be caused by interference between jobs that share resources, in particular the memory bus. Actually, except for the results from Section 6.6, all experiments in this thesis have been carried out on the same cluster.

[52]The majority of these instances is not publicly available; they come from an internal IBM CPLEX model library.



(a) Instance `pg5_34`



(b) Instance `enlight13`

Figure 11.1.: Performance variability: Solution times for 100 permutations of two instances from Miplib 2010 [KAA+11]

For the computational experiments in this chapter, we used a time limit of one hour and a memory limit of 30 GB. The optimality gap threshold was set to 0.0 (as for all other experiments in this thesis), which actually is the default of SCIP. Commercial solvers often use a small positive value, e.g., 0.01 % for Cplex, as an optimality gap threshold for declaring an instance to be solved. For most applications, this is a reasonable approach, but it might be inappropriate in other situations. In the mentioned supply chain management project that the author participated in, such a threshold was pointed out as undesirable by the industry partner. This was due to one of the main problems with using a non-zero optimality gap threshold: the results are not invariant under adding an offset to the objective function.

Performance variability

In two additional experiments, we analyzed the impact of *performance variability* on our results. The term *performance variability* has been introduced by Danna [Dan08] to denote variations in performance measures for the same problem that are caused by seemingly performance-neutral changes in the environment or the input format. Note that this is different from changes in the model formulation, which also may affect performance drastically, even if the changes seem to be minor [AKT08]. Loosely speaking, performance variability comprises unexpected changes in performance.[53]

To address the impact of performance variability, we re-ran the experiments on five more copies of the test sets, using perturbed models. Those were generated by randomly permuting variables and constraints of the original problem formulation, using a different random seed for each of the five copies. This method has been introduced in [KAA$^+$11] as a good variability generator that affects all types of problems and all components of an MINLP solver such as SCIP.

The effect that permuting variables and constraints of an instance can have is depicted in Figures 11.1(a) and 11.1(b). These figures were originally published as part of the author's contribution to the Miplib 2010 paper [KAA$^+$11]. They show the distribution of performance over 100 permutations of two instances from our MIP test set. Each of the yellow dots depicts the performance of one permuted instance when being solved with SCIP 2.0.1.3. They are sorted by non-decreasing solution time. The blue dot corresponds to the performance of the original formulation. Instance `pg5_34`, see Figure 11.1(a), shows a behavior that is often observed: the different solution times are nearly uniformly distributed around the median value, and the original formulation is "somewhere in the middle". Some instances, however, are much more sensitive towards the effects of performance variability, e.g., `enlight13`, which is a pure feasibility problem. The distribution of solution times for this instances can be seen in Figure 11.1(b). We observe a heavily random behavior: the best ten permutations need less than three minutes, but nearly twenty percent of the permuted instances need more than ten hours.

Statistical tests

As argued above, small changes in the starting conditions might have a big impact on the performance of MINLP solvers. As a consequence, if we observe an improvement or deterioration in the average running times of just a few percent, this might be the result of the performance altering "at random" rather than the impact of different preconditions (e.g., solver settings) that we wanted to evaluate. Enlarging the test set by adding permuted instances is one way to address this situation. Using statistical tests to analyze the consistency of the results is another approach. Both are not mutual exclusive and we will use them in combination.

The purpose of statistical tests is to give an indication of how safe it is to draw a certain conclusion from observations on a limited set of samples. Therefore, they estimate the likelihood that a random draw of observations would have given a similar result.

The common practice of statistical tests is to formulate a *null-hypothesis*,

[53] That performance differences resulting from changing, e.g., the operating system, are indeed irritating for users might be anecdotally illustrated by the fact that this is frequently raised as a question or even reported as a bug via the SCIP mailing list.

e.g. "SCIP with setting A performs worse than SCIP with setting B", or "SCIP performs equal with either setting". The null-hypothesis typically is the opposite of the "desired" result. Hence, the task is to show that the null-hypothesis is very unlikely to hold. A *p-value* is computed from the observations (i.e. the computational results). The p-value is an estimate of the probability that the null-hypothesis holds. More precisely, the p-value gives the probability that drawing results at random would give a result that is similarly or even more biased in favor of the alternative of the null-hypothesis. Hence, the smaller the p-value, the better (given that the null-hypothesis states the opposite of the result that we want to test). Note that for small samples sizes (where "small" typically is between 10 and 20), test statistics are compared against critical values from reference tables, instead of computing a p-value.

As an example assume the null-hypothesis to be "SCIP's performance deteriorates when using primal heuristics". If this null-hypothesis gets rejected with a p-value of, say, $p = 0.01$, this means that SCIP most likely benefits from using primal heuristics. According to the statistical test (and the input data), there is a 1 % chance that a random draw of observations would have given a result that is at least as much in favor of one setting (here: using primal heuristics) as the computational results that we used as input for the statistical test.

For nominal data, e.g., whether an instance has been solved to optimality or not, we use a McNemar test [McN47]. For rational data, e.g., the running time to solve an instance, we use a variant of the Wilcoxon signed rank test [Wil45]. Note that many statistical tests assume that the observed data follows a certain distribution, e.g., a normal distribution. We do not believe that this is a valid assumption when considering typical performance measures for MINLP solvers, e.g., the overall running time. The two tests that we use do not make any assumptions on the distribution of the data.

The McNemar test is typically applied to *2×2 contingency tables*. In our example, the row and column data would be "Did SCIP solve the instance within the time limit without using primal heuristics (yes/no)?" and "Did SCIP solve the instance within the time limit with using primal heuristics (yes/no)?". The important information are the two numbers on the counter-diagonal, i.e., the number of instances solved with heuristics but not without, and the number of instances solved without heuristics, but not with. When we denote those by b and c, the McNemar test statistic is $\chi^2 = \frac{(b-c)^2}{b+c}$. The corresponding null hypothesis would be "Both settings lead to an equal number of instances being solved". Under the assumption that χ^2 follows a chi-squared distribution [Hel76], one can compute the probability of the null hypothesis being rejected.

The classical Wilcoxon signed rank test sorts observations by the absolute value of their difference, then assigns ranks from 1 to the number of observations, splits them in two groups depending on whether their actual difference is positive or negative, and finally takes the sums of the rank values of both groups. Loosely speaking, the more different those two sums are, the more likely it is that the one with the larger sum outperforms the one with the smaller sum.

The variant of the Wilcoxon signed rank test that we use takes ratios rather than differences to rank the sums. This corresponds to using geometric means instead of arithmetic means: the important bit of information is the factor by which running times, number of nodes, and so forth change.

As an example, let us assume that we want to analyze the running time to proven optimality. Then, the observations are the running times of all instances which SCIP solved within the time limit in both cases, with and without primal heuristics. All instances for which the running times differ by less than one percent are omitted.

The null-hypothesis is "SCIP with primal heuristics is worse than SCIP without primal heuristics." We rank the instances by the *absolute ratio* of their running times $\max(\frac{t_{\text{w.heur}}}{t_{\text{wo.heur}}}, \frac{t_{\text{wo.heur}}}{t_{\text{w.heur}}})$. Here, $t_{\text{w.heur}}$ and $t_{\text{wo.heur}}$ are the running times with and without heuristics, respectively. Thus, the instance with the smallest absolute ratio has rank one, the instance with the largest absolute ratio the rank N, with N being the number of instances. In case of identical absolute ratios, all tied instances get assigned the average of the ranks they span.

Afterwards, the instances are split into two sets. One comprises the instances for which SCIP without primal heuristics was faster, the other one those for which SCIP with primal heuristics was faster. Let the sums of the ranks of all instances in both sets be $W_{\text{w.heur}}$ and $W_{\text{wo.heur}}$. The Wilcoxon test statistic then reads

$$z = \frac{\min(W_{\text{w.heur}}, W_{\text{wo.heur}}) - \frac{N(N+1)}{4}}{\sqrt{\frac{N(N+1)(2N+1)}{24}}}.$$

and follows (approximately) a normal distribution. That is, z is a standard score from which the p-value can be determined.

11.2. Computational results for academic test sets

As a first test, we ran SCIP 3.0.2 with and without primal heuristics on the three academic benchmark sets, to evaluate the impact of primal heuristics on very heterogeneous model libraries.

The results for MIP can be seen in Tables B.22 and B.23 in the appendix, those for MIQCP in Tables B.24 and B.25, those for MINLP in Tables B.26 and B.27; aggregated results can be found in Table 11.1.

Reading the tables

Each table presents the names of the instances, Nodes, the number of branch-and-bound nodes needed to solve the instance to optimality, and three different running times (in seconds): First, the time needed to find the first feasible

Table 11.1.: Impact of primal heuristics (academic benchmarks)

	all					all feas		all opt	
	feas	opt	obj	t_{heur}	$\frac{P(t_{max})}{t_{max}}$	$time_1$	sols	nodes	time
MMM			(167 instances)			(149 instances)		(116 instances)	
default	160	121	24	11.0 %	8.9 %	9.3	32.2	2 348	78.9
no heur	149	116	3	–	17.2 %	35.3	5.7	3 538	90.9
GloMIQO			(167 instances)			(137 instances)		(118 instances)	
default	156	118	25	10.2 %	9.0 %	3.4	11.2	559	9.8
no heur	138	119	3	–	19.7 %	7.8	4.1	689	10.4
MinlpLib			(240 instances)			(196 instances)		(160 instances)	
default	218	162	47	11.6 %	15.5 %	1.9	8.7	780	8.9
no heur	196	162	0	–	23.5 %	6.3	3.8	1 005	9.1

solution and Total, the running time needed to solve the instance to proven optimality. Obviously, it holds that the times given in Column First are less then or equal to those in Column Total. When the time limit of one hour was hit without solving the instance, this is indicated by the word “limit” in Column Total. In this case, Nodes shows how many branch-and-bound nodes were processed within the hour. The Tables in the appendix further depict two pieces of information about the primal bound. Column Prim Int gives the average primal gap $P(t_{max})/t_{max}$ over one hour, with t_{max}= 3600 seconds and $P(\cdot)$ being the primal integral function from Chapter 3. The Column LP Sols states the number of primal solutions that have been found as integral optima of an LP relaxation of some branch-and-bound node.

The three tables which correspond to test runs with activated primal heuristics, namely Tables B.22, B.24, and B.26, further show three statistics which are specific to primal heuristics. Column Sols gives the number of solutions that have been found by some heuristic (as opposed to the number of LP Sols, which come from the tree). Column Time depicts the sum of the overall running times of all heuristics, Column # shows how many different primal heuristics were called for the particular instance.

Table 11.1 summarizes the results from Tables B.22–B.27. Each line of it represents one combination of a test set and a setting. Columns feas and opt state for how many instances a feasible solution has been found and how many instances have been solved to proven optimality within the time limit, respectively. Column obj depicts for how many instances a setting produced a primal solution that was at least 10 % better than for the respective other setting. Column t_{heur} shows which percentage of running time has been spent for primal heuristics in shifted geometric mean. Column $\frac{P(t_{max})}{t_{max}}$ gives the average primal gap, as defined by the primal integral[54]. Two statistics are given for the set of instances for which both settings found a feasible solution

[54]Note that the average primal integral is a multiple of the average primal gap.

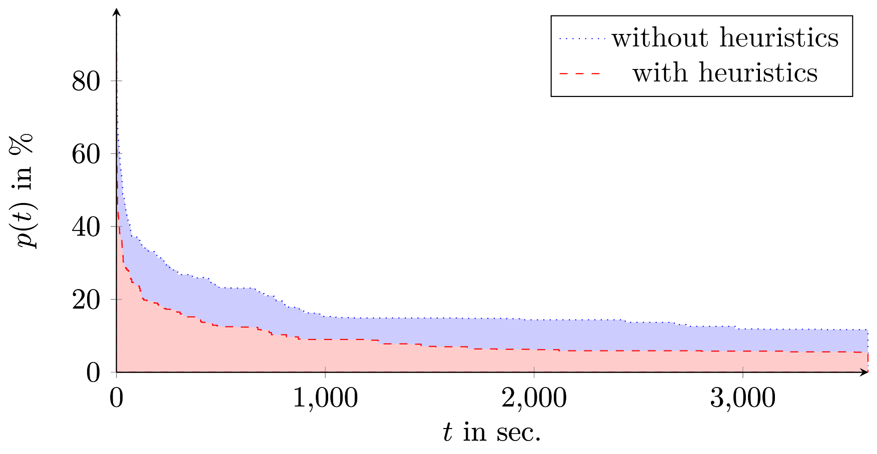


Figure 11.2.: Course of the primal gap when running SCIP with and without primal heuristics on MMM test set

(shown in the double-column all feas): time_1, the mean running time for finding a first feasible solution and sols, the mean number of primal solutions per instance that were found with a certain setting. Finally, for all instances which have been solved to optimality by either setting (all opt), we give the shifted geometric means of the number of branch-and-bound nodes and the overall running time to proven optimality. For the computation of the shifted geometric mean, we used a shift of 100 for the number of branch-and-bound nodes, 10 for all times, and 1 for the number of solutions and the heuristic time ratio $\mathsf{t}_{\mathsf{heur}}$.

Figures 11.2, 11.3, and 11.4 show the evolution of the primal gap as a function over time. The red dashed line corresponds to the average primal gap function, when running SCIP in default mode, with primal heuristics activated. The red shaded area corresponds to the average primal integral of this setting. Accordingly, the blue dotted line and the blue shaded (plus the red shaded) area correspond to the average primal gap function and the average primal integral when running SCIP without heuristics.

Evaluation of the results

From Table 11.1, we see that for all three academic benchmark sets, the majority of the performance indicators speak in favor of primal heuristics.

Let us first consider the three measures that simply "count" instances: feas, opt, and obj. When using primal heuristics, there are more instances for which a feasible solution could be found within the time limit, 160:149 for MIP, 156:138 for MIQCP, 218:196 for MINLP. The number of instances solved to proven optimality, however, is hardly affected: it increases by five for the MMM test set, decreases by one for GloMIQO and stays constant for MinlpLib. Nevertheless, the quality of the incumbent solution at the time limit clearly benefits from the use of primal heuristics. Summing up over all

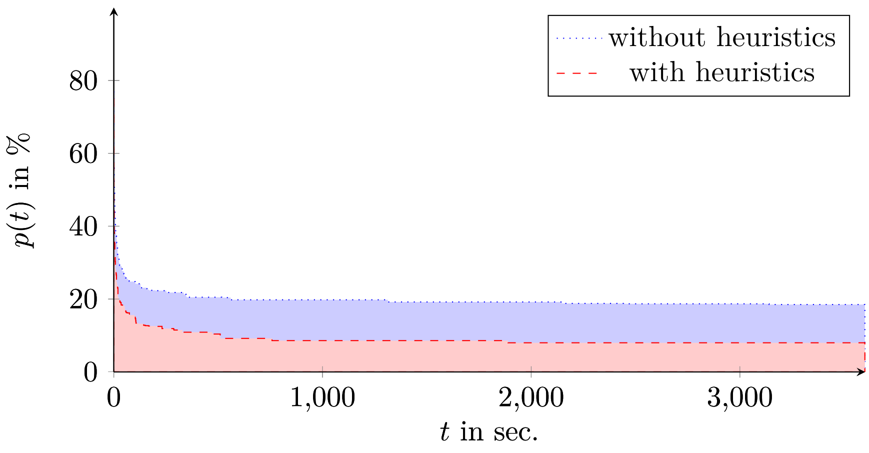


Figure 11.3.: Course of the primal gap when running SCIP with and without primal heuristics on GloMIQO test set

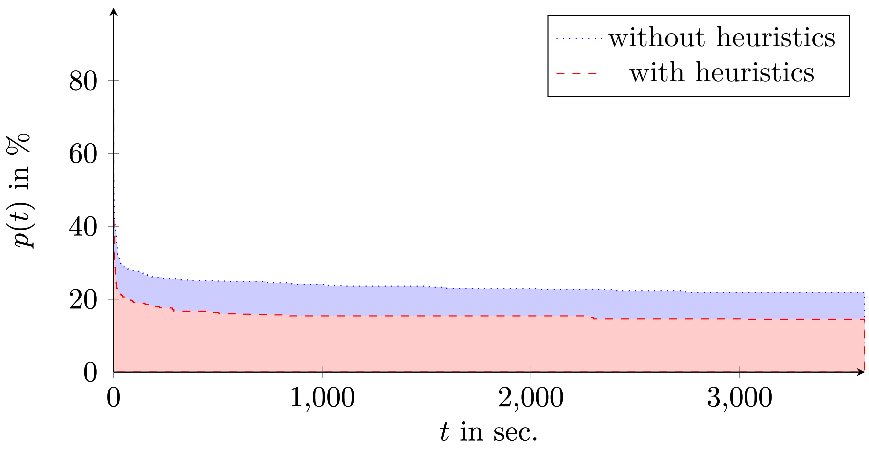


Figure 11.4.: Course of the primal gap when running SCIP with and without primal heuristics on MinlpLib test set

three test sets, there are 96 cases for which the incumbent's objective value is better when using heuristics and only six for which it deteriorates. These four measures emphasize that primal heuristics mainly have an impact on the primal side of the problem.

In our results, the number of instances that can be solved within the time limit hardly depends on primal heuristics; but for those instances which cannot be solved, primal heuristics very often lead to better solutions at termination. This can be seen as an indicator that applying primal heuristics is particularly worthwhile for hard instances which are not expected to be solved to proven optimality within a reasonable amount of time – which is a typical situation for real-world applications.

Besides the primal bound at termination, the time to find a first solution and the primal integral are the two performance measures with the largest differences. When disabling primal heuristics, the time to prove feasibility increases to 380 % (MIP), 229 % (MIQCP), and 331 % (MINLP) of the values for a run with enabled heuristics. The average primal integral amounts to 193 %, 218 %, and 151 % of the default values when switching off heuristics. For all three test sets it holds that the average primal gap drops quickly in the very beginning, flattening out and staying nearly constant after a few minutes, see Figures 11.2, 11.3, and 11.4. Further, the lines corresponding to the setting with heuristics are below the lines corresponding to the setting without heuristics for the whole time period, with relative distances that do not change much over time. This indicates that the results for the primal integral are rather independent of the chosen time limit.

Figures 11.5, 11.6, and 11.7 visualize the impact of primal heuristics on the number of nodes needed to find a first feasible solution. The charts show one bar for each instance and each of the two settings. The height of a bar indicates after how many percent of the nodes the first feasible solution was found. Red and blue bars show the performance without and with primal heuristics, respectively. For each setting, the instances have been sorted by the measured percentage in non-decreasing order. Therefore, a read and a blue bar at the same position do not necessarily correspond to the same instance.

For all three test sets, we observe the blue line of "with heuristics" bars clearly lies right of the red "without heuristics" bars. Having a closer look at the blue bars, we observe an "ℓ-shape": for most of the instances, the first solution is either found very early (mostly at the root node) or not until the very end. Note here, that the set of instances for which the ratio is 100 % compiles many different cases: pure feasibility instances, infeasible instances, instances which get solved at the root, and instances for which no solution is found before the time limit.

Not surprisingly, much more solutions are found when using primal heuristics, see Column sols of Table 11.1. Note that some heuristics, e.g., the rounding heuristics from Chapter 4 might produced solutions that are worse than the current incumbent – whereas for a run without heuristics, every

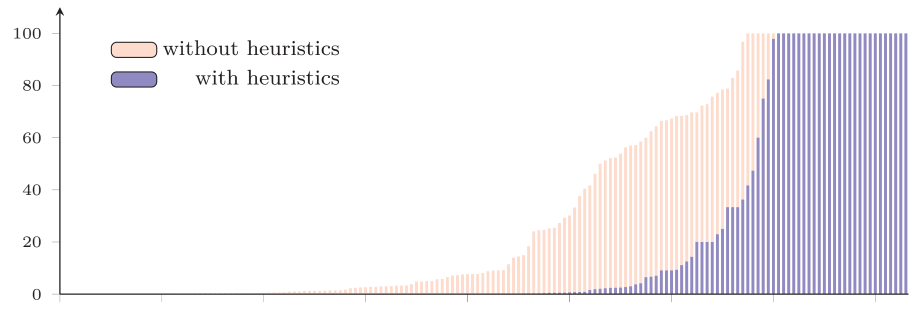


Figure 11.5.: Percent of nodes to first solution: mmm test set

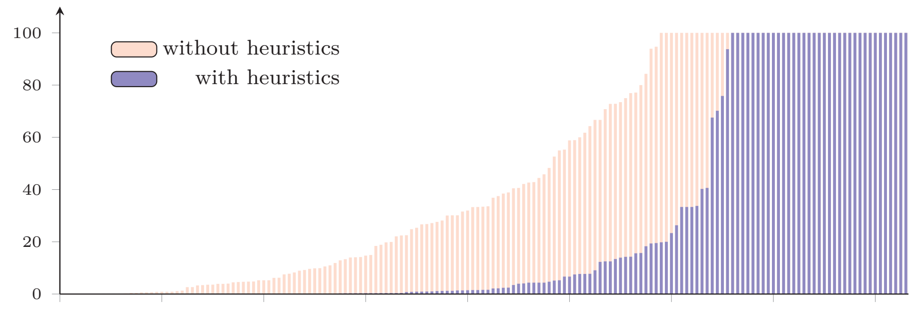


Figure 11.6.: Percent of nodes to first solution: GloMIQO test set

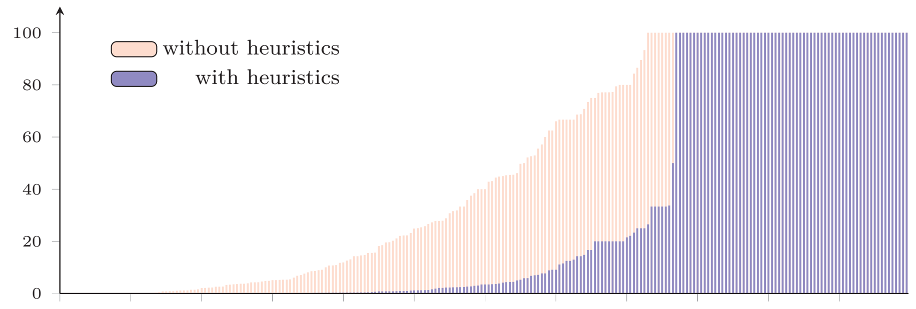


Figure 11.7.: Percent of nodes to first solution: MinlpLib

solution will be incumbent at the time when it is found. The MMM test set features the largest mean numbers of solution for both settings and at the same time the largest ratio between them: 33.2 solutions per instance with heuristics versus 5.7 without, nearly a factor of six. The mean percentage of running time that is spent in primal heuristics, see Column $\mathsf{t_{heur}}$ of Table 11.1, is surprisingly similar for all problem classes: 11.0 % for MIP, 10.2 % for MIQCP up to 11.6 % for MINLP.

Despite of this significant overhead, the mean total running time, probably the most noticed performance measure, improved in all cases. As argued in Chapter 3, the improvements could be expected to be much less than for the primal integral or the running time to a first feasible solution. They ranged from 2 % for the MINLP test set via 6 % for MIQCP to 15 % for our MIP test set, which is consistent to previous results, see in particular Chapter 3.

The savings in the number of branch-and-bound nodes were much bigger: 50 % for the MMM test set and around 25 % for the two nonlinear benchmark sets. Since branch-and-bound nodes are the prime cause of a solver's memory consumption, this is an important observation for parallel MIP and MINLP solving. Koch et al. conjecture in [KRS12] that for future exa-scale supercomputers "memory may become more of a bottleneck resource" since "the memory per core will rather decrease." Smaller tree sizes further lead to a reduction of the message passing overhead of parallel MIP solvers with distributed memory [SAB+12, SAB+13]. In this light, savings of 25 % to 50 % of branch-and-bound nodes constitute a significant improvement, even if they are accompanied only be a small reduction in CPU time.

In this section, we compared two settings of SCIP, default and without primal heuristics, with respect to seven different performance measures: number of instances for which a feasible solution is found, number of instances for which optimality is proven, the primal integral, time to find the first solution, number of overall solutions, number of branch-and-bound nodes, time to proven optimality. For the first two, we presented the results in absolute numbers, for the third one, we gave an average, for the others we used shifted geometric means. Figure 11.8 visualizes the results from Table 11.1 by presenting the relative difference of the two settings w.r.t. these seven performance measures. We computed the relative difference in such a way that a positive value always corresponds to an improvement:

▷ For measures for which larger numbers are better (feas, opt, sols), we took the value of the default setting minus the value of the setting without heuristics and divided the difference by the value of the setting without heuristics.

▷ For measures for which smaller numbers are better ($\mathsf{P(t_{max})/t_{max}}$, $\mathsf{time_1}$, time, nodes), we took the value of the setting without heuristics minus the value of the default setting and divided the difference by the value of the default setting.

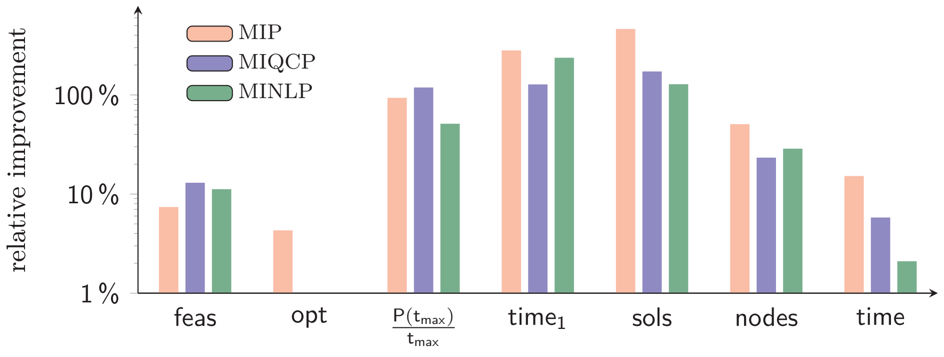


Figure 11.8.: Bar chart illustrating the relative difference between SCIP running with and without primal heuristics w.r.t. seven performance measures. Scaled such that positive values correspond to improvements by using heuristics.

Figure 11.8 uses a logarithmic scale. Thus, we omitted relative differences below one percent from the presentation. For the same reason, the only degradation (−0.8 % instances solved to optimality for the MIQCP test set) is not shown in the diagram. Figure 11.8 illustrates that, on all three academic test sets and for six out of seven performance measures, primal heuristics improve the performance by 2 % to 462 %.

Statistics of primal heuristics

As can be seen in Table B.22 in the appendix, the instance `ex9` is the only instance in the MMM test set for which only one heuristic got called: the *trivial* heuristic finds the optimal solution, whose optimality gets proven by the root LP relaxation. For all other instances, at least five primal heuristics, for the majority (139 of 167) of instances more than ten, are applied. The peak is achieved by four instances, `bell3a`, `bell5`, `momentum2`, and `rocII-4-11`, for which SCIP tries 19 different primal heuristics. There is no direct connection between the computational complexity and these numbers: `bell5` solves in less than a second, `momentum2` hits the time limit. For the two nonlinear test sets, up to 23 heuristics were called. Similarly to the MIP case, more than ten different heuristics were invoked for the majority of instances (138/167 and 192/240).

For the MMM test set, the percentage of time spent in primal heuristics ranges from 0.2 % for `ex9` to 64.4 % for `disctom`; the median is 11.4 %, hence very similar to the shifted geometric mean of 11.0 %. Again, this does not depend on the computational complexity of the instance: of the two instances with the largest amount of time devoted to heuristics, one (`liu`) hits the time limit, the other (`disctom`) solves in 4.5 seconds; among the ten instances with least time spent in heuristics, four solve within less than a minute, four others time out. In the nonlinear case, the whole bandwidth of running time percentage is covered: For `nuclear14`, SCIP spends 0.1 % of its running time in heuristics, for `nuclear49b`, it is 99.6 %; both instances hit the time limit. For the GloMIQO test set, the median is 12.0 %, hence similar to the shifted geometric mean. For the MinlpLib, the median is 14.9 %, thus, a bit larger than the shifted geometric mean from Table 11.1. This can be explained by the larger number of trivial instances[55] in the MinlpLib as compared to mmm or GloMIQO.

Results for permuted instances

We performed an additional experiment to analyze the impact of *performance variability* on our results. Therefore, we considered five copies of every instance, each with a different permutation of the variables and constraints. The results are summarized in Table 11.2.

Table 11.2.: Impact of primal heuristics (academic benchmarks, permuted)

	all					all feas		all opt	
	feas	opt	obj	t_{heur}	$\frac{P(t_{\max})}{t_{\max}}$	time_1	sols	nodes	time
MMM			(835 instances)			(741 instances)		(555 instances)	
default	798	598	125	10.8 %	10.5 %	9.3	34.3	2 189	71.9
no heur	746	565	25	–	17.3 %	36.2	5.7	3 216	80.0
GloMIQO			(827 instances)			(679 instances)		(583 instances)	
default	764	591	98	10.5 %	10.2 %	2.9	11.7	523	10.2
no heur	686	588	11	–	20.1 %	7.4	4.0	623	9.8
MinlpLib			(1182 instances)			(967 instances)		(790 instances)	
default	1096	816	232	11.7 %	14.2 %	1.8	8.4	807	9.1
no heur	982	804	5	–	23.2 %	5.7	3.8	949	8.7

In this experiment, we found that there is a general tendency towards the default setting being slightly worse on the permuted instances than on the original ones, compare Table 11.2 against Table 11.1. The difference in time to optimality on the mmm test set, e.g., is only 12 % instead of 15 %. On the one hand, this might be due to an overtuning effect on the original instance formulations. On the other hand, algorithms that seek to detect certain structures in a problem formulation, such as the Shift-and-Propagate heuristic from Chapter 4 or Undercover from Chapter 8, tend to generally perform worse on randomly permuted instances.

Nevertheless, the tendency and also the magnitudes of the measures and their differences are the same in most cases. Thus, the effects of performance variability seem to vanish in the average. There is one major exception: For

[55] 27 instances of the MinlpLib solve in less than 0.05 seconds and are therefore listed with 0.0 seconds running time in Table B.26. For those instances, the heuristics' total running times often are below the measure tolerance of 0.005 seconds.

the GloMIQO and the MinlpLib test set, the time to optimality increases by 2 % and 3 % when using primal heuristics on permuted instances, whereas it decreased by 2 % and 6 % on the original instances, respectively. As we will see below, the improvements on the original test set fails a standard statistical test and should hence not be considered reliable. This is confirmed by the fact that on permuted instances this improvement disappears. The number of instances that could be solved to proven optimality, however, increases slightly when using primal heuristics on the permuted instances, where it decreased by one on the original GloMIQO instances and did not change on the original MinlpLib instances.

If for each instance, one could automatically choose the permutation which gives the best performance (w.r.t. overall running time) by some kind of oracle, this would improve the mean running time by about 10 % for mmm and GloMIQO. This can be exploited in the ramp-up phase of massively parallel MIP and MINLP solvers, as we showed in [SAB$^+$12, SAB$^+$13]. Speeding up MIP search by exploiting performance variability is also the main driver of recent papers by Carvajal et al. [CAN$^+$14], Fischetti and Monaci [FM14] and Fischetti et al. [FLM$^+$13].

Statistical tests

We further performed statistical tests to analyze the significance of the presented results. As we saw in the previous part, even improvements of up to 6 % on a test set of more than a hundred instances can be questionable.

First, we applied a McNemar test to the three performance criteria from Table 11.1 which count instances: feas, opt, and obj. As null hypotheses we assume that SCIP with and without primal heuristics is equally likely to find a feasible solution, to prove optimality and to terminate with a superior primal bound, respectively.

For all three test sets, the null hypotheses concerning feas and obj get rejected with very small p-values, indicating a large statistical significance and hence a very high probability that the chances to find a feasible solution and to terminate with a better primal bound are affected by primal heuristics. Precisely speaking the assumption that primal heuristics do not affect the number of instances for which feasibility is proven, gets rejected with p-values 0.000 911, 0.000 022 09, and 0.000 002 73 for the mmm, the GloMIQO, and the MinlpLib test set, respectively. The p-values for the assumption that primal heuristics do not affect the solution quality were 0.000 054 06, 0.000 032 16, and less than 0.000 01.[56] Thus, all of the mentioned p-values were orders of magnitude smaller than 0.05 which is a conventional criterion to consider a result to be statistically significant.

The third assumption about primal heuristics not affecting the chance to solve an instance to proven optimality, however, gets only rejected with

[56]We used the statistic calculator of `http://www.socscistatistics.com/pvalues/chidistribution.aspx`, which uses 0.000 01 as a lower precision bound.

statistical significance for the mmm test set, with a p-value of 0.025 347. For GloMIQO and MinlpLib, the p-values are 0.3173 and 0.6171, and thus do not indicate a significance of the result. We conclude that primal heuristics have a clear impact on finding solutions at all and on finding high-quality solutions, but not on proving optimality.

Finally, we performed a variant of the Wilcoxon signed rank test to analyze the statistical significance of the results for the running times for finding a first feasible solution, for the primal integral, and for the running time and the number of branch-and-bound nodes required to prove optimality. For a Wilcoxon signed rank test, the instances of a test set are ordered and ranked by their differences w.r.t. a certain performance measure. In analogy to using (shifted) geometric means instead of averages, we ranked the results not by the absolute differences but by the relative factors of the corresponding performance measure, compare also Chapter 7.5. Further, we excluded instances that showed no or hardly any performance difference (less than 1 %). As null hypotheses, we assume that primal heuristics have no impact on the running time to the first feasible solution, on the overall running time, on the number of branch-and-bound nodes and on the value of the primal integral, respectively.

For the mmm test set, the null hypothesis that primal heuristics do not alter the running time to optimality gets clearly rejected with a p-value of 0.000 48. For the GloMIQO test set and the MinlpLib, however, the Wilcoxon signed rank test does not indicate a significant performance improvement on the overall running time; the p-values are 0.418 005 and 0.423 479, respectively. This is not too surprising, given that the running time improvement on the original instances was much larger for mmm (15 %) than for the other two test sets.

The Wilcoxon test leaves hardly any doubt in the significance of using heuristics for the time needed to find a first feasible solution, for the number of branch-and-bound nodes, and for the primal integral. The null hypotheses get rejected with p-values of less than 0.000 01 for all three criteria for each of the three academic test sets. We conclude that primal heuristics are extremely relevant for the time to find a first solution, but not so much for the time to prove optimality. This finding aligns well with the above observation that there is a very significant difference for the question whether SCIP can find a feasible solution but not for whether it can prove optimality within the time limit. Further, the significance level of the primal integral was in between the one for time to the first solution and the one for time to optimality for all test sets which is in line with the findings of Chapter 3 that the latter two are extrema, whereas the primal integral balances time and quality of solutions.

Further findings

In the course of preparing our final experiments, we performed some additional test runs with larger time limits and different settings. During those, we could find a feasible solution with objective function value −1.150 151 360 for the instance `nuclear10b`, for which, to the best of our knowledge, no primal bound has been reported in the literature before. The `nuclear` instances are a notoriously hard to solve group of MINLP models that arise from an application in which reloading patterns for the fuel rods of nuclear power plants should be designed [QvGH+99]. Three out of four instances that are listed on the MinlpLib webpage [Min] as “without an integer solution” are from this group. Our solution for `nuclear10b` was found by the rens heuristic that has been introduced in Chapter 7 of this thesis.

11.3. Computational results for industrial test sets

As the second main experiment, we ran SCIP 3.0.2 with and without primal heuristics on the three test sets of real-world instances, coming from projects with industry partners, to evaluate the impact of primal heuristics on very homogeneous sets of practically relevant models.

Table 11.3.: Impact of primal heuristics (real-world instances)

	all					all feas		all opt	
	feas	opt	obj	t_{heur}	$\frac{P(t_{max})}{t_{max}}$	$time_1$	sols	nodes	time
SAP			(40 instances)			(17 instances)		(16 instances)	
default	36	16	19	13.1 %	22.2 %	3.9	4.7	137	4.4
no heur	17	16	0	–	57.6 %	4.7	2.0	259	5.6
Siemens			(27 instances)			(0 instances)		(0 instances)	
default	19	0	19	7.4 %	37.1 %	–	–	–	–
no heur	0	0	0	–	100.0 %	–	–	–	–
ForNe			(430 instances)			(198 instances)		(293 instances)	
default	221	315	23	6.8 %	25.8 %	3.0	1.0	84	3.2
no heur	198	293	0	–	36.0 %	30.7	1.0	1 684	18.7

We present results for MIP, MIQCP, and MINLP, modeling problems arising in supply chain management, power management for smart buildings, and nomination validation for gas networks, respectively. Our findings for the SAP MIP instances are shown in Tables B.28 and B.29 in the appendix, those for the Siemens MIQCPs in Tables B.30 and B.31, those for the MINLPs from the ForNe project in Tables B.32 and B.33; aggregated results can be found in Table 11.3.

Reading the tables

As in the previous section, each table lists the instances, the number of branch-and-bound Nodes, the time to find a First solution, the Total running time, the average primal gap (Prim Int), and the number of relaxation solutions (LP Sols). A timeout is indicated by the term "limit" in Column Total. For Tables B.28, B.30, and B.32, Columns Sols and Time give the number of heuristic solutions and the ratio of the heuristics' running time to the total running time, respectively.

Table 11.3 summarizes and averages the results from Tables B.28–B.33. Columns feas and opt show the number of instances for which feasibility and optimality could be proven, respectively, obj depicts the number of instances with a better primal bound, $\mathsf{t}_{\mathsf{heur}}$ the percentage of running time spent for primal heuristics, $\frac{\mathsf{P}(\mathsf{t}_{\mathsf{max}})}{\mathsf{t}_{\mathsf{max}}}$ the average primal gap. For the instances proved feasible by both SCIP versions, time_1 depicts the mean running time for proving feasibility, and sols gives the mean number of solutions. For the instances solved to optimality by both versions, the number of branch-and-bound nodes and the overall running time are presented. Figures 11.9, 11.10, and 11.11 show the evolution of the average primal gap over time; the red dashed line corresponds to the default runs with primal heuristics, the blue dotted line to the runs with deactivated heuristics.

Evaluation of the results

Results from homogeneous test sets tend to be more extreme than the ones on heterogeneous sets, as we will see in the following. Further, it is our experience that for industry projects there often are no large test sets available, but rather a "handful" of important instances – often enough only one single MIP/MINLP model of "the problem" which needs to be solved. Thus, the SAP and Siemens test sets of 40 and 27 instances are rather small compared to the academic test sets from the previous section. Then again, the ForNe collection is the largest single test set that we consider in this thesis. It is common to all three test sets, that the emphasis lies on the primal bound. The ForNe test set actually contains pure feasibility instances, without an objective function.

Analogously to the academic experiments, the majority of the performance indicators speak in favor of primal heuristics, see Table 11.3. The most extreme difference can be observed for the Siemens test set, for which not even one instance can be proven to be feasible within one hour when primal heuristics are deactivated. As a consequence, the all feas and all opt subsets are empty and therefore, many of the used performance indicators could not be evaluated in a meaningful manner. The only real information we can gain is that with primal heuristics, for 19 instances a feasible solution could be found. Thus, most evaluations in the remainder of this part will only be made for the other two test sets.

For the 19 instances of the Siemens instance for which a feasible solution has been found, we conducted an additional experiment in which we changed the gap limit of SCIP to 1 %. Of course, it is always a speculation what percentage of gap might by considered acceptable or sufficient in practice for models that cannot be solved to optimality within the desired time. The additional experiment revealed that for all 19 instances, a solution with less than 1 % optimality gap could be found. This took 69.0 seconds and 2425 branch-and-bound nodes in shifted geometric mean.

For the SAP test set, we see from Table 11.3 that with primal heuristics, there are more than twice as many instances (36 vs. 17) for which a feasible solution could be found; the number of instances solved to optimality, however, is identical. When disabling primal heuristics, we observe a deterioration in the mean running time to optimality of 27 % (5.6 seconds vs. 4.4). This value is larger than for any of the academic test sets and astonishingly, even larger than the improvement in the time to find a first feasible solution on the SAP test set, which is 20 % (4.7 seconds vs. 3.9). For the number of branch-and-bound nodes and the average primal integral, we see factors of 1.9 and 2.6, respectively. These are again larger than for any of the academic test sets. Some of these results, however, should be taken with a grain of salt, as we will see from the control runs on permuted instances.

To analyze the results for the ForNe test set, it is important to note that none of the instances has an objective function, and some are infeasible. Thus, this is the only test set, for which the number of instances solved to proven optimality, see Column opt, is greater than the number of instances for which a feasible solution could be found, see Column feas. The latter set of instances is a proper subset of the former one, since finding a feasible solution is equivalent to solving the instance. The set of instances solved to optimality, however, additionally contains those instances for which infeasibility could be proven within the given time limit. As corollaries, the all opt subset is larger than the all feas set, the average number of solutions in the all feas set is exactly 1.0 for both settings, and the Columns First and Total will always be equal in Tables B.32 and B.33.

There are 23 (of 430) more instances for which the default setting with primal heuristics found a feasible solution, but only 22 more which were solved (since the setting without heuristics proved infeasibility for one instance for which the default setting timed out). Looking at the mean time needed to find a solution and to solve an instance (the second one also including infeasible instances), we see some of the most extreme results of this thesis: The SCIP version with heuristics is 10.2 and 5.7 times faster, respectively. For the mean number of branch-and-bound nodes, the factor is nearly 20! It remains to be said, however, that on the subset of infeasible instances, SCIP with heuristics was a factor 1.2 slower and needed 12 % more nodes. Nevertheless, the benefit of heuristics on feasible instances is much larger than their burden on infeasible instances.

From the plots of the primal integral for SAP and Siemens, see Figures 11.9 and 11.10, we see that the curve corresponding to the setting with-

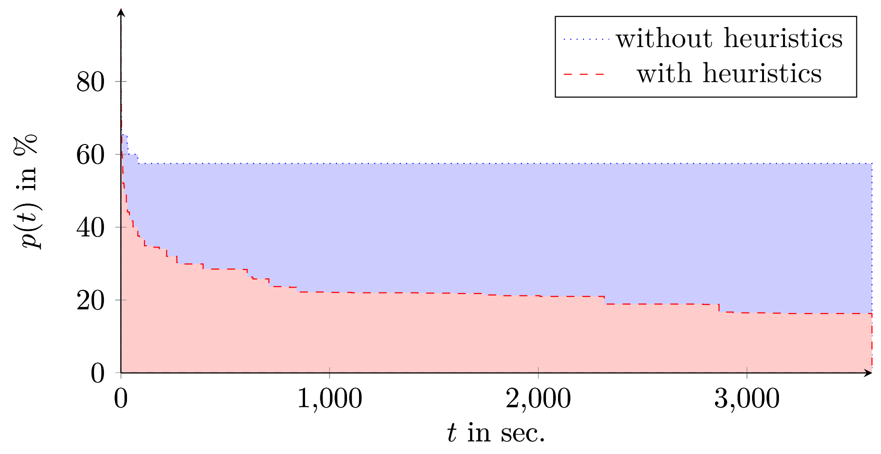


Figure 11.9.: Course of the primal gap when running SCIP with and without primal heuristics on SAP test set

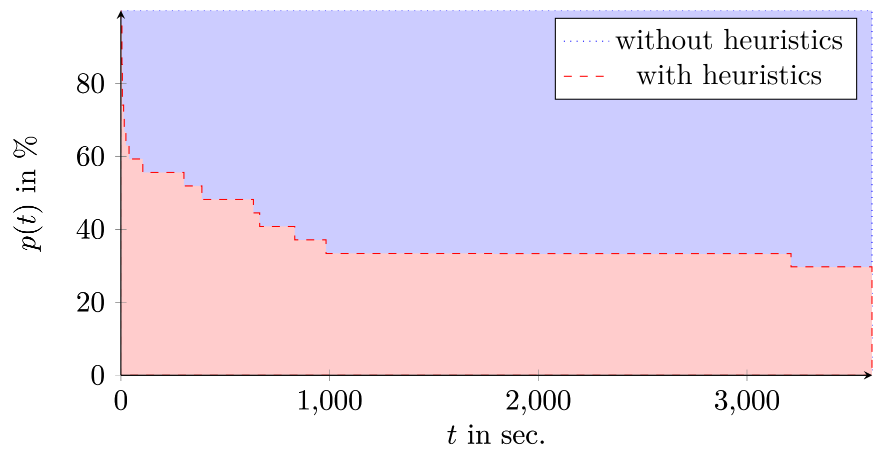


Figure 11.10.: Course of the primal gap when running SCIP with and without primal heuristics on Siemens test set

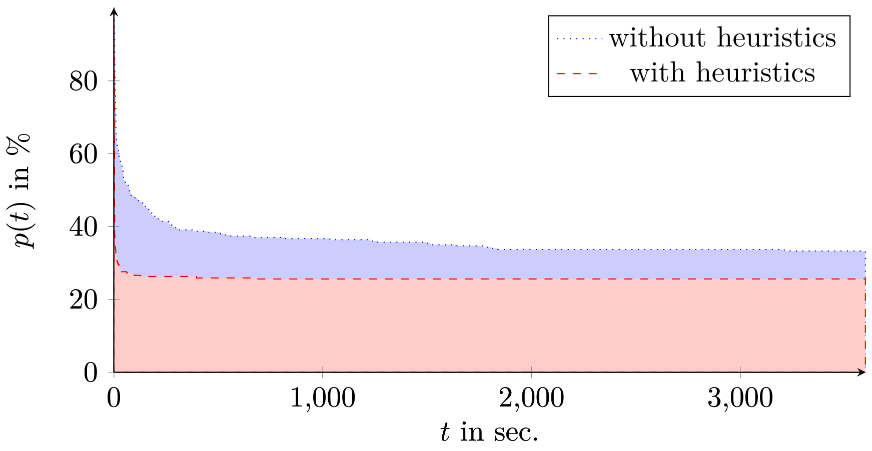


Figure 11.11.: Course of the primal gap when running SCIP with and without primal heuristics on ForNe test set

out heuristics stalls quickly at a constant level, whereas the curve corresponding to the setting with heuristics slowly decreases over time, especially for the SAP test set. For the ForNe test set, see Figure 11.11, it is rather the other way around: the setting with heuristics stalls after about one minute (at a low level, though) and the setting without heuristics catches up over time. Thus, for the default setting, a solution is either found immediately or not within the time limit. This observation, together with the lessons learned about performance variability, motivates an approach of running many copies of a solver with different permutation seeds for a short time rather than running one copy on the original formulation for a long time. In particular for models for which any solution will be optimal, this is a promising strategy, see below the results for permuted instances of the ForNe test set.

Generally, the instances from the industrial test sets were harder than those from the benchmark sets. This can be argued from the facts that a lower percentage of instances could be solved to proven optimality within the time limit and that the average primal integral is larger. We also observed that the magnitudes of the improvements caused by using primal heuristics were larger than for the academic benchmark sets. We see this as a further indicator that primal heuristics are more helpful the harder the problems are, consider in particular the performance on the Siemens test set.

In this section, we compared two settings of SCIP, default and without primal heuristics, with respect to seven different performance measures: number of instances for which a feasible solution is found, number of instances for which optimality is proven, the primal integral, time to find the first solution, number of overall solutions, number of branch-and-bound nodes, time to proven optimality. Figure 11.12 visualizes the results from Table 11.3 by presenting the relative difference of the two settings w.r.t. these seven performance measures. We computed the relative difference in such a way that

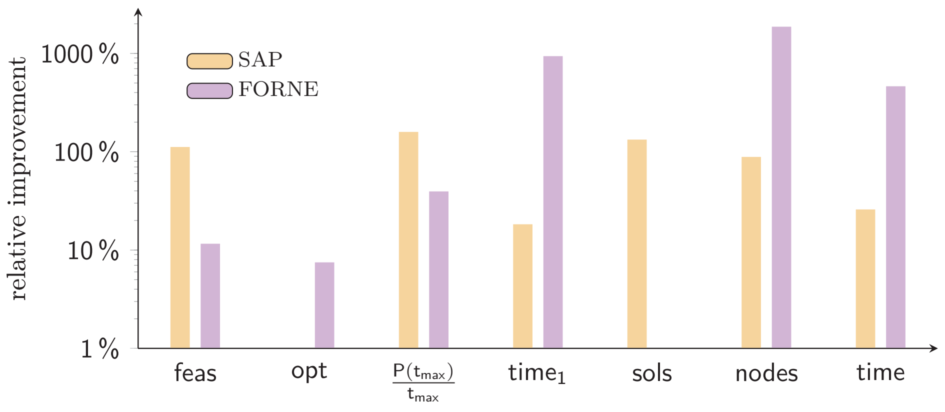


Figure 11.12.: Bar chart illustrating the relative difference between SCIP running with and without primal heuristics w.r.t. seven performance measures. Scaled such that positive values correspond to improvements by using heuristics.

a positive value always corresponds to an improvement, see Section 11.2.

Figure 11.12 uses a logarithmic scale, relative differences below one percent are omitted from the presentation. Further, we excluded the results for the Siemens test set, since most of the values for the setting without heuristics are zero and hence a relative comparison is not meaningful. Figure 11.12 illustrates that for the other two industrial test sets, six out of seven performance measures show improvements by 7.5 % to 1900 % when using primal heuristics.

Statistics of primal heuristics

The Siemens test set was the most homogeneous w.r.t. number of primal heuristics being called; for all of the instances this number was between 17 and 21, see Table B.30. For the ForNe test set, there were 40 very easy instances that were solved in presolving and for which only one heuristic was called. For all other instances, at least seven heuristics and at most 19 were called. All numbers between 7 and 19 occured several times, 18 and 19 being the most common ones, see Table B.32. This corresponds to the full portfolio of primal heuristics without improvement heuristics – recall that the ForNe test set consists of pure feasibility problems. For the SAP test set, the primal heuristics statistics were rather diverse, too: there were a couple of instances for which only two heuristics were called, some for which 19 were called, see Table B.28.

The Siemens test set is by many means the most homogeneous test set that we considered in this thesis. Interestingly, the amount of time spent in primal heuristics ranges from 2.4 % to 24.1 %, hence there is a factor of ten involved. This occurs despite the facts that all instances time out, hence the base value of 3600 seconds is constant and that the number of called heuristics is very similar, see the previous paragraph. Also, the larger percentages are not achieved by one single outlier, but by the sum of many single heuristics. For instance `pmbrp-096-05-09-1`, which achieves the maximum value of 24.1 % time spent in primal heuristics, this is caused by five diving heuristics which take between 100 and 200 seconds each, being called between 300 and 600 times each. For the instance `pmbrp-108-10-12-1`, which achieves the minimum value of 2.4 % time spent in primal heuristics, each diving heuristic takes less than one second in total, each being called around 170 times. There were two important aspects contributing to this difference. Firstly, the diving depth for `pmbrp-108-10-12-1` is only six nodes on average, whereas for `pmbrp-096-05-09-1` it is nearly 100. Secondly, for `pmbrp-096-05-09-1` one iteration of the simplex algorithm takes about three times longer as compared to `pmbrp-108-10-12-1`. Both instances needed around six simplex iterations on average to re-optimize a diving LP.

The SAP MIPs are very diverse w.r.t. their size, the time it takes to solve and consequently also w.r.t. the percentage of time that is spent in primal heuristics. For all instances from the `snp-004*` group primal heuristics take less than ten percent, for two of them actually only 0.2 %. The maximum is achieved by `snp-003` with 98.9 %. Here, the reason is that RENS consumes almost all the time for solving its sub-MIP. For the ForNe test set, every instance that times out takes less than 2 % of running time for primal heuristics. The other instances mainly solve within a few seconds, the bandwidth of time spent in heuristics ranges from 0.6 % to 79.5 %, wherefrom two thirds of the instances consumed between 5 and 25 % of the total in heuristics.

Results for permuted instances

As in the previous section, we performed an additional experiment to analyze the effect of *performance variability*, considering test sets that contain five permuted copies of each instance. The results are summarized in Table 11.4.

Table 11.4.: Impact of primal heuristics (real-world instances, permuted)

	all					all feas		all opt	
	feas	opt	obj	t_{heur}	$\frac{P(t_{max})}{t_{max}}$	$time_1$	sols	nodes	time
SAP			(200 instances)			(85 instances)		(80 instances)	
default	180	80	95	11.7 %	24.6 %	4.3	5.0	118	4.6
no heur	85	80	0	–	58.8 %	6.3	2.8	282	7.7
Siemens			(135 instances)			(0 instances)		(0 instances)	
default	102	0	102	5.7 %	35.0 %	–	–	–	–
no heur	0	0	0	–	100.0 %	–	–	–	–
ForNe			(2150 instances)			(966 instances)		(1401 instances)	
default	1088	1524	119	6.3 %	27.5 %	2.7	1.0	76	3.1
no heur	981	1417	13	–	37.6 %	36.7	1.0	2 099	22.1

For the Siemens and the ForNe test set, the results on permuted instances were very similar to the results on the original formulations. The success rate on Siemens was a bit higher: there were 102 instances for which a feasible solution could be found, compared to 19 on the original set. Consequently, the average primal integral reduced slightly. Unlike for the original test set, the ForNe set showed some (precisely: 13 out of 2150) cases for which the settings without heuristics could find a feasible solution, but the setting with heuristics could not. The difference between time and nodes needed to solve the instances is even a bit more extreme than on the original test set. The factors are 7.1 and 27, compared to 5.7 and 20, for time and branch-and-bound nodes, respectively.

With a portfolio approach of running the setting with heuristics on six permutations, one finds feasible solutions for 243 instead of 222 instances of the ForNe test set. That the usage of permutations increases the number of solved instances by about 10 % is a desirable result in two senses. First, it shows that running the solver on different permutations in parallel can be beneficial. Such an approach has been implemented within the LaMaTTO++ framework [KBE$^+$14, Lam] for solving MINLPs on networks [KBE$^+$14]. Second, that the improvement is "only" 10 % shows that the cases for which the performance of the solver is particularly bad on the original problem formulation are relatively rare.

For the SAP test set, all numbers that count instances, namely `feas`, `opt`, and `obj`, were simply the fivefold of the numbers in Table 11.3. The mean time needed to prove optimality with the default setting is very similar on the original and the permuted SAP test set. The mean time for the setting without heuristics, however, is much worse on permuted instances: it is 7.7 seconds on the permuted test set, see Table 11.4, and 5.6 on the original, see Table 11.3. This can be explained mainly by the small size of the original set – there are 16 instances that got solved to optimality – which makes it vulnerable to the impact of outliers. To see this, we need to compare all six permutations, including the original, amongst each others. It turns out that four of them behave rather similar, with mean run times of 6.9, 7.0, 7.6, and 8.2 seconds. One of the permuted test sets is an outlier in the negative direction with a mean running time of 9.3 seconds. The original set, however, is an outlier in the positive direction with a mean running time of 5.6 seconds.

This is mainly due to one single instance, `snp-004-02`, which gets solved in 38.5 seconds in the original formulation, whereas the permuted versions take between 128.2 and 700.1 seconds. When removing this single instance, the pairwise difference between the running times for the permuted test sets is at most 15 %, as compared to 40 % with this instance. This experiment demonstrates the added value of running tests on permuted test sets to reduce the impact of outliers, in particular when the original test set is very small.

Statistical tests

For SAP and Siemens, the Wilcoxon signed rank test and the McNemar tests for the number of feasible instances and the better objective at the time limit all confirmed statistical significance, i.e., the respective null-hypotheses that primal heuristics would not have an impact on these performance measures all got rejected with very small p-values. For the other three criteria, the number of samples for which these differed (applying the exclusion rules explained in Section 11.2) was too small.

For the ForNe test set, however, all six statistical tests passed, i.e., null-hypotheses got rejected. Note that since these are pure feasibility problems, many of the performance measures are equivalent or the only difference is whether infeasible instances are included or not. For instance, the difference of the number of instances with feasible solution is the same as for the number of instances with a better primal bound at termination and the primal integral is a multiple of the time to first solution. Nevertheless, the results for the ForNe test set are the most extreme ones speaking in favor of primal heuristics, as mentioned above, and it comes at no surprise that the statistical tests confirm the significance of these results.

Further findings

Again, we performed some additional test runs with larger time limits and different settings to determine good bounds on the optimal objective value. It turned out that with a ten hour time limit and with emphasis setting for heuristics set to "aggressive", SCIP could find a feasible solution for all instances of the Siemens test set. Still, none of the instances could be solved to proven optimality, but in all cases the final primal-dual gap was below 0.5 %.

For the ForNe test set, we could decide the feasibility status for 417 of 431 instances in additional experiments with different settings, limits and SCIP versions. Originally, the status was known only for 309 instances.

11.4. All instances grouped by complexity

In this section, we take a second glance at the results of the previous two sections. So far, we have grouped our test instances by the problem class they belong to and by the application they arise from. Now, we re-analyze our results, grouping the test instances by the enumerative effort that they take to solve. More precisely, we break up the union of our six test sets into groups of *easy*, *medium*, *hard* and *timeout* instances.

We call an instance *easy*, when the worse of the two SCIP settings (with and without primal heuristics) solves it to proven optimality in at most 100 branch-and-bound nodes. If it takes more than 100, but less than 10,000 nodes to solve it with the worse setting, we call the instance *medium*. Instances that can be solved to optimality, but not with less than 10,000 nodes, are called *hard*. Finally, we classify instances as *timeout* when SCIP times out with at least one setting.

This is in the spirit of a recent computational study by Achterberg and Wunderling [AW13], who subdivided their test set into "brackets" which are defined by the time Cplex needs to solve them to proven optimality. Note that they defined the brackets such that they are proper subsets of each other, whereas by our definition, the groups of *easy*, *medium*, *hard* and *timeout* instances are pairwise disjoint.

Table 11.5.: Impact of primal heuristics (all instances, grouped by the number of branch-and-bound nodes of the worse setting).

	all					all feas		all opt	
	feas	opt	obj	t_{heur}	$\frac{P(t_{\max})}{t_{\max}}$	time_1	sols	nodes	time
all			(983 instances)			(630 instances)		(639 instances)	
default	732	669	144	9.2 %	17.8 %	4.3	4.3	322	9.4
no heur	631	642	6	–	30.1 %	24.2	2.3	1 739	21.6
easy			(195 instances)			(136 instances)		(195 instances)	
default	136	195	0	7.1 %	0.2 %	0.8	3.1	9	1.6
no heur	136	195	0	–	0.3 %	5.1	1.5	13	1.6
medium			(264 instances)			(232 instances)		(264 instances)	
default	232	264	0	19.3 %	1.7 %	3.2	3.5	197	7.0
no heur	232	264	0	–	2.1 %	8.5	2.2	828	9.6
hard			(180 instances)			(175 instances)		(180 instances)	
default	175	180	0	13.5 %	1.1 %	5.1	9.1	7 554	57.2
no heur	175	180	0	–	6.5 %	54.4	4.4	96 135	195.4
timeout			(344 instances)			(87 instances)		(0 instances)	
default	189	30	144	4.6 %	47.2 %	3.7	2.1	–	–
no heur	88	3	6	–	77.9 %	10.4	0.8	–	–

Table 11.5 shows how various performance measures differ by the complexity of the test instances. For a description of the table columns, please see the explanation of Table 11.1 on page 177. Figure 11.13 visualizes the results from Table 11.5 by presenting the relative difference of the two settings w.r.t. seven performance measures.

For some performance measures, the difference between both settings is similar among all groups, e.g. the number of found solutions raised by around 100 % when using primal heuristics. For others measures, the importance of employing primal heuristics clearly rises with the "hardness" of the instances. Where for easy instances, running SCIP without primal heuristics leads to increases of only 5 % in running time and 31 % in the number of branch-and-bound nodes, for medium instances these are 38 % and 318 %, respectively. For hard instances, abstaining from primal heuristics causes a 241 % deterioration in running time and an increase of 1173 % in the number of branch-and-bound nodes needed to solve the instance to proven optimality.

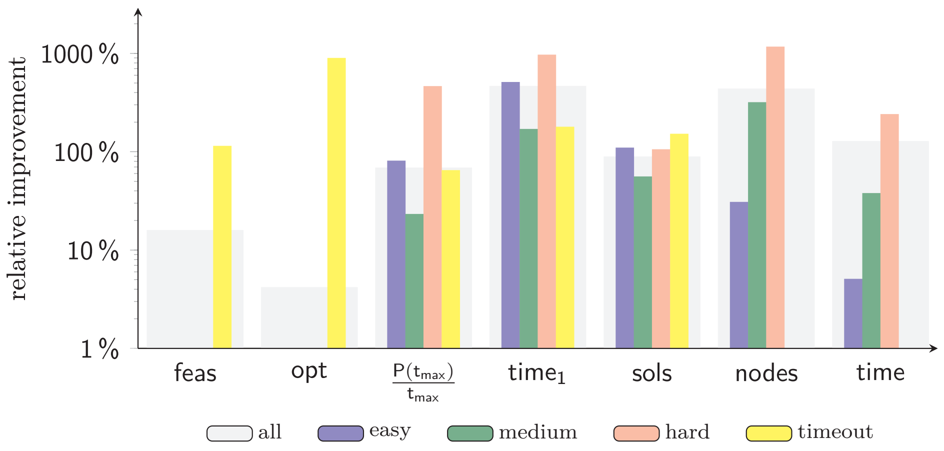


Figure 11.13.: Bar chart illustrating the relative difference between SCIP running with and without primal heuristics w.r.t. seven performance measures. Scaled such that positive values correspond to improvements by using heuristics. Grouped into easy, medium, hard, and timeout instances by the number of branch-and-bound nodes

Long story short: Primal heuristics hardly affect instances that are easy to solve anyway, but they are crucial for hard instances.

Among the 314 instances which timed out with at least one setting, there were 189 instances for which primal heuristics could find at least one feasible solution, without heuristics there were only 88. For nearly one half of the instances, 144 times, the primal bound at termination was at least 10 % better when using primal heuristics. Only six times, the setting without heuristics led to a better primal bound. There were 30 which could be solved when using primal heuristics, but not without. Only three times it was the other way around. We conclude that for instances that are too hard to be solved within a given time limit, employing primal heuristics is very likely to be beneficial.

11.5. Conclusion

In the introduction of this chapter, we pointed out that computational results always depend on heuristic decisions by the researcher who conducts them. Concerning the choice of parameter settings, we opted for simplicity and broke our experiments down to one crucial question: “Are primal heuristics beneficial for MIP and MINLP solvers and if yes, in which respect?” Concerning the choice of instances, we opted for comprehensiveness and analyzed results for six different test sets, accompanied by results on permuted instances, and a final evaluation on the instances grouped by their computational complexity.

These experiments led to some distinct conclusions. We saw that primal heuristics give rise to considerable improvements in various measures, namely the number of found solutions, the time to find a first solution, the quality of the primal bound at termination, the primal integral (see Chapter 3), and the number of branch-and-bound nodes required to prove optimality. These results were consistent over all test sets that we considered, on the original formulations as well as on permuted formulations. Except for pure feasibility problems, they did not affect the number of instances solved to optimality. The mean time to optimality improved significantly for two industry-related test sets and the academic MIP benchmark set. We further observed that primal heuristics are especially beneficial for hard instances and problems that cannot be solved within a given time limit. For instances that need more than 10,000 branch-and-bound nodes to be solved to optimality, the mean running time of SCIP reduced by a factor of more than three when using heuristics.

In a nutshell, the presented results give a strong indication that the particular value of primal heuristics lies in obtaining high quality solutions fast. The more difficult the problem, the more important is the deployment of heuristics.

12. Conclusion

A direct conclusion of the preceding eleven chapters is that heuristic algorithms are ever-present in global solvers for MINLP. In this thesis, we presented diverse new contributions, all of them of heuristic nature, to computational MIP and MINLP. We evaluated the impact of the suggested methods on the performance of the global solver SCIP, of which the author has been a main developer for the past seven years.

In an attempt to break down 199 pages to a single paragraph, we summarize the main findings of this thesis as follows:

▷ Primal heuristics are an essential component of nowadays MIP and MINLP solvers; their benefit, however, becomes apparent only on a second glance: rather than speeding up overall computational time by a tremendous amount, they support a quick convergence of primal and dual bounds (see in particular Chapters 3 and 11). This feature is of major importance for most practitioners that use MI(NL)P solvers.

▷ The harder a problem is, the more important are primal heuristics. On hard and unsolved instances, primal heuristics showed the largest improvements in our computational study, see Chapter 11.4.

▷ Primal heuristics are most successful when they follow a "fast fail" strategy; it is crucial to rapidly identify situations with low chances of success and bail out quickly.

▷ Large neighborhood search is not only beneficial for improvement strategies; we introduced two successful LNS-based start heuristics in Chapters 7 and 8.

The implementations of all methods described in Chapters 3, 4, and 6 to 10 are part of the SCIP standard distribution and can thereby be accessed in source code. The majority of them are nowadays employed by default in SCIP.

In [BR07], Bixby and Rothberg stretched the importance of academic research on MIP solution technology for the commercial development of MIP software and its application in industry. They note that the progression of MIP towards a general, powerful technology was "enabled by methods and theory that have been available in the academic literature" and that for long time simply "these techniques had not made it into commercial MIP software, despite their effectiveness". In this light, it is an honor to see that some of the methods described in this thesis have already been re-implemented within several MIP and MINLP software packages [BDV14, Jen13, Ach, Per, Bon, Cbc], commercial and non-commercial.

We strongly believe that computational MINLP is about to experience a rise comparable to that of computational MIP over the last twenty years. In this course, heuristic algorithms can be expected to play a key role when black-box solvers shall be enabled to solve large-scale MINLP instances of practical relevance. The increasing number of publications on primal heuristics for MINLP over the last five years indicates that the journey has already begun.

Figure 12.1.: Solutions can sometimes be found at surprising places.
(Originator of image: Michaela Keinert, Berlin)

Bibliography

[AA95] Erling D. Andersen and Knut D. Andersen. Presolving in linear programming. *Mathematical programming*, 71:221–245, 1995. Cited on page 44.

[AB07] Tobias Achterberg and Timo Berthold. Improving the Feasibility Pump. *Discrete Optimization*, Special Issue 4(1):77–86, 2007. Cited on pages 22, 23, 61, 62, 65, 70, and 99.

[AB09] Tobias Achterberg and Timo Berthold. Hybrid branching. In Willem-Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009*, volume 5547 of *Lecture Notes in Computer Science*, pages 309–311. Springer Berlin Heidelberg, May 2009. Cited on pages 139, 142, 149, and 152.

[ABCC95] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. Finding cuts in the TSP (A preliminary report). Technical Report 95-05, DIMACS, 1995. Cited on pages 149 and 151.

[ABCC07] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, USA, 2007. Cited on page 151.

[ABE$^+$02] Ernst Althaus, Alexander Bockmayr, Matthias Elf, Michael Jünger, Thomas Kasper, and Kurt Mehlhorn. SCIL – symbolic constraints in integer linear programming. In *Algorithms – ESA 2002*, pages 75–87, 2002. Cited on page 138.

[ABH$^+$00] Karen Aardal, Robert E. Bixby, Cor A. J. Hurkens, Arjen K. Lenstra, and Job W. Smeltink. Market split and basis reduction: Towards a solution of the Cornuéjols-Dawande instances. *INFORMS Journal on Computing*, 12(3):192–202, 2000. Cited on pages 18 and 157.

[ABH12] Tobias Achterberg, Timo Berthold, and Gregor Hendel. Rounding and propagation heuristics for mixed integer programming. In Diethard Klatte, Hans-Jakob Lüthi, and Karl Schmedders, editors, *Operations Research Proceedings 2011*, pages 71–76. Springer Berlin Heidelberg, 2012. Cited on pages 40 and 42.

[ABH$^+$14] Thomas Arnold, Timo Berthold, Stefan Heinz, Stefan Vigerske, René Henrion, Martin Grötschel, Thorsten Koch, Caren Tischendorf, and Werner Römisch. A jack of all trades? solving stochastic mixed-integer nonlinear constraint programs. In Peter Deuflhard, Martin Grötschel, Dietmar Hömberg, Ulrich Horst, Jürg Kramer, Volker Mehrmann, Konrad Polthier, Frank Schmidt, Christof Schütte, Martin Skutella, and Jürgen Sprekels, editors, MATHEON – *Mathematics for Key Technologies*, volume 1 of *EMS Series in Industrial and Applied Mathematics*, pages 135–146. European Mathematical Society, 2014. Cited on page 172.

[ABKW08] Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. Constraint integer programming: A new approach to integrate CP and MIP. In Laurent Perron and Michael A. Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008*, volume 5015 of *Lecture Notes in Computer Science*, pages 6–20. Springer Berlin Heidelberg, May 2008. Cited on page 138.

[Ach] Tobias Achterberg. personal communication. Cited on pages 38, 60, 81, 115, 157, and 200.

[Ach04] Tobias Achterberg. SCIP - a framework to integrate constraint and mixed integer programming. Technical Report 04-19, Zuse Institute Berlin, 2004. `http://www.zib.de/Publications/abstracts/ZR-04-19/`. Cited on page 40.

[Ach07a] Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, 2007. Cited on pages 7, 100, 104, 124, 138, 139, 140, 141, 142, and 153.

[Ach07b] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007. Cited on pages 11, 13, 16, 22, 42, 43, 44, 45, 75, 83, 124, 138, 139, 142, 149, 150, 151, 152, 153, and 166.

[Ach09] Tobias Achterberg. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. Cited on pages 5, 7, 15, 20, 33, 62, 75, 87, 123, 138, and 164.

[Ach10] Tobias Achterberg. LP basis selection and cutting planes. Presentation slides from MIP 2010 workshop in

Atlanta. `http://www2.isye.gatech.edu/mip2010/program/program.pdf`, 2010. Cited on pages 66, 102, 159, and 164.

[Ach11] Tobias Achterberg. LP relaxation modification and cut selection in a MIP solver. US patent, US20110131167, 2011. Cited on pages 66, 102, 159, and 164.

[ACR03] David Applegate, William J. Cook, and André Rohe. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003. Cited on page 43.

[AHY04] Ionuţ D. Aron, John N. Hooker, and Tallys H. Yunes. SIMPL: A system for integrating optimization techniques. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2004*, volume 3011 of *Lecture Notes in Computer Science*, pages 21–36, 2004. Cited on page 138.

[AKM05] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005. Cited on pages 149, 150, 151, and 152.

[AKM06] Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006. Cited on pages 55, 96, 101, 144, 150, 164, and 169.

[AKT08] Tobias Achterberg, Thorsten Koch, and Andreas Tuchscherer. On the effect of minor changes in model formulations. ZIB-Report 08-29, Zuse Institute Berlin, 2008. Cited on page 173.

[AL97] Emile Aarts and Jan K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, Inc., 1997. Cited on page 80.

[ALL10] Kumar Abhishek, Sven Leyffer, and Jeff Linderoth. Filmint: An outer approximation-based solver for convex mixed-integer nonlinear programs. *INFORMS Journal on Computing*, 22(4):555–567, 2010. Cited on page 7.

[AMF95] Ioannis P. Androulakis, Costas D. Maranas, and Christodoulos A. Floudas. αbb: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995. Cited on page 20.

[APT03] Edoardo Amaldi, Marc E. Pfetsch, and Leslie E. Trotter, Jr. On the maximum feasible subsystem problem, IISs and IIS-hypergraphs. *Mathematical Programming*, 95(3):533–554, 2003. Cited on page 140.

[AS14] Roberto Amadini and Peter J. Stuckey. Sequential time splitting and bounds communication for a portfolio of optimization solvers. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming (CP 2014)*, volume 8656 of *LNCS*, pages 108–124. Springer, 2014. Cited on page 38.

[AW13] Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In Michael Jünger and Gerhard Reinelt, editors, *Facets of Combinatorial Optimization – Festschrift for Martin Grötschel*, pages 449–481. Springer Berlin Heidelberg, 2013. Cited on pages 31, 149, 172, and 196.

[Bal71] Egon Balas. Intersection cuts—a new type of cutting planes for integer programming. *Operations Research*, 19(1):19–39, 1971. Cited on page 156.

[Bal75] Egon Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8:146–164, 1975. Cited on pages 10 and 97.

[Bas07] Oliver Bastert. Xpress-MP performance. Poster presentation at MIP2007 conference in Montreal. `http://www.crm.umontreal.ca/MIP2007/pdf/Bastert-poster.pdf`, 2007. Cited on page 31.

[BBC$^+$08] Pierre Bonami, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, Francois Margot, Nicolas Sawaya, and Andreas Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5:186–204, 2008. Cited on pages 7, 20, 67, and 118.

[BC11] Daniel Baena and Jordi Castro. Using the analytic center in the feasibility pump. *Operations Research Letters*, 39(5):310–317, 2011. Cited on pages 61 and 65.

[BCD$^+$01] Egon Balas, Sebastiàn Ceria, Milind Dawande, Francois Margot, and Gábor Pataki. OCTANE: A new heuristic for pure 0-1 programs. *Operations Research*, 49(2):207–225, 2001. Cited on pages 10, 23, 39, 42, and 97.

[BCLM09] Pierre Bonami, Gérard Cornuéjols, Andrea Lodi, and Francois Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119(2):331–352, 2009. Cited on pages 2, 61, 67, 68, 69, 70, 72, 98, 99, and 118.

[BCMS98] Robert E. Bixby, Sebastiàn Ceria, Cassandra M. McZeal, and Martin W.P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998. Cited on pages 55, 96, 101, 144, 164, and 169.

[BDM03] Michael R. Bussieck, Arne S. Drud, and Alexander Meeraus. MINLPLib – a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003. Cited on pages 21, 75, 96, 102, 135, and 170.

[BDV14] Michael R. Bussieck, Steven P. Dirkse, and Stefan Vigerske. PAVER 2.0: An open source environment for automated performance analysis of benchmarking data. *Journal of Global Optimization*, 59(2–3):259–275, 2014. Cited on pages 38 and 200.

[Bea79] Evelyn M.L. Beale. Branch and bound methods for mathematical programming systems. In Peter L. Hammer, Ellis L. Johnson, and Bernhard H. Korte, editors, *Discrete Optimization II Proceedings*, volume 5 of *Annals of Discrete Mathematics*, pages 201–219. Elsevier, 1979. Cited on page 24.

[BEE$^+$11] Natashia L. Boland, Andrew C. Eberhard, Faramroze G. Engineer, Matteo Fischetti, Martin W. P. Savelsbergh, and Angelos Tsoukalas. Boosting the feasibility pump, 2011. `http://www.optimization-online.org/DB_FILE/2012/01/3322.pdf`. Cited on pages 61 and 65.

[BEET12] Natashia L. Boland, Andrew C. Eberhard, Faramroze G. Engineer, and Angelos Tsoukalas. A new approach to the feasibility pump in mixed integer programming. *SIAM Journal on Optimization*, 22(3):831–861, 2012. Cited on pages 61 and 66.

[BEG$^+$11] Thierry Benoist, Bertrand Estellon, Frédéric Gardi, Romain Megel, and Karim Nouioua. LocalSolver 1.x: a black-box local-search solver for 0-1 programming. *4OR*, 9(3):299–316, 2011. Cited on page 26.

[Ber06] Timo Berthold. Primal heuristics for mixed integer programs. Diploma thesis, Technische Universität Berlin, 2006. Cited on pages 2, 13, 22, 23, 24, 40, 41, 42, 62, 79, 80, 82, 83, 85, 96, 98, 99, 103, 104, 114, 121, and 153.

[Ber08] Timo Berthold. Heuristics of the branch-cut-and-price-framework SCIP. In Jörg Kalcsics and Stefan Nickel, editors, *Operations Research Proceedings 2007*, pages 31–36. Springer-Verlag, 2008. Cited on page 153.

[Ber13] Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013. Cited on pages 5, 29, and 90.

[Ber14] Timo Berthold. RENS – the optimal rounding. *Mathematical Programming Computation*, 6(1):33–54, 2014. Cited on pages 5, 23, 70, 80, 82, 95, 118, and 121.

[Bes06] Christian Bessiere. Constraint propagation. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, chapter 3, pages 29–83. Elsevier, 2006. Cited on page 44.

[BFG$^+$00] Robert E. Bixby, Mary Fenelon, Zonghao Gu, Ed Rothberg, and Roland Wunderling. MIP: Theory and practice – closing the gap. In Michael J. D. Powell and Stefan Scholtes, editors, *Systems Modelling and Optimization: Methods, Theory, and Applications*, pages 19–49. Kluwer Academic Publisher, 2000. Cited on pages 2, 7, 22, 24, 31, and 151.

[BFL07] Livio Bertacco, Matteo Fischetti, and Andrea Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, Special Issue 4(1):63–76, 2007. Cited on pages 23, 61, 62, 64, and 99.

[BFM98] Ralf Borndörfer, Carlos Ferreira, and Alexander Martin. Decomposing matrices into blocks. *SIAM Journal on Optimization*, 9(1):236–269, 1998. Cited on page 150.

[BFS10] Timo Berthold, Thibaut Feydy, and Peter J. Stuckey. Rapid learning for binary programs. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Proc. of CPAIOR 2010*, volume 6140 of *LNCS*, pages 51–55. Springer Berlin Heidelberg, June 2010. Cited on pages 5, 23, 137, 143, 144, 146, and 154.

[BFZ10] Egon Balas, Matteo Fischetti, and Arrigo Zanette. On the enumerative nature of Gomory's dual cutting plane method. *Mathematical Programming*, 125(2):325–351, 2010. Cited on page 158.

[BG10] Timo Berthold and Ambros M. Gleixner. Undercover – a primal heuristic for MINLP based on sub-MIPs generated by set covering. In Pierre Bonami, Leo Liberti, Andrew J. Miller, and Annick Sartenaer, editors, *Proceedings of the EWMINLP*, pages 103–112, April 2010. Cited on pages 2 and 117.

[BG12] Pierre Bonami and João Gonçalves. Heuristics for convex mixed integer nonlinear programs. *Computational Optimization and Applications*, 51:729–747, 2012. Cited on pages 2, 24, 61, 67, 70, 76, 80, 85, 86, 98, and 118.

[BG13] Timo Berthold and Ambros M. Gleixner. Undercover branching. In Vincenzo Bonifaci, Saverio Caminiti, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Proc. of SEA 2013*, volume 7933 of *LNCS*, pages 212–223. Springer Berlin Heidelberg, 2013. Cited on pages 135, 154, and 158.

[BG14] Timo Berthold and Ambros M. Gleixner. Undercover: a primal MINLP heuristic exploring a largest sub-MIP. *Mathematical Programming*, 144(1–2):315–346, 2014. Cited on pages 2, 5, 117, and 158.

[BGG$^+$71] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O. Vincent. Experiments in mixed-integer programming. *Mathematical Programming*, 1:76–94, 1971. Cited on pages 149, 150, and 151.

[BGHV12] Timo Berthold, Ambros M. Gleixner, Stefan Heinz, and Stefan Vigerske. Analyzing the computational impact of MIQCP solver components. *Numerical Algebra, Control and Optimization*, 2(4):739–748, 2012. Cited on pages 88, 96, and 101.

[BGKV12] Andreas Bley, Ambros M. Gleixner, Thorsten Koch, and Stefan Vigerske. Comparing MIQCP solvers to a specialised algorithm for mine production scheduling. In Hans Georg Bock, Xuan Phu Hoang, Rolf Rannacher, and Johannes P. Schlöder, editors, *Modeling, Simulation and Optimization of Complex Processes*, pages 25–39. Springer Berlin Heidelberg, 2012. Cited on page 117.

[BH15] Timo Berthold and Gregor Hendel. Shift-and-propagate. *Journal of Heuristics*, 21(1):73–106, 2015. Cited on pages 5 and 39.

[BHdV10] Oliver Bastert, Benjamin Hummel, and Sven de Vries. A generalized Wedelin heuristic for integer programming. *INFORMS Journal on Computing*, 22(1):93–107, 2010. Cited on page 25.

[BHPV11] Timo Berthold, Stefan Heinz, Marc E. Pfetsch, and Stefan Vigerske. Large neighborhood search beyond MIP. In Luca Di Gaspero, Andrea Schaerf, and Thomas Stützle, editors, *Proceedings of the 9th Metaheuristics International Conference (MIC 2011)*, pages 51–60, 2011. Cited on pages 2, 5, 79, 80, 86, 87, 94, 96, 98, and 118.

[BHV11] 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. Springer New York, 2011. Cited on page 20.

[Bix] Robert E. Bixby. personal communication. Cited on page 115.

[BK98] Alexander Bockmayr and Thomas Kasper. Branch-and-infer: A unifying framework for integer and finite domain constraint programming. *INFORMS Journal on Computing*, 10(3):287–300, 1998. Cited on page 138.

[BKL12] Pierre Bonami, Mustafa Kilinç, and Jeff Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In Jon Lee and Sven Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 1–39. Springer New York, 2012. Cited on page 20.

[BKR98] Rainer E. Burkard, Michael Kocher, and Rüdiger Rudolf. Rounding strategies for mixed integer programs arising from chemical production planning. *The Yugoslav Journal of Operations Research*, 8(1), 1998. Cited on page 42.

[BLL$^+$09] Pietro Belotti, Jon Lee, Leo Liberti, Francois Margot, and Andreas Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods & Software*, 24:597–634, 2009. Cited on pages 5, 7, 20, 61, 75, 118, and 129.

[BM80] Egon Balas and Clarence H. Martin. Pivot and complement–a heuristic for 0-1 programming. *Management Science*, 26(1):86–96, 1980. Cited on pages 24 and 39.

[BM86] Egon Balas and Clarence H. Martin. Pivot-and-Shift: A Heuristic for Mixed Integer Programming. GSIA, Carnegie Mellon University, August 1986. Cited on page 25.

[BMW75] A.L. Brearley, G. Mitra, and H.P. Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical Programming*, 8:54–83, 1975. Cited on pages 44, 45, and 65.

[Bon] Bonmin (Basic Open-source Nonlinear Mixed INteger programming). `https://projects.coin-or.org/Bonmin`. Cited on pages 20, 115, 129, and 200.

[Bor98] Ralf Borndörfer. *Aspects of Set Packing, Partitioning, and Covering.* PhD thesis, TU Berlin, 1998. Cited on page 46.

[BR07] Robert Bixby and Edward Rothberg. Progress in computational mixed integer programming—a look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37–41, 2007. Cited on pages 15 and 199.

[BRG09a] Robert E. Bixby, Ed Rothberg, and Zonghao Gu. MIP Heuristics. Presentation slides from CO@Work 2009 conference in Berlin. `http://co-at-work.zib.de/berlin2009/downloads/2009-09-25/2009-09-25-0900-BB-MIP-2.pdf`, 2009. Cited on pages 69 and 82.

[BRG09b] Robert E. Bixby, Ed Rothberg, and Zonghao Gu. The Gurobi Solver V1.0. Presentation slides from CO@Work 2009 conference in Berlin. `http://co-at-work.zib.de/berlin2009/downloads/2009-09-29/2009-09-29-0915-BB-Gurobi.pdf`, 2009. Cited on page 103.

[BS13] Timo Berthold and Domenico Salvagnin. Cloud branching. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 28–43. Springer Berlin Heidelberg, 2013. Cited on pages 5, 66, and 149.

[BSW04] Egon Balas, Stefan Schmieta, and Christopher Wallace. Pivot and shift—a mixed integer programming heuristic. *Discrete Optimization*, 1(1):3–12, 2004. Cited on pages 23, 25, and 39.

[BT70] Evelyn Martin Lansdowne Beale and John A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In John Lawrence, editor, *OR69: Proceedings of the Fifth International Conference on Operations Research*, pages 2447–2454. Tavistock Publications, London, 1970. Cited on page 154.

[BT76] Itshak Borosh and L. Bruce Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proceedings of the American Mathematical Society*, 55(2):299–304, 1976. Cited on pages 2 and 14.

[BV10] Michael R. Bussieck and Stefan Vigerske. MINLP solver software. In James J. Cochran, Louis A. Cox, Pinar Keskinocak, Jeffrey P. Kharoufeh, and J. Cole Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2010. Online publication. Cited on page 20.

[CAN$^+$14] Rodolfo Carvajal, Shabbir Ahmed, George Nemhauser, Kevin Furman, Vikas Goel, and Yufen Shao. Using diversification, communication and parallelism to solve mixed-integer linear programs. *Operations Research Letters*, 42(2):186–189, 2014. Cited on page 185.

[Cbc] CBC user guide – COIN-OR. `http://www.coin-or.org/Cbc`. Cited on pages 20, 33, 62, 75, 115, and 200.

[CDFG09] William Cook, Sanjeeb Dash, Ricardo Fukasawa, and Marcos Goycoolea. Numerically safe gomory mixed-integer cuts. *INFORMS Journal on Computing*, 21(4):641–649, 2009. Cited on pages 17 and 27.

[CDM78] Harlan P. Crowder, Ron S. Dembo, and John M. Mulvey. Reporting computational experiments in mathematical programming. *Mathematical Programming*, 15:316–329, 1978. Cited on page 30.

[Chr76] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976. Cited on page 1.

[Chr05] Philipp M. Christophel. An improved heuristic for the MOPS mixed integer programming solver. Diploma thesis, Universität Paderborn, 2005. Cited on page 42.

[CKS90] William J. Cook, Ravi Kannan, and Alexander Schrijver. Chvátal closures for mixed integer programming problems. *Mathematical Programming*, 47(1–3):155–174, 1990. Cited on page 155.

[CKSW13] William Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305–344, 2013. Cited on page 27.

[CL10] Alberto Caprara and Marco Locatelli. Global optimization problems and domain reduction strategies. *Mathematical Programming*, 125:123–137, 2010. Cited on pages 159 and 167.

[CLN11] Gérard Cornuéjols, Leo Liberti, and Giacomo Nannicini. Improved strategies for branching on general disjunctions. *Mathematical Programming*, 130(2):225–247, 2011. Cited on page 156.

[CMM98] Joseph Czyzyk, Michael P. Mesnier, and Jorge J. Moré. The NEOS server. *Computational Science & Engineering, IEEE*, 5(3):68–75, 1998. `http://www.neos-server.org/neos/`. Cited on page 164.

[Coh95] Paul R. Cohen. *Empirical Methods for Artificial Intelligence.* MIT Press, 1995. Cited on page 166.

[Coo08] William J. Cook. mathematical programming computation: a new MPS journal. *Optima*, 78:1, 7, 8, 11, 2008. Cited on page 30.

[Cor10] COR@L MIP Instances. `http://coral.ie.lehigh.edu/data-sets/mixed-integer-instances/`, 2010. Cited on page 164.

[Cpp] CppAD. A Package for Differentiation of C++ Algorithms. `http://www.coin-or.org/CppAD/`. Cited on pages 101, 129, and 172.

[Dak65] Robert J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965. Cited on pages 21 and 151.

[Dan08] Emilie Danna. Performance variability in mixed integer programming. Presentation slides from MIP 2008 workshop in New York City. `http://coral.ie.lehigh.edu/~jeff/mip-2008/program.pdf`, 2008. Cited on page 173.

[DBS02] Bruce Davey, Natashia Boland, and Peter J. Stuckey. Efficient intelligent backtracking using linear programming. *INFORMS Journal of Computing*, 14(4):373–386, 2002. Cited on page 138.

[DFLL10] Claudia D'Ambrosio, Antonio Frangioni, Leo Liberti, and Andrea Lodi. Experiments with a feasibility pump approach for nonconvex MINLPs. In Paola Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 350–360. Springer Berlin Heidelberg, 2010. Cited on pages 2, 61, 68, 69, 77, 98, and 99.

[DFLL12] Claudia D'Ambrosio, Antonio Frangioni, Leo Liberti, and Andrea Lodi. A storm of feasibility pumps for nonconvex MINLP. *Mathematical Programming*, 136:375–402, 2012. Cited on pages 2, 61, 67, 68, 69, 70, 71, 72, 77, 98, 99, and 118.

[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. Cited on pages 19 and 67.

[DGM$^+$09] Bistra Dilkina, Carla P. Gomes, Yuri Malitsky, Ashish Sabharwal, and Meinolf Sellmann. Backdoors to combinatorial optimization: Feasibility and optimality. In Willem-Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 56–70. Springer Berlin Heidelberg, 2009. Cited on page 154.

[DL11] Claudia D'Ambrosio and Andrea Lodi. Mixed integer nonlinear programming tools: a practical overview. *4OR: A Quarterly Journal of Operations Research*, 9:329–349, 2011. Cited on page 20.

[DM02] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002. Cited on page 33.

[Dri66] Norman J. Driebeek. An algorithm for the solution of mixed integer programming problems. *Management Science*, 12(7):576–587, 1966. Cited on page 151.

[DRP04] Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2004. Cited on pages 23, 24, 31, 40, 79, 80, 82, 95, 96, 97, 98, 103, 118, 121, and 134.

[DSLR13] Marianna De Santis, Stefano Lucidi, and Francesco Rinaldi. A new class of functions for measuring solution integrality in the feasibility pump approach. *SIAM Journal on Optimization*, 23(3):1575–1606, 2013. Cited on pages 61, 66, and 67.

[DSLR14] Marianna De Santis, Stefano Lucidi, and Francesco Rinaldi. Feasibility pump-like heuristics for mixed integer problems. *Discrete Applied Mathematics*, 165:152–167, 2014. Cited on pages 61, 66, and 67.

[EC68] Robert E. Echols and Leon Cooper. Solution of integer linear programming problems by direct search. *Journal of the Association for Computing Machinery*, 15(1):75–84, 1968. Cited on page 23.

[EMDS11] Issmail Elhallaoui, Abdelmoutalib Metrane, Guy Desaulniers, and François Soumis. An improved primal simplex algorithm for degenerate linear programs. *INFORMS Journal on Computing*, 23(4):569–577, 2011. Cited on page 158.

[EN07] Jonathan Eckstein and Mikhail Nediak. Pivot, cut, and dive: a heuristic for 0-1 mixed integer programming. *Journal of Heuristics*, 13(5):471–503, 2007. Cited on pages 25, 39, and 66.

[FF56] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956. Cited on page 17.

[FGL05] Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005. Cited on pages 22, 61, 62, 64, 67, 70, 99, and 159.

[FIC] FICO Xpress Optimizer. `http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-Optimizer.aspx`. Cited on pages 15, 20, 33, and 62.

[FL03] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003. Cited on pages 18, 23, 79, 80, 81, 95, 98, 157, and 158.

[FL08] Matteo Fischetti and Andrea Lodi. Repairing MIP infeasibility through local branching. *Computers & Operations Research*, 35(5):1436–1445, 2008. Special Issue: Algorithms and Computational Methods in Feasibility and Infeasibility. Cited on page 82.

[FL10] Matteo Fischetti and Andrea Lodi. Heuristics in mixed integer programming. In James J. Cochran, Louis A. Cox, Pinar Keskinocak, Jeffrey P. Kharoufeh, and J. Cole Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2010. Online publication. Cited on pages 2, 23, and 79.

[FLH05] John Forrest and Robin Lougee-Heimer. Cbc user guide. *INFORMS Tutorials in Operations Research*, pages 257–277, 2005. Cited on page 62.

[FLM$^+$13] Matteo Fischetti, Andrea Lodi, Michele Monaci, Domenico Salvagnin, and Andrea Tramontani. Tree search stabilization by random sampling. Technical report, DEI, University of Padova, 2013. Cited on pages 38 and 185.

[FLRKS78] Bennett L. Fox, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and Linus E. Schrage. Branching from the largest upper bound folklore and facts. *European Journal of Operational Research*, 2(3):191–194, 1978. Cited on page 150.

[FLS10] Matteo Fischetti, Andrea Lodi, and Domenico Salvagnin. Just MIP it! In Vittorio Maniezzo, Thomas Stützle, and Stefan Voß, editors, *Matheuristics*, volume 10 of *Annals of Information Systems*, pages 39–70. Springer US, 2010. Cited on pages 80 and 125.

[FM11] Matteo Fischetti and Michele Monaci. Backdoor branching. In Oktay Günlük and Gerhard J. Woeginger, editors, *Integer Programming and Combinatoral Optimization - 15th International Conference, IPCO 2011, New York, NY, USA, June 15-17, 2011. Proceedings*, volume 6655 of *Lecture Notes in Computer Science*, pages 183–191. Springer, 2011. Cited on page 154.

[FM12] Matteo Fischetti and Michele Monaci. Branching on nonchimerical fractionalities. *Operations Research Letters*, 40(3):159–164, 2012. Cited on pages 18, 152, 163, 164, and 168.

[FM13] Matteo Fischetti and Michele Monaci. Proximity search for 0-1 mixed-integer convex programming. Technical report, DEI, University of Padova, 2013. Cited on pages 38, 64, 80, 84, 85, and 98.

[FM14] Matteo Fischetti and Michele Monaci. Exploiting erraticism in search. *Operations Research*, 62(1):114–122, 2014. Cited on page 185.

[FMS14] Matteo Fischetti, Michele Monaci, and Domenico Salvagnin. Mixed-integer linear programming heuristics for the PrePack optimization problem. Technical report, DEI, University of Padova, 2014. Cited on page 38.

[Fou03] Robert Fourer. Software survey: Linear programming. *OR/MS Today*, 30(6):34–43, 2003. Cited on page 146.

[Fou13] Robert Fourer. Software survey linear programming: twelfth in a series of lp surveys. *OR/MS Today*, 40(6):40–43, 2013. Cited on page 20.

[FS09] Matteo Fischetti and Domenico Salvagnin. Feasibility pump 2.0. *Mathematical Programming Computation*, 1:201–222, 2009. Cited on pages 22, 23, 61, 65, 70, 71, 99, 107, and 123.

[Fuk11] Komei Fukuda. lecture notes "Polyhedral Computation", 2011. `http://stat.ethz.ch/ifor/teaching/lectures/poly_comp_ss11/lecture3`. Cited on page 14.

[FW56] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956. Cited on page 66.

[Gam10] Gerald Gamrath. Generic branch-cut-and-price. Diploma thesis, Technische Universität Berlin, 2010. Cited on page 85.

[Gam13] Gerald Gamrath. Improving strong branching by propagation. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 347–354. Springer Berlin Heidelberg, 2013. Cited on pages 152 and 168.

[Gam14] Gerald Gamrath. Improving strong branching by domain propagation. *EURO Journal on Computational Optimization*, 2:99–122, 2014. Cited on pages 152 and 168.

[Gar13] Frédéric Gardi. Toward a mathematical programming solver based on local search. Habilitation thesis, Université Pierre et Marie Curie, 2013. Cited on page 26.

[GGM$^+$97] Ian P. Gent, Stuart A. Grant, Ewen MacIntyre, Patrick Prosser, Paul Shaw, Barbara M. Smith, and Toby Walsh. How not to do it. Research report series, University of Leeds, School Of Computer Studies, 1997. Cited on page 30.

[GH05] Shubhashis Ghosh and Ryan Hayward. Pivot and gomory cut: a 0-1 mixed integer programming heuristic. Preprint, Department of Computing Science: University of Alberta, 2005. Cited on page 25.

[Gho07] Shubhashis Ghosh. DINS, a MIP improvement heuristic. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization, 12th International IPCO Conference, Proceedings*, volume 4513 of *LNCS*, pages 310–323. Springer Berlin Heidelberg, 2007. Cited on pages 23, 79, 80, 84, 95, 96, 98, 118, and 121.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979. Cited on pages 2 and 122.

[GKM$^+$13] Gerald Gamrath, Thorsten Koch, Alexander Martin, Matthias Miltenberger, and Dieter Weninger. Progress in presolving for mixed integer programming. ZIB-Report 13-48, Zuse Institute Berlin, 2013. Cited on pages 44, 146, and 171.

[GL97a] Fred Glover and Manuel Laguna. General Purpose Heuristics for Integer Programming - Part I. *Journal of Heuristics 3*, 1997. Cited on page 25.

[GL97b] Fred Glover and Manuel Laguna. General Purpose Heuristics for Integer Programming - Part II. *Journal of Heuristics 3*, 1997. Cited on page 25.

[GL06] Wasu Glankwamdee and Jeff Linderoth. Lookahead branching for mixed integer programming. Technical report, Technical Report 06T-004, Lehigh University, 2006. Cited on page 153.

[GL10] Gerald Gamrath and Marco E. Lübbecke. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In Paola Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 239–252. Springer Berlin Heidelberg, 2010. Cited on page 85.

[Glo] GloMIQO 2.0. `http://helios.princeton.edu/GloMIQO/`. Cited on pages 129, 135, and 170.

[Glp] the GNU linear programming kit. `http://www.gnu.org/software/glpk`. Cited on pages 20 and 62.

[GLS81] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. Cited on page 14.

[GLW00] Fred Glover, Arne Løkketangen, and David L. Woodruff. Scatter search to generate diverse MIP solutions. In M. Laguna and J.L. González-Velarde, editors, *OR Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 299–317. Kluwer Academic Publishers, 2000. Cited on page 23.

[GM91] Harvey J. Greenberg and Frederic H. Murphy. Approaches to diagnosing infeasible linear programs. *ORSA Journal on Computing*, 3(3):253–261, 1991. Cited on page 140.

[GNS13] Menal Guzelsoy, George Nemhauser, and Martin Savelsbergh. Restrict-and-relax search for 0-1 mixed-integer programs. *EURO Journal on Computational Optimization*, pages 1–18, 2013. online first publication. Cited on page 26.

[Gol91] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. Cited on page 27.

[Gom58] Ralph Edward Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958. Cited on pages 16 and 25.

[Gom60] Ralph E. Gomory. An algorithm for the mixed integer problem. Technical report, RAND Corporation, 1960. Cited on pages 16 and 156.

[GR77] Jean-Michel Gauthier and Gerard Ribière. Experiments in mixed-integer linear programming using pseudo-costs. *Mathematical Programming*, 12(1):26–47, 1977. Cited on pages 149, 150, 151, and 152.

[GR85] Omprakash K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management science*, 31(12):1533–1546, 1985. Cited on page 19.

[Gra75] Jack E. Graver. On the foundations of linear and integer linear programming I. *Mathematical Programming*, 9(1):207–226, 1975. Cited on page 18.

[GS02] Ignacio E. Grossman and Nikolaos V. Sahinidis. Special issue on mixed integer programming and its application to engineering, part I. *Optimization and Engineering*, 3(4), 2002. Cited on page 2.

[GS11] Andrew Gilpin and Tuomas Sandholm. Information-theoretic approaches to branching in search. *Discrete Optimization*, 8(2):147–159, 2011. Cited on page 153.

[GSS13] Thiago M. Gomes, Haroldo G. Santos, and Marcone J. F. Souza. A pre-processing aware RINS based MIP heuristic. In *Hybrid Metaheuristics*, pages 1–11. Springer, 2013. Cited on page 82.

[Gur] Gurobi optimization. `http://www.gurobi.com/`. Cited on pages 15, 20, 33, and 62.

[Gur14] Gurobi. Experimental no relaxation heuristic, 2014. `http://www.gurobi.com/resources/norel-heuristic`. Accessed 15. February 2014. Cited on page 60.

[GW08] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Society for Industrial and Applied Mathematics (SIAM), 2008. Cited on page 125.

[GW13] Ambros M. Gleixner and Stefan Weltge. Implementation of an OBBT propagator in SCIP. Presentation slides from SCIP workshop 2012 in Darmstadt. `http://www.math.uni-magdeburg.de/~weltge/talks/darmstadt-2013-obbt.pdf`, 2013. Cited on page 22.

[HE80] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980. Cited on page 41.

[Hei] Stefan Heinz. *Global structures and dual reductions for cumulative scheduling*. PhD thesis, Technische Universität Berlin. To appear. Cited on pages 142 and 143.

[Hel76] Friedrich Robert Helmert. Über die Wahrscheinlichkeit der Potenzsummen der Beobachtungsfehler und über einige damit in Zusammenhange stehende Fragen. *Zeitschrift für Mathematik und Physik*, 21:102–219, 1876. Cited on page 175.

[Hen11] Gregor Hendel. New rounding and propagation heuristics for mixed integer programming. Bachelor's thesis, TU Berlin, 2011. Cited on pages 39, 43, 49, and 103.

[HG95] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 607–615, 1995. Cited on page 43.

[Hil00] David Hilbert. Mathematische Probleme. *Nachrichten der Königlichen Gesellschaft der Wissenschaften zu Göttingen, mathematisch-physikalische Klasse*, 3:253–297, 1900. Cited on page 14.

[HJ92] Pierre Hansen and Brigitte Jaumard. Reduction of indefinite quadratic programs to bilinear programs. *Journal of Global Optimization*, 2(1):41–60, 1992. Cited on page 121.

[HJP75] Peter L. Hammer, Ellis L. Johnson, and Uri N. Peled. Facets of regular 0-1 polytopes. *Mathematical Programming*, 8:179–206, 1975. Cited on pages 10 and 97.

[HKLW10] Raymond Hemmecke, Matthias Köppe, Jon Lee, and Robert Weismantel. Nonlinear integer programming. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958–2008*, pages 561–618. Springer Berlin Heidelberg, 2010. Cited on pages 13 and 18.

[HKW01] Utz-Uwe Haus, Matthias Köppe, and Robert Weismantel. The integral basis method for integer programming. *Mathematical Methods of Operations Research*, 53(3):353–361, 2001. Cited on pages 18 and 19.

[HLM10] Saïd Hanafi, Jasmina Lazić, and Nenad Mladenović. Variable neighbourhood pump heuristic for 0-1 mixed integer programming feasibility. *Electronic Notes in Discrete Mathematics*, 36(0):759–766, 2010. Cited on pages 61 and 64.

[HM01] Pierre Hansen and Nenad Mladenović. Variable neighbourhood search: principles and applications. *European Journal of Operations Research*, 130:449–467, 2001. Cited on page 70.

[HMSW53] Alan J. Hoffman, Murray Mannos, Daniel Sokolowsky, and Norman A. Wiegmann. Computational experience in solving linear programs. *Journal of the Society for Industrial and Applied Mathematics*, 1(1):17–33, 1953. Cited on page 29.

[HMU06] Pierre Hansen, Nenad Mladenović, and Dragan Urošević. Variable neighborhood search and local branching. *Computers & Operations Research*, 33(10):3034–3045, 2006. Cited on pages 23 and 70.

[Hoo94] John N. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42:201–212, 1994. Cited on page 30.

[Hoo07] John N. Hooker. *Integrated Methods for Optimization.* International Series in Operations Research & Management Science. Springer, New York, 2007. Cited on page 7.

[HP93] Karla L. Hoffman and Manfred Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6), 1993. Cited on page 30.

[HT96] Reiner Horst and Hoang Tuy. *Global optimization: Deterministic approaches*. Springer, 1996. Cited on page 16.

[IBM] IBM ILOG CPLEX Optimizer. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`. Cited on pages 15, 20, 33, 62, and 129.

[Ipo] Ipopt (Interior Point OPTimizer). `http://www.coin-or.org/Ipopt/`. Cited on pages 75, 101, and 172.

[JB00] Narendra Jussien and Vincent Barichard. The PaLM system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pages 118–133, 2000. Cited on page 138.

[JBNP91] Richard Henry Frymuth Jackson, Paul T. Boggs, Stephen G. Nash, and Susan Powell. Guidelines for reporting results of computational experiments. Report of the ad hoc committee. *Mathematical Programming*, 49:413–425, 1991. Cited on pages 30 and 36.

[Jen13] Bo Jensen. Exploiting degeneracy in Sulum MIP Optimizer. Presentation slides from INFORMS Annual Meeting 2013 in Minneapolis. `http://www.sulumoptimization.com/cmsms/sulumdocs/Minneapolis/TechnicalTalk/slides/minneapolis.html`, 2013. Cited on pages 167 and 200.

[JKS85] Ellis L. Johnson, Michael M. Kostreva, and Uwe H. Suhl. Solving 0-1 integer programming problems arising from large scale planning models. *Operations Research*, 33:803–819, 1985. Cited on page 37.

[JOP$^+$] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. Cited on page 113.

[KAA$^+$11] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelmann, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011. Cited on pages 20, 30, 33, 35, 36, 40, 43, 55, 96, 101, 144, 158, 164, 165, 166, 169, 173, and 174.

[Kar72] Richard Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. Cited on pages 2 and 14.

[KB05] George Katsirelos and Fahiem Bacchus. Generalised nogoods in CSPs. In *Proceedings of AAAI-2005*, pages 390–396, 2005. Cited on page 138.

[KBE$^+$14] Thorsten Koch, Dagmar Bargmann, Mirko Ebbers, Armin Fügenschuh, Björn Geißler, Nina Geißler, Ralf Gollmer, Uwe Gotzes, Christine Hayn, Holger Heitsch, René Henrion, Benjamin Hiller, Jesco Humpola, Imke Joormann, Veronika Kühl, Thomas Lehmann, Hernan Leövey, Alexander Martin, Radoslava Mirkov, Andris Möller, Antonio Morsi, Djamal Oucherif, Antje Pelzer, Marc E. Pfetsch, Lars Schewe, Werner Römisch, Jessica Rövekamp, Martin Schmidt, Rüdiger Schultz, Robert Schwarz, Jonas Schweiger, Klaus Spreckelsen, Claudia Stangl, Marc C. Steinbach, Ansgar Steinkamp, Isabel Wegner-Specht, Bernhard M. Willert, and Stefan Vigerske. *Evaluating Gas Network Capacities.* MOS-SIAM Series on Optimization, 2014. In preparation. Cited on pages 171 and 194.

[KC11] Miroslav Karamanov and Gérard Cornuéjols. Branching on general disjunctions. *Mathematical Programming*, 128(1-2):403–436, 2011. Cited on page 156.

[Kha79] Leonid G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244(5):1093–1096, 1979. english translation in Soviet Math. Dokl. 20(1):191–194, 1979. Cited on pages 2 and 14.

[KKNS09] Fatma Kılınç Karzan, George L. Nemhauser, and Martin W. P. Savelsbergh. Information-based branching schemes for binary linear mixed-integer programs. *Mathematical Programming Computation*, 1(4):249–293, 2009. Cited on pages 139, 152, 153, and 154.

[KLMP07] Zeynep Kiziltan, Andrea Lodi, Michela Milano, and Fabio Parisini. CP-based local branching. In Christian Bessière, editor, *Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *LNCS*, pages 847–855. Springer, 2007. Cited on pages 80 and 85.

[KLMP12] Zeynep Kiziltan, Andrea Lodi, Michela Milano, and Fabio Parisini. Bounding, filtering and diversification in CP-based local branching. *Journal of Heuristics*, 18(3):353–374, 2012. Cited on pages 80 and 85.

[KM72] Victor Klee and George J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities*, volume III, pages 159–175. Academic Press, New York, 1972. Cited on page 22.

[KMP13] Thorsten Koch, Alexander Martin, and Marc E. Pfetsch. Progress in academic computational integer programming. In Michael Jünger and Gerhard Reinelt, editors, *Facets of Combinatorial Optimization*, pages 483–506. Springer Berlin Heidelberg, 2013. Cited on page 30.

[Köp12] Matthias Köppe. On the complexity of nonlinear mixed-integer optimization. In Jon Lee and Sven Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 533–557. Springer New York, 2012. Cited on page 13.

[KRS12] Thorsten Koch, Ted K. Ralphs, and Yuji Shinano. Could we use a million cores to solve an integer program? *Mathematical Methods of Operations Research*, 76(1):67–93, 2012. Cited on page 182.

[LA97] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proc. of CP*, pages 342–356, Autriche, 1997. Springer. Cited on page 152.

[Lam] LaMaTTO++. `http://www.mso.math.fau.de/edom/projects/lamatto.html`. Cited on page 194.

[LBN14] Pierre Le Bodic and George Nemhauser. How important are branching decisions: fooling MIP solvers. Technical Report 4324, Optimization Online, 2014. to appear in Operations Research Letters. Cited on page 157.

[LD60] Ailsa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960. Cited on pages 15, 21, 22, and 120.

[Ley01] Sven Leyffer. Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization and Applications*, 18(3):295–309, 2001. Cited on page 19.

[Lib00] Paolo Liberatore. On the complexity of choosing the branching literal in DPLL. *Artificial Intelligence*, 116(1–2):315–326, 2000. Cited on pages 21, 150, and 157.

[Lin] Lindo Systems, Inc. `http://www.lindo.com`. Cited on page 20.

[LJ83] Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of operations research*, pages 538–548, 1983. Cited on page 155.

[LJS94] Arne Løkketangen, Kurt Jörnsten, and Sverre Storøy. Tabu search within a pivot and complement framework. *International Transactions in Operational Research*, 1(3):305–316, 1994. Cited on page 25.

[LK73] Shen Lin and Brian W. Kernighan. An effective heuristic algorithm for the travelling-salesman problem. *Operations Research*, 21:498–516, 1973. Cited on page 43.

[LL02] Adam N. Letchford and Andrea Lodi. Primal cutting plane algorithms revisited. *Mathematical Methods of Operations Research*, 56(1):67–81, 2002. Cited on page 18.

[LL10] Jeffrey T. Linderoth and Andrea Lodi. Milp software. In James J. Cochran, Louis A. Cox, Pinar Keskinocak, Jeffrey P. Kharoufeh, and J. Cole Smith, editors, *Wiley encyclopedia of operations research and management science*. John Wiley & Sons, Inc., 2010. online publication. Cited on page 20.

[LMN11] Leo Liberti, Nenad Mladenović, and Giacomo Nannicini. A recipe for finding good solutions to MINLPs. *Mathematical Programming Computation*, 3:349–390, 2011. Cited on pages 2, 26, 98, and 118.

[LNM10] Leo Liberti, Giacomo Nannicini, and Nenad Mladenović. A good recipe for solving MINLPs. In Vittorio Maniezzo, Thomas Stützle, and Stefan Voß, editors, *Matheuristics*, volume 10 of *Annals of Information Systems*, pages 231–244. Springer, 2010. Cited on pages 26, 98, and 118.

[Loc] Localsolver: News. `http://www.localsolver.com/news.html?id=32`. Accessed 30. July 2014. Cited on page 92.

[Lod10] Andrea Lodi. Mixed integer programming computation. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958–2008*, pages 619–645. Springer Berlin Heidelberg, 2010. Cited on pages 7 and 40.

[Lod13] Andrea Lodi. The heuristic (dark) side of MIP solvers. In El-Ghazali Talbi, editor, *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 273–284. Springer Berlin Heidelberg, 2013. Cited on pages 23 and 79.

[Løk02] Arne Løkketangen. Heuristics for 0-1 mixed integer programming. *Handbook of Applied Optimization*, pages 474–477, 2002. Cited on page 23.

[LP12] Marco Lübbecke and Christian Puchert. Primal heuristics for branch-and-price algorithms. In *Operations Research Proceedings 2011*, pages 65–70. Springer, 2012. Cited on page 85.

[Lps] lp_solve 5.5.2. `http://lpsolve.sourceforge.net`. Cited on page 20.

[LPT$^+$09] Richard Laundy, Michael Perregaard, Gabriel Tavares, Horia Tipi, and Alkis Vazacopoulos. Solving hard mixed-integer programming problems with Xpress-MP: a MIPLIB 2003 case study. *INFORMS Journal on Computing*, 21(2):304–313, 2009. Cited on pages 2, 7, and 22.

[LRRS11] Andrea Lodi, Ted K. Ralphs, Fabrizio Rossi, and Stefano Smriglio. Interdiction branching. Technical report, Technical Report OR/09/10, DEIS-Università di Bologna, 2011. Cited on page 157.

[LS99] Jeff T. Linderoth and Martin W. P. Savelsbergh. A computational study of strategies for mixed integer programming. *INFORMS Journal on Computing*, 11:173–187, 1999. Cited on pages 150 and 151.

[LS09] Youdong Lin and Linus Schrage. The global solver in the LINDO API. *Optimization Methods and Software*, 24(4–5):657–668, 2009. Cited on page 20.

[LTL$^+$09] R. Luce, J. Duintjer Tebbens, J. Liesen, R. Nabben, M. Grötschel, T. Koch, and O. Schenk. On the factorization of simplex basis matrices. ZIB-Report 09-24, TU Berlin, Zuse Institute Berlin, University of Basel, 2009. Cited on page 22.

[Mah09] Ashutosh Mahajan. *On selecting disjunctions for solving mixed integer programming problems*. PhD thesis, Lehigh University, 2009. Cited on page 155.

[Mah10] Ashutosh Mahajan. Presolving mixed–integer linear programs. In James J. Cochran, Louis A. Cox, Pinar Keskinocak, Jeffrey P. Kharoufeh, and J. Cole Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2010. online publication. Cited on page 44.

[Mar99] Alexander Martin. Integer programs with block structure. Preprint SC 99-03, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1999. Habilitation Thesis. Cited on page 153.

[Mar11] Jakub Mareček. *Exploiting Structure in Integer Programs*. PhD thesis, University of Nottingham, 2011. Cited on page 23.

[Mat70] Juri V. Matijasevič. Enumerable sets are Diophantine. *Soviet Mathematics. Doklady*, 11(2):354–358, 1970. Cited on pages 2 and 14.

[MC13] Hanan Mahmoud and John W. Chinneck. Achieving MILP feasibility quickly using general disjunctions. *Computers & Operations Research*, 40(8):2094–2102, 2013. Cited on pages 24, 156, and 158.

[McC76] Garth P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. *Mathematical Programming*, 10:146–175, 1976. Cited on pages 16, 69, and 75.

[McG96] Catherine C. McGeoch. Toward an experimental method for algorithm simulation. *INFORMS Journal on Computing*, 8(1):1–15, 1996. Cited on page 30.

[McN47] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. Cited on pages 110 and 175.

[MF13] Ruth Misener and Christodoulos A. Floudas. GloMIQO: Global mixed-integer quadratic optimizer. *Journal of Global Optimization*, 57:3–50, 2013. Cited on pages 20, 129, and 170.

[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. Cited on pages 20 and 118.

[Min] MINLP Library. `http://www.gamsworld.org/minlp/minlplib.htm`. Cited on page 187.

[Mio] MINOTAUR: a toolkit for MINLP. `http://wiki.mcs.anl.gov/minotaur`. Cited on pages 20 and 118.

[Mit] Hans Mittelmann. Benchmarks for optimization software: Miplib2010. `http://plato.asu.edu/bench.html`. Cited on pages 20, 33, 34, 36, and 172.

[Mit13] Hans Mittelmann. Benchmarks for optimization software: Feasibility benchmark, 2013. `http://plato.asu.edu/ftp/feas_bench.html`. Accessed 1. April 2013. Cited on page 23.

[MJPL90] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the eighth National conference on Artificial intelligence*, pages 17–24. AAAI Press, 1990. Cited on page 50.

[MLK12] Ashutosh Mahajan, Sven Leyffer, and Christian Kirches. Solving mixed-integer nonlinear programs by QP-diving. Preprint ANL/MCS-2071-0312, Argonne National Laboratory, Mathematics and Computer Science Division, 2012. Cited on pages 19, 24, and 168.

[MMZ$^+$01] Mattew H. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC'01*, pages 530–535, 2001. Cited on pages 46, 123, 124, 138, and 152.

[Mos] MOSEK ApS. `http://www.mosek.com/`. Cited on page 20.

[MR09] Ashutosh Mahajan and Ted K. Ralphs. Experiments with branching using general disjunctions. In John W. Chinneck, Bjarni Kristjansson, and Matthew J. Saltzman, editors, *Operations Research and Cyber-Infrastructure*, volume 47 of *Operations Research/Computer Science Interfaces*, pages 101–118. Springer US, 2009. Cited on page 156.

[MR10] Ashutosh Mahajan and Ted K. Ralphs. On the complexity of selecting disjunctions in integer programming. *SIAM Journal on Optimization*, 20(5):2181–2198, 2010. Cited on page 156.

[MSS99] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions of Computers*, 48:506–521, 1999. Cited on pages 138 and 152.

[MW01] Hugues Marchand and Laurence A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49(3):363–371, 2001. Cited on page 21.

[NB10] Giacomo Nannicini and Pietro Belotti. Rounding-based heuristics for nonconvex MINLPs. In Pierre Bonami, Leo Liberti, Andrew J. Miller, and Annick Sartenaer, editors, *Proceedings of the EWMINLP*, pages 159–167, April 2010. Cited on page 33.

[NB12] Giacomo Nannicini and Pietro Belotti. Rounding-based heuristics for nonconvex MINLPs. *Mathematical Programming Computation*, 4(1):1–31, 2012. Cited on pages 2, 26, 98, 111, and 118.

[NBL08] Giacomo Nannicini, Pietro Belotti, and Leo Liberti. A local branching heuristic for MINLPs. *ArXiv e-prints*, 2008. Cited on pages 2, 80, 85, 86, 98, and 118.

[NS04] Arnold Neumaier and Oleg Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming*, 99(2):283–296, 2004. Cited on page 27.

[NS13] Joe Naoum-Sawaya. Recursive central rounding for mixed integer programs. *Computers & Operations Research*, 2013. Cited on pages 42 and 65.

[NSHV05] Arnold Neumaier, Oleg Shcherbina, Waltraud Huyer, and Tamás Vinkó. A comparison of complete global optimization solvers. *Mathematical Programming*, 103(2):335–356, 2005. Cited on page 20.

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley, 1988. Cited on page 22.

[OLRS08] James Ostrowski, Jeff Linderoth, Fabrizio Rossi, and Stefano Smriglio. Constraint orbital branching. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Integer Programming and Combinatorial Optimization*, volume 5035 of *Lecture Notes in Computer Science*, pages 225–239. Springer Berlin Heidelberg, 2008. Cited on page 157.

[OLRS11] James Ostrowski, Jeff Linderoth, Fabrizio Rossi, and Stefano Smriglio. Orbital branching. *Mathematical Programming*, 126(1):147–178, 2011. Cited on page 157.

[OM01] Jonathan H. Owen and Sanjay Mehrotra. Experimental results on using general disjunctions in branch-and-bound for general-integer linear programs. *Computational Optimization and Applications*, 20(2):159–170, 2001. Cited on page 156.

[OSC09] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009. Cited on page 138.

[PC07] Jagat Patel and John W. Chinneck. Active-constraint variable ordering for faster feasibility of mixed integer linear programs. *Mathematical Programming*, 110:445–474, 2007. Cited on pages 24, 153, 156, and 158.

[PC11] Jennifer Pryor and John W. Chinneck. Faster integer-feasibility in mixed-integer linear programs by branching to force change. *Computers & Operations Research*, 38(8):1143–1152, 2011. Cited on pages 24, 153, and 158.

[Per] Michael Perregaard. personal communication. Cited on pages 167 and 200.

[Per68] Richard K. Perrin. *Use of the Linear Programming Facility of the IBM Mathematical Programming System (MPS/360)*. North Carolina State University, 1968. Cited on page 20.

[Per11] Michael Perregaard. Split Disjunctions in Xpress: Combining Lift-and-Project with Reduce-and-Split for cutting and branching in Xpress. Presentation slides from MIP 2011 workshop in Waterloo. `http://www.math.uwaterloo.ca/~mip2011/abstracts.pdf`, 2011. Cited on page 157.

[Pfe08] Marc E. Pfetsch. Branch-and-cut for the maximum feasible subsystem problem. *SIAM Journal on Optimization*, 19(1):21–38, 2008. Cited on page 171.

[PFl$^+$12] Marc E. Pfetsch, Armin Fügenschuh, Björn Geißler, Nina Geißler, Ralf Gollmer, Benjamin Hiller, Jesco Humpola, Thorsten Koch, Thomas Lehmann, Alexander Martin, Antonio Morsi, Jessica Rövekamp, Lars Schewe, Martin Schmidt, Rüdiger Schultz, Robert Schwarz, Jonas Schweiger, Claudia Stangl, Marc Steinbach, Stefan Vigerske, and Bernhard Willert. Validation of nominations in gas network optimization:Models, methods, and solutions. *Optimization Methods and Software*, 2012. Cited on page 171.

[PG99] Gilles Pesant and Michel Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–279, 1999. Cited on page 80.

[PM12] Fabio Parisini and Michela Milano. Sliced neighborhood search. *Expert Systems with Applications*, 39(5):5739–5747, 2012. Cited on page 85.

[PQ08] Gilles Pesant and Claude-Guy Quimper. Counting solutions of knapsack constraints. In Laurent Perron and Michael A. Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5015 of *Lecture Notes in Computer Science*, pages 203–217. Springer, May 2008. Cited on page 153.

[PQZ12] Gilles Pesant, Claude-Guy Quimper, and Alessandro Zanarini. Counting-based search: Branching heuristics for constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 43(1):173–210, 2012. Cited on pages 50 and 153.

[PR10] David Pisinger and Stefan Røpke. Large neighborhood search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, chapter 13, pages 399–420. Springer, 2nd edition, 2010. Cited on page 80.

[PSF04] Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In Mark Wallace, editor, *Proc. of CP 2004*, volume 3258 of *LNCS*, pages 468–481. Springer Berlin Heidelberg, 2004. Cited on page 80.

[Puc11] Christian Puchert. Primal heuristics for branch-and-price algorithms. Master-thesis, Technische Universität Darmstadt, 2011. Cited on page 85.

[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. Cited on pages 19 and 85.

[QG93] Ignacio Quesada and Ignacio E. Grossmann. Global optimization algorithm for heat exchanger networks. *Industrial & Engineering Chemistry Research*, 32(3):487–499, 1993. Cited on page 22.

[QvGH+99] Arie J. Quist, René van Geemert, Jan E. Hoogenboom, Tibor Illés, Etienne de Klerk, Cornelis Roos, and Tamaás Terlaky. Finding optimal nuclear reactor core reload patterns using nonlinear optimization and search heuristics. *Engineering Optimization*, 32(2):143–176, 1999. Cited on page 187.

[RF81] David M. Ryan and Brian A. Foster. An integer programming approach to scheduling. *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pages 269–280, 1981. Cited on page 154.

[Rot07] Edward Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007. Cited on pages 23, 79, 80, 83, 96, and 98.

[RWH99] Robert Rodosek, Mark G. Wallace, and Mozafar T. Hajian. A new approach to integrating mixed integer programming and constraint logic programming. *Annals of Operations Research*, 86(1):63–87, 1999. Cited on page 138.

[SAB+12] Yuji Shinano, Tobias Achterberg, Timo Berthold, Stefan Heinz, and Thorsten Koch. ParaSCIP – a parallel extension of SCIP. In Christian Bischof, Heinz-Gerd Hegering, Wolfgang E. Nagel, and Gabriel Wittum, editors, *Competence in High Performance Computing 2010*, pages 135–148. Springer, 2012. Cited on pages 182 and 185.

[SAB+13] Yuji Shinano, Tobias Achterberg, Timo Berthold, Stefan Heinz, Thorsten Koch, and Michael Winkler. Solving hard MIPLIB 2003 problems with ParaSCIP on supercomputers: An update.

ZIB-Report 13-66, Zuse Institute Berlin, 2013. Cited on pages 182 and 185.

[Sah74] Sartaj Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4):262–279, 1974. Cited on page 2.

[Sah96] Nikolaos V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2):201–205, 1996. Cited on pages 20, 118, and 129.

[Sal14] Domenico Salvagnin. Detecting and exploiting permutation structures in MIPs. In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, volume 8451 of *Lecture Notes in Computer Science*, pages 29–44. Springer Berlin Heidelberg, 2014. Cited on page 38.

[Sav94] Martin W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994. Cited on pages 10, 22, 44, and 97.

[Sbb] SBB. ARKI Consulting & Development A/S and GAMS Inc. `http://www.gams.com/solvers/solvers.htm#SBB`. Cited on page 20.

[Sci] SCIP. Solving Constraint Integer Programs. `http://scip.zib.de/`. Cited on pages 5, 20, 115, and 128.

[SH92] Robert M. Saltzman and Frederick S. Hillier. A heuristic ceiling point algorithm for general integer linear programming. *Management Science*, 38(2):263–283, 1992. Cited on page 25.

[Sha98] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming – CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin Heidelberg, 1998. Cited on page 80.

[Sha13] Shaurya Sharma. Mixed-integer nonlinear programming heuristics applied to a shale gas production optimization problem. Master's thesis, Norwegian University of Science and Technology, 2013. Cited on page 67.

[Son86] Gyögy Sonnevend. An "analytical centre" for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In A. Prékopa, J. Szelezsáan, and B. Strazicky, editors, *System Modelling and Optimization*, volume 84 of *Lecture Notes in Control and Information Sciences*, pages 866–875. Springer Berlin Heidelberg, 1986. Cited on pages 42 and 65.

[Sop] SoPlex. An open source LP solver implementing the revised simplex algorithm. `http://soplex.zib.de/`. Cited on pages 33, 55, 101, and 172.

[SP97] Edward M. B. Smith and Costas C. Pantelides. Global optimisation of nonconvex MINLPs. *Computers & Chem. Engineering*, 21:791–796, 1997. Cited on page 69.

[Spi04] Kurt Spielberg. The optimization systems MPSX and OSL. In Josef Kallrath, editor, *Modeling Languages in Mathematical Optimization*, volume 88 of *Applied Optimization*, pages 267–278. Springer US, 2004. Cited on page 20.

[SS77] Richard M. Stallman and Gerald J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196, 1977. Cited on page 138.

[SS94] Uwe H. Suhl and Ralf Szymanski. Supernode processing of mixed-integer models. *Computational Optimization and Applications*, 3(4):317–331, 1994. Cited on page 22.

[SS06] Tuomas Sandholm and Robert Shields. Nogood learning for mixed integer programming. In *Workshop on Hybrid Methods and Branching Rules in Combinatorial Optimization, Montréal*, 2006. Cited on page 138.

[Sym] SYMPHONY. development home page. `https://projects.coin-or.org/SYMPHONY`. Cited on pages 20 and 62.

[Tac09] Guido Tack. *Constraint Propagation – Models, Techniques, Implementation*. Doctoral dissertation, Saarland University, 2009. Cited on page 44.

[Tom71] John A. Tomlin. An improved branch-and-bound method for integer programming. *Operations Research*, 19(4):1070–1075, 1971. Cited on page 151.

[TS02] Mohit Tawarmalani and Nikolaos V. Sahinidis. *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications*, volume 65 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, 2002. Cited on pages 16 and 74.

[TS04] Mohit Tawarmalani and Nikolaos V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004. Cited on pages 20, 69, 118, and 129.

[ULP$^+$07] Zsolt Ugray, Leon Lasdon, John Plummer, Fred Glover, James Kelly, and Rafael Martí. Scatter search and local NLP solvers: A multistart framework for global optimization. *INFORMS Journal on Computing*, 19(3):328–340, 2007. Cited on page 26.

[Vav90] Stephen A. Vavasis. Quadratic programming is in NP. *Information Processing Letters*, 36(2):73–77, 1990. Cited on page 14.

[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. submitted. Cited on pages 16, 20, 21, 74, 87, 123, and 172.

[Wal10] Chris Wallace. ZI round, a MIP rounding heuristic. *Journal of Heuristics*, 16(5):715–722, 2010. Cited on pages 23, 40, 41, and 103.

[WB06] Andreas Wächter and Lorenz T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. Cited on pages 75, 101, 129, and 172.

[Wed95] Dag Wedelin. An algorithm for large scale 0–1 integer programming with application to airline crew scheduling. *Annals of Operations Research*, 57(1):283–301, 1995. Cited on page 25.

[Wei98] Robert Weismantel. Test sets of integer programs. *Mathematical Methods of Operations Research*, 47(1):1–37, 1998. Cited on page 18.

[WGS03] Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *IJCAI*, volume 3, pages 1173–1178, 2003. Cited on page 154.

[Wil45] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, pages 80–83, 1945. Cited on pages 113 and 175.

[Win14] Michael Winkler. Presolving for pseudo-Boolean optimization problems. Diploma thesis, Technische Universität Berlin, 2014. Cited on page 46.

[WL01] Tapio Westerlund and Kurt Lundqvist. *Alpha-ECP, Version 5.01: An interactive MINLP-solver based on the Extended Cutting Plane Method.* Åbo Akademi, 2001. Cited on page 20.

[Wol75] Laurence A. Wolsey. Faces for a linear inequality in 0-1 variables. *Mathematical Programming*, 8:165–178, 1975. Cited on pages 10 and 97.

[Wol06] Kati Wolter. Implementation of Cutting Plane Separators for Mixed Integer Programs. Master's thesis, Technische Universität Berlin, 2006. Cited on page 17.

[WP95] Tapio Westerlund and Frank Pettersson. An extended cutting plane method for solving convex minlp problems. *Computers & Chemical Engineering*, 19:131–136, 1995. Cited on page 19.

[Wun96] Roland Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996. Cited on pages 75, 101, 164, and 172.

[YAH10] Tallys H. Yunes, Ionut D. Aron, and John N. Hooker. An integrated solver for optimization problems. *Operations Research*, 58(2):342–356, 2010. Cited on pages 7 and 138.

[You68] Richard D. Young. A simplified primal (all-integer) integer programming algorithm. *Operations Research*, 16(4):750–782, 1968. Cited on page 18.

[ZG99] Juan M. Zamora and Ignacio E. Grossmann. A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. *Journal of Global Optimization*, 14:217–249, 1999. Cited on pages 159 and 167.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 279–285. IEEE Press, 2001. Cited on pages 138, 139, and 141.

List of Figures

List of Tables

A. Notation

	number sets
$\mathbb{Z}$	set of integer numbers
$\mathbb{Z}_{\geqslant 0}$	set of nonnegative integer numbers
$\mathbb{Q}$	set of rational numbers
$\mathbb{R}$	set of real numbers
$\mathbb{R}_{\geqslant 0}$	set of nonnegative real numbers
e_j	unit vector, j-th component is 1, all others 0
E_n	identity matrix, 1's on the diagonal, all others 0
	runtime complexity
$\mathcal{P}$	class of problems that are solvable in polynomial time
$\mathcal{NP}$	class of problems that are verifiable in polynomial time
	problem definition: index sets
$\mathcal{M}$	set of constraint indices: $\mathcal{M} = \{1, \dots, m\}$
$\mathcal{N}$	set of variable indices: $\mathcal{N} = \{1, \dots, n\}$
$\mathcal{I}$	(index) set of integer variables: $\mathcal{I} \subseteq \mathcal{N}$
$\mathcal{B}$	(index) set of binary variables: $\mathcal{B} \subseteq \mathcal{I}$
$\mathcal{R}$	(index) set of continuous variables: $\mathcal{R} = \mathcal{N} \setminus \mathcal{I}$
	problem definition: vectors, matrices, functions
P	MINLP problem
$\bar{P}$	relaxation of P
$\tilde{P}$	a subproblem of P
x	n-dimensional vector of variables
c	objective function vector, $c \in \mathbb{R}^n$
l	lower bound vector, $l \in (\mathbb{R} \cup \{-\infty\})^n$, $l_j \leqslant x_j$ for all $j \in \mathcal{N}$
u	upper bound vector, $u \in (\mathbb{R} \cup \{+\infty\})^n$, $x_j \leqslant u_j$ for all $j \in \mathcal{N}$
$l(\tilde{P})$	lower bound vector of a subproblem
$u(\tilde{P})$	upper bound vector of a subproblem
D_j	domain interval of the j-th variable
A	coefficient matrix of an MIP or LP
b	right hand side vector of an inequality or equation system, $b \in \mathbb{R}^m$
g	constraint function system of a MINLP, $g_i : \mathbb{R}^n \to \mathbb{R}$ for all $i \in \mathcal{M}$
A	constraint matrix of a MIP, $A \in \mathbb{R}^{m \times n}$
$A_{i\cdot}$	i-th row of the constraint matrix

Q	coefficient matrix of a quadratic constraint
d	constant offset of a constraint
f	nonlinear objective function
	solution values and vectors
$\bar{x}$	solution to a relaxation
$\tilde{x}$	vector that fulfills the integrality requirements
$x^\star$	optimal solution
$c^\star$	optimal objective value
$\underline{c}$	dual bound on the optimal objective value
$\overline{c}$	primal bound on the optimal objective value
$\mathcal{R}(\bar{x})$	set of roundings of $\bar{x}$: $\{x \in \mathbb{R}^n \mid x_j \in \{\lfloor \bar{x}_j \rfloor, \lceil \bar{x}_j \rceil\}$ for all $j \in \mathcal{I}\}$
	branching and global information
$\overline{\kappa}_j$	number of constraints that can be violated by shifting variable j up
$\underline{\kappa}_j$	number of constraints that can be violated by shifting var. j down
s	branching score
Ψ	pseudocosts
ς	predicted objective gain
Δ	actual objective gain
	sets
$\mathcal{F}$	set of fractional variables
$\mathcal{K}$	set of unfixed variables
$\mathcal{L}$	set of lower bounds in a conflict constraint
$\mathcal{U}$	set of upper bounds in a conflict constraint
$\mathcal{C}$	cover of an MINLP
$\mathcal{S}$	cloud: set of alternative relaxation solutions
$\mathcal{T}$	a testset for a MIP
	global optimization
$\mathcal{C}^2$	set of twice continuously differentiable functions
J	Jacobian of a function; vector of its first order partial derivatives
H	Hessian of a function; matrix of its second order partial derivatives
L	Lagrangian (function) of an optimization problem
	means and gaps
ϕ	geometric mean
ψ	arithmetic mean
γ^p	primal gap
γ^d	dual gap
γ^{pd}	primal-dual gap

B. Tables

Table B.1.: Shift-and-Propagate: results of different variable sortings w.r.t. to the primal bound and the heuristic time.

	$\lvert\{b<0\}\rvert\downarrow$		$\lvert\cdot\rvert\downarrow$		random		$\lvert\{b<0\}\rvert\uparrow$		$\lvert\cdot\rvert\uparrow$	
Problem Name	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)
10teams	–	0.02	–	0.02	–	0.01	–	0.01	–	0.02
30n20b8	–	0.04	–	0.19	–	0.09	–	0.11	–	0.03
a1c1s1	–	0.03	–	0.05	–	0.04	–	0.03	–	0.02
acc-tight5	–	0.00	–	0.01	–	0.01	–	0.01	–	0.01
aflow30a	4606.0	0.01	4280.0	0.01	–	0.01	4606.0	0.01	4606.0	0.01
aflow40b	8300.0	0.04	7672.0	0.02	–	0.02	8300.0	0.04	8300.0	0.04
air03	6.2e+05	0.04	6.6e+05	0.04	7.2e+05	0.04	1.2e+06	0.07	1.2e+06	0.07
air04	–	0.03	–	0.03	–	0.03	–	0.03	–	0.04
air05	–	0.03	–	0.03	–	0.08	–	0.04	–	0.04
app1-2	–	9.69	–	10.44	–	9.98	–	9.69	–	10.48
arki001	–	0.01	–	0.04	–	0.01	–	0.01	–	0.03
atlanta-ip	–	0.10	–	0.13	–	0.11	–	0.11	–	0.11
beasleyC3	–	0.01	–	0.01	–	0.02	–	0.01	–	0.01
bell3a	–	0.00	–	0.00	–	0.00	–	0.00	–	0.01
bell5	–	0.00	–	0.00	9.1e+06	0.00	–	0.01	–	0.00
bab5	–	2.06	–	2.18	–	1.82	–	0.59	–	0.48
biella1	9.5e+07	1.23	1.2e+08	1.24	3.2e+08	0.37	–	0.13	–	0.18
bienst2	–	0.00	–	0.01	66	0.01	–	0.00	–	0.00
binkar10_1	11244.2	0.01	11596.1	0.02	11505.7	0.01	11244.2	0.02	1e+04	0.02
blend2	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
bley_xl1	515.0	0.03	575.0	0.05	465.0	0.03	480.0	0.05	–	0.03
bnatt350	–	0.01	–	0.01	–	0.02	–	0.01	–	0.01
cap6000	-87646.0	0.06	-2e+05	0.11	-2.3e+05	0.09	-1.8e+05	0.05	-77813.0	0.06
core2536-691	–	197.52	–	194.44	12280.0	0.66	2e+04	39.20	–	293.35
cov1075	56	0.01	56	0.01	63	0.01	56	0.01	56	0.01
csched010	–	0.01	–	0.01	–	0.03	–	0.02	–	0.02
dano3mip	–	0.53	807.6	0.34	974.3	0.20	–	0.53	847.8	0.23
danoint	–	0.01	66	0.01	–	0.02	–	0.01	–	0.02
dcmulti	–	0.01	–	0.01	–	0.00	–	0.00	–	0.01
dfn-gwin-UUM	–	0.01	–	0.00	–	0.01	–	0.01	–	0.00
disctom	–	0.09	–	0.09	–	0.15	–	0.09	–	0.08

continued on next page

Table B.1 continued

	$\lvert\{b<0\}\rvert\downarrow$		$\lvert\cdot\rvert\downarrow$		random		$\lvert\{b<0\}\rvert\uparrow$		$\lvert\cdot\rvert\uparrow$	
Problem Name	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)
ds	1308.2	0.48	1308.2	0.50	2090.6	0.49	5418.6	0.49	5418.6	0.49
dsbmip	–	0.35	–	0.33	–	0.33	–	0.28	–	0.33
egout	667.1	0.00	663.5	0.00	624.9	0.00	667.1	0.00	626.6	0.00
eil33-2	1321.7	0.04	1321.7	0.04	2736.8	0.04	5050.2	0.03	5050.2	0.03
eilB101	2427.3	0.02	2427.3	0.02	3196.7	0.02	5e+03	0.02	5e+03	0.01
enigma	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
enlight13	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
fast0507	1.2e+05	0.47	1.2e+05	0.47	69159.0	0.48	8276.0	0.45	8276.0	0.43
fiber	–	0.00	–	0.01	–	0.01	–	0.00	–	0.01
fixnet6	–	0.01	–	0.01	–	0.01	–	0.01	–	0.02
flugpl	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
gen	1.1e+05	0.00	1.1e+05	0.01	1.2e+05	0.01	1.1e+05	0.01	–	0.01
gesa2-o	–	0.03	–	0.02	1.3e+08	0.02	1.3e+08	0.02	–	0.02
gesa2	4.7e+07	0.02	9.2e+07	0.02	–	0.02	1.5e+08	0.02	1.5e+08	0.02
gesa3	5.9e+07	0.02	–	0.02	–	0.03	1.5e+08	0.02	1.5e+08	0.02
gesa3_o	–	0.02	–	0.01	–	0.01	–	0.01	–	0.01
glass4	–	0.01	–	0.00	–	0.01	–	0.01	–	0.01
gmu-35-40	-5e+04	0.02	-5e+04	0.01	-5e+04	0.02	-5e+04	0.02	-5e+04	0.01
gt2	–	0.00	–	0.01	3e+05	0.00	–	0.00	–	0.00
harp2	-5.2e+07	0.01	–	0.01	–	0.00	-5.2e+07	0.00	–	0.01
iis-100-0-cov	46	0.02	46	0.01	64	0.01	88	0.02	88	0.02
iis-bupa-cov	98	0.03	98	0.02	224.0	0.02	315.0	0.02	315.0	0.02
iis-pima-cov	92	0.05	92	0.05	398.0	0.04	700.0	0.04	700.0	0.04
khb05250	1.6e+08	0.01	1.6e+08	0.01	1.4e+08	0.01	1.6e+08	0.01	1.6e+08	0.01
lectsched-4-obj	277.0	0.03	–	0.02	–	0.02	–	0.03	–	0.03
liu	6450.0	0.02	6450.0	0.02	6450.0	0.03	6450.0	0.03	6450.0	0.03
l152lav	–	0.02	–	0.01	–	0.02	–	0.03	–	0.04
lseu	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
m100n500k4r1	0	0.00	0	0.00	0	0.00	0	0.00	0	0.01
macrophage	1409.0	0.02	1e+03	0.02	1562.0	0.03	1582.0	0.02	1581.0	0.02
manna81	0	0.05	0	0.04	0	0.04	0	0.04	0	0.05
map18	0	0.19	0	0.19	0	0.19	0	0.19	0	0.18
map20	0	0.19	0	0.19	0	0.19	0	0.19	0	0.18

continued on next page

Table B.1 continued

	$\|\{b<0\}\|\downarrow$		$\|\cdot\|\downarrow$		random		$\|\{b<0\}\|\uparrow$		$\|\cdot\|\uparrow$	
Problem Name	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)
markshare1	7286.0	0.00	7286.0	0.00	7286.0	0.00	7286.0	0.01	7286.0	0.00
markshare2	10512.0	0.00	10512.0	0.00	10512.0	0.00	10512.0	0.01	10512.0	0.00
mas74	1.5e+05	0.00	1.5e+05	0.01	1.5e+05	0.01	1.5e+05	0.01	1.5e+05	0.01
mas76	1.5e+05	0.01	1.5e+05	0.01	1.5e+05	0.00	1.5e+05	0.01	1.5e+05	0.01
mcsched	4.8e+05	0.06	4.8e+05	0.07	4.3e+05	0.07	4.8e+05	0.10	4.8e+05	0.05
mik-250-1-100-1	0	0.02	0	0.01	0	0.01	0	0.01	0	0.01
mine-166-5	0	0.03	0	0.03	0	0.01	0	0.02	0	0.02
mine-90-10	0	0.03	0	0.01	0	0.01	0	0.03	0	0.01
misc03	–	0.00	–	0.00	–	0.01	–	0.00	–	0.00
misc06	13951.9	0.01	13951.9	0.02	13951.9	0.02	13951.9	0.03	13951.9	0.01
misc07	–	0.00	–	0.00	–	0.01	–	0.00	–	0.01
mitre	1.6e+05	0.03	1.3e+05	0.03	1.4e+05	0.04	1.5e+05	0.02	1.2e+05	0.02
mkc	0	0.21	0	0.19	0	0.12	0	0.21	0	0.04
mod008	1452.0	0.00	783.0	0.00	839.0	0.00	498.0	0.01	536.0	0.00
mod010	–	0.02	–	0.02	–	0.02	–	0.03	–	0.02
mod011	0	0.05	0	0.06	0	0.06	0	0.06	0	0.05
modglob	3.6e+07	0.01	3.6e+07	0.00	3.6e+07	0.00	3.6e+07	0.00	3.6e+07	0.00
momentum1	3.6e+05	0.11	–	0.13	–	0.06	–	0.08	4.8e+05	0.11
momentum2	–	0.13	–	0.16	–	0.20	–	0.24	–	0.12
momentum3	–	2.91	–	3.91	–	2.16	–	2.59	–	2.18
msc98-ip	–	0.06	–	0.06	–	0.11	–	0.05	–	0.09
mspp16	–	40.05	–	44.84	–	20.68	–	16.58	385.0	40.61
mzzv11	0	0.14	0	0.15	0	0.14	0	0.08	0	0.08
mzzv42z	0	0.19	0	0.38	0	0.27	0	0.16	0	0.14
n3div36	2.5e+06	0.69	3.6e+06	0.75	1.8e+06	0.80	2.9e+06	0.65	5.5e+06	0.74
n3seq24	9.7e+07	7.87	1.4e+08	8.45	7.8e+07	2.44	–	29.05	–	43.57
n4-3	–	0.06	3e+07	0.05	–	0.06	3e+07	0.06	–	0.04
neos-1109824	–	0.02	–	0.02	–	0.03	–	0.01	–	0.02
neos-1337307	–	0.03	–	0.13	–	0.31	–	0.06	–	0.06
neos-1396125	–	0.01	–	0.01	–	0.01	–	0.01	–	0.01
neos13	-28	0.96	-28	0.95	-28	0.96	-28	0.95	-28	0.95
neos-1601936	–	0.09	–	0.08	–	0.07	–	0.06	–	0.07
neos18	19	0.01	18	0.02	61	0.02	62	0.04	62	0.03

continued on next page

Table B.1 continued

	$\lvert\{b<0\}\rvert\downarrow$		$\lvert\cdot\rvert\downarrow$		random		$\lvert\{b<0\}\rvert\uparrow$		$\lvert\cdot\rvert\uparrow$	
Problem Name	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)
neos-476283	–	21.46	411.9	11.31	509.7	13.22	411.9	11.04	411.9	11.30
neos-686190	–	0.08	–	0.05	–	0.06	–	0.07	–	0.06
neos-849702	–	0.01	–	0.02	–	0.01	–	0.02	–	0.02
neos-916792	–	0.17	–	0.18	–	0.34	–	0.17	–	0.44
neos-934278	4.7e+05	0.41	1.8e+05	0.15	3.4e+05	0.11	3.6e+05	0.20	4.7e+05	0.48
net12	–	0.04	296.0	0.09	–	0.05	–	0.05	–	0.04
netdiversion	4.9e+06	20.10	–	6.83	–	16.27	–	48.24	–	9.32
newdano	–	0.00	–	0.01	80	0.01	–	0.01	–	0.00
noswot	-5	0.00	-5	0.00	-5	0.00	–	0.01	–	0.00
ns1208400	–	0.06	–	0.06	–	0.05	–	0.04	–	0.04
ns1688347	–	0.02	–	0.04	–	0.01	–	0.03	–	0.01
ns1758913	-236.8	2.09	-102.3	2.11	-226.7	2.84	-236.8	2.04	-460.5	6.03
ns1830653	–	0.02	–	0.03	–	0.03	–	0.03	–	0.03
nsrand-ipx	1.3e+06	0.05	2.7e+06	0.05	7.2e+05	0.05	8.5e+05	0.05	1.6e+05	0.06
nw04	29430.0	0.25	29430.0	0.24	57134.0	7.72	1.4e+05	19.05	1.4e+05	16.94
opm2-z7-s2	0	0.22	0	0.12	0	0.06	0	0.23	0	0.09
opt1217	0	0.01	0	0.00	0	0.00	0	0.00	0	0.01
p0033	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
p0201	12855.0	0.01	13210.0	0.00	10815.0	0.01	–	0.01	11260.0	0.01
p0282	9.1e+05	0.00	1.1e+06	0.00	6.5e+05	0.01	8.8e+05	0.01	6.5e+05	0.01
p0548	–	0.01	–	0.00	–	0.01	–	0.01	–	0.01
p2756	6595.0	0.06	1.3e+05	0.03	51211.0	0.06	1.5e+05	0.05	19419.0	0.08
pg5_34	0	0.02	0	0.02	0	0.01	0	0.02	0	0.02
pigeon-10	0	0.01	0	0.01	0	0.01	0	0.00	0	0.00
pk1	731.0	0.01	731.0	0.00	731.0	0.00	731.0	0.00	731.0	0.00
pp08a	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00
pp08aCUTS	–	0.00	–	0.00	–	0.01	–	0.00	–	0.00
protfold	-11	0.04	–	0.03	–	0.03	–	0.03	–	0.03
pw-myciel4	22	0.01	22	0.01	19	0.01	–	0.02	–	0.01
qiu	–	0.01	–	0.02	–	0.01	–	0.02	–	0.02
qnet1	2.2e+05	0.01	2.3e+05	0.01	4e+05	0.02	1.7e+05	0.02	4.3e+05	0.02
qnet1_o	78355.2	0.01	2.6e+05	0.02	3.8e+05	0.02	4.1e+05	0.01	4.1e+05	0.02
rail507	–	3599.02	–	3599.54	–	2209.66	8276.0	1.86	7674.0	1.42

continued on next page

Table B.1 continued

	$\|\{b<0\}\|\downarrow$		$\|\cdot\|\downarrow$		random		$\|\{b<0\}\|\uparrow$		$\|\cdot\|\uparrow$	
Problem Name	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)
ran16x16	6e+03	0.01	6e+03	0.01	6712.0	0.01	6e+03	0.00	6e+03	0.01
reblock67	0	0.01	0	0.01	0	0.01	0	0.01	0	0.00
rd-rplusc-21	–	1.52	–	0.25	–	0.28	–	1.10	–	0.40
rentacar	–	0.39	–	0.56	–	0.45	–	0.51	–	0.44
rgn	445.0	0.00	445.0	0.01	445.0	0.00	445.0	0.00	445.0	0.00
rmatr100-p10	817.0	0.07	763.0	0.07	772.0	0.07	817.0	0.07	975.0	0.07
rmatr100-p5	1414.0	0.14	1374.0	0.13	1555.0	0.14	1414.0	0.14	2e+03	0.17
rmine6	0	0.05	0	0.01	0	0.02	0	0.06	0	0.01
rocII-4-11	–	0.03	–	0.03	–	0.03	–	0.02	–	0.02
rococoC10-001000	2.1e+05	0.03	–	0.01	34348.0	0.03	5e+04	0.03	54116.0	0.03
roll3000	–	0.04	–	0.06	–	0.02	–	0.02	–	0.04
rout	2375.2	0.01	2375.2	0.01	2375.2	0.01	2375.2	0.02	2375.2	0.00
satellites1-25	97	0.11	79	0.10	–	0.12	80	0.14	–	0.10
set1ch	1e+05	0.01	1e+05	0.01	1.1e+05	0.00	1e+05	0.01	1e+05	0.01
seymour	1269.0	0.02	1269.0	0.03	1204.0	0.03	1276.0	0.04	1276.0	0.03
sp97ar	2.6e+10	0.19	4.6e+10	0.19	2.3e+10	0.18	2.7e+10	0.20	7.8e+09	0.22
sp98ic	1.1e+10	0.20	1.6e+10	0.20	6.2e+09	0.22	8.2e+09	0.18	5.2e+09	0.21
sp98ir	5.2e+08	0.03	4.4e+08	0.03	–	0.03	–	0.05	4.4e+08	0.03
stein27	23	0.01	23	0.00	21	0.00	23	0.00	23	0.00
stein45	38	0.00	38	0.00	37	0.00	39	0.00	39	0.01
stp3d	–	0.90	–	1.17	–	1.93	–	0.89	–	12.95
swath	713.2	0.06	–	0.11	–	0.13	928.6	0.10	1e+03	0.08
t1717	–	0.13	–	0.12	–	0.09	–	0.09	–	0.10
tanglegram1	33625.0	0.25	33564.0	0.24	32825.0	0.30	33807.0	0.24	33807.0	0.24
tanglegram2	4172.0	0.06	4158.0	0.05	4241.0	0.05	4269.0	0.05	4267.0	0.05
timtab1	–	0.01	–	0.00	–	0.01	–	0.01	–	0.00
timtab2	–	0.00	–	0.00	–	0.01	–	0.01	–	0.00
tr12-30	–	0.02	–	0.02	–	0.03	–	0.02	–	0.02
triptim1	–	0.17	–	0.18	–	0.23	–	0.24	–	0.61
unitcal_7	–	0.12	–	0.11	–	0.12	–	0.10	–	1.04
vpm1	23	0.01	–	0.00	–	0.00	23	0.01	24	0.00
vpm2	–	0.00	–	0.00	–	0.01	–	0.00	–	0.00
vpphard	–	0.37	–	0.21	2e+04	5.39	–	7.41	–	4.71

continued on next page

Table B.1 continued

	$\lvert\{b<0\}\rvert\downarrow$		$\lvert\cdot\rvert\downarrow$		random		$\lvert\{b<0\}\rvert\uparrow$		$\lvert\cdot\rvert\uparrow$	
Problem Name	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)
zib54-UUE	–	0.02	–	0.02	–	0.04	–	0.02	–	0.02

Table B.2.: Shift-and-Propagate: results of the root experiment for all three settings in terms of heuristic/root solving time and objective value for all 163 instances.

	SandP		RandI		Both	
Problem Name	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)
10teams	–	0.00/0.18	–	0.00/0.32	–	0.02/0.33
30n20b8	–	0.02/9.60	–	0.01/9.59	–	0.04/9.55
a1c1s1	–	0.02/0.27	–	0.02/0.28	–	0.03/0.29
acc-tight5	–	0.00/2.14	–	0.01/2.00	–	0.01/2.15
aflow30a	4606.0	0.01/0.19	–	0.00/0.09	4606.0	0.03/0.21
aflow40b	8300.0	0.03/0.81	–	0.00/0.78	8300.0	0.07/0.84
air03	6.2e+05	0.04/5.08	–	0.02/5.05	6.2e+05	0.06/5.32
air04	–	0.02/3.63	–	0.00/3.51	–	0.05/3.55
air05	–	0.04/1.22	–	0.00/1.11	–	0.04/1.20
app1-2	–	9.70/24.08	-23	0.07/13.82	–	9.86/24.23
arki001	–	0.01/0.38	–	0.00/0.39	–	0.01/0.38
atlanta-ip	–	0.10/18.18	–	0.06/18.06	–	0.15/18.27
beasleyC3	–	0.00/0.07	951.0	0.05/0.19	951.0	0.05/0.17
bell3a	–	0.00/0.01	9.2e+05	0.00/0.00	9.2e+05	0.01/0.02
bell5	–	0.00/0.01	–	0.00/0.03	–	0.00/0.03
bab5	–	2.05/18.84	–	0.03/16.92	–	2.13/19.19
biella1	9.5e+07	1.23/5.70	–	0.00/4.50	9.5e+07	1.86/6.30
bienst2	–	0.01/0.02	–	0.00/0.01	–	0.01/0.05
binkar10_1	11244.2	0.02/0.13	–	0.00/0.12	11244.2	0.01/0.13
blend2	–	0.01/0.04	–	0.00/0.04	–	0.01/0.04
bley_xl1	515.0	0.05/351.94	–	0.00/310.20	275.0	0.05/311.14
bnatt350	–	0.01/0.75	–	0.00/0.74	–	0.02/0.75
cap6000	-87646.0	0.06/0.73	-2.5e+06	0.16/0.83	-2.5e+06	0.20/0.98
core2536-691	–	1.56/13.46	795.0	0.09/12.15	795.0	1.65/13.68
cov1075	56	0.01/0.30	26	0.00/0.31	26	0.02/0.31
csched010	–	0.01/0.27	–	0.00/0.27	–	0.01/0.27
dano3mip	–	0.53/22.28	–	0.03/21.43	–	0.55/22.10
danoint	–	0.01/0.60	–	0.00/0.51	–	0.01/0.60
dcmulti	–	0.01/0.03	–	0.01/0.04	–	0.01/0.03
dfn-gwin-UUM	–	0.01/0.03	2.3e+05	0.00/0.03	2.3e+05	0.00/0.03
disctom	–	0.05/1.81	–	0.01/1.57	–	0.06/1.75
ds	1308.2	0.51/21.53	–	0.08/21.30	1308.2	1.10/22.18
dsbmip	–	0.16/0.42	–	0.01/0.37	–	0.35/0.59
egout	667.1	0.00/0.01	625.3	0.00/0.01	625.3	0.00/0.00
eil33-2	1321.7	0.03/0.50	–	0.01/0.46	1321.7	0.09/0.55
eilB101	2427.3	0.02/0.41	–	0.00/0.41	2427.3	0.04/0.43
enigma	–	0.00/0.01	–	0.00/0.01	–	0.00/0.01
enlight13	–	0.00/0.02	–	0.00/0.02	–	0.01/0.02
fast0507	1.2e+05	0.48/13.71	240.0	0.57/13.87	257.0	0.94/14.22
fiber	–	0.01/0.04	–	0.00/0.04	–	0.02/0.04
fixnet6	–	0.01/0.04	4536.0	0.02/0.05	4536.0	0.03/0.07
flugpl	–	0.00/0.01	–	0.00/0.00	–	0.00/0.01
gen	1.1e+05	0.01/0.06	–	0.00/0.05	1.1e+05	0.02/0.07
gesa2-o	–	0.00/0.05	–	0.00/0.10	–	0.02/0.12
gesa2	4.7e+07	0.02/0.16	1.9e+08	0.01/0.13	4.6e+07	0.05/0.18
gesa3	5.9e+07	0.02/0.14	1.9e+08	0.00/0.12	5.9e+07	0.02/0.14
gesa3_o	–	0.02/0.11	–	0.00/0.09	–	0.02/0.10
glass4	–	0.01/0.04	–	0.00/0.04	–	0.01/0.04
gmu-35-40	-5e+04	0.01/0.22	-1.5e+06	0.00/0.21	-1.5e+06	0.01/0.20
gt2	–	0.00/0.01	–	0.00/0.01	–	0.00/0.01
harp2	-5.2e+07	0.01/0.18	–	0.01/0.18	-5.6e+07	0.06/0.24
iis-100-0-cov	46	0.02/0.57	35	0.01/0.49	36	0.02/0.57
iis-bupa-cov	98	0.01/1.13	48	0.01/1.30	47	0.03/1.30
iis-pima-cov	92	0.05/1.26	44	0.03/1.50	42	0.05/1.39
khb05250	1.6e+08	0.01/0.03	1.3e+08	0.00/0.01	1.3e+08	0.01/0.03

continued on next page

Table B.2 continued

	SandP		RandI		Both	
Problem Name	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)
lectsched-4-obj	–	0.02/1.13	–	0.01/1.29	–	0.02/1.14
liu	6450.0	0.04/0.10	–	0.00/0.05	6450.0	0.05/0.12
l152lav	–	0.02/0.15	–	0.00/0.15	–	0.02/0.17
lseu	–	0.00/0.01	–	0.00/0.01	–	0.00/0.01
m100n500k4r1	0	0.01/0.03	-18	0.00/0.03	-18	0.02/0.04
macrophage	1409.0	0.03/0.13	609.0	0.02/0.13	458.0	0.03/0.15
manna81	0	0.04/0.26	-13162.0	0.07/0.19	-13162.0	0.11/0.31
map18	0	0.19/7.88	0	0.17/7.11	0	0.30/8.00
map20	0	0.19/7.37	0	0.18/6.51	0	0.32/7.62
markshare1	7286.0	0.00/0.01	125.0	0.00/0.01	125.0	0.00/0.01
markshare2	10512.0	0.00/0.00	161.0	0.00/0.00	161.0	0.00/0.01
mas74	1.5e+05	0.01/0.02	–	0.00/0.01	1.5e+05	0.02/0.02
mas76	1.5e+05	0.01/0.02	–	0.00/0.01	1.5e+05	0.01/0.01
mcsched	4.8e+05	0.06/0.79	–	0.01/0.80	4.8e+05	0.08/0.81
mik-250-1-100-1	0	0.01/0.04	0	0.01/0.02	0	0.01/0.04
mine-166-5	0	0.03/1.66	-7.3e+06	0.02/1.79	-7.3e+06	0.03/1.75
mine-90-10	0	0.04/1.14	-1.6e+07	0.02/1.13	-1.6e+07	0.03/1.24
misc03	–	0.00/0.04	–	0.00/0.04	–	0.00/0.05
misc06	13951.9	0.02/0.13	–	0.00/0.13	13951.9	0.03/0.13
misc07	–	0.00/0.09	–	0.00/0.11	–	0.01/0.11
mitre	1.6e+05	0.03/4.94	–	0.00/5.03	1.4e+05	0.04/4.97
mkc	0	0.21/0.49	0	0.11/0.40	0	0.25/0.53
mod008	1452.0	0.00/0.01	308.0	0.00/0.02	308.0	0.01/0.02
mod010	–	0.02/0.12	–	0.01/0.20	–	0.02/0.21
mod011	0	0.05/0.50	-4.3e+07	0.04/0.48	-4.3e+07	0.07/0.53
modglob	3.6e+07	0.00/0.02	2.1e+07	0.00/0.01	2.1e+07	0.00/0.02
momentum1	3.6e+05	0.11/5.42	–	0.02/5.47	3.6e+05	0.12/5.47
momentum2	–	0.13/11.70	–	0.01/11.43	–	0.13/11.07
momentum3	–	1.82/387.71	–	0.05/375.27	–	1.76/379.42
msc98-ip	–	0.07/6.41	–	0.01/6.35	–	0.08/6.24
mspp16	–	40.01/758.94	–	0.83/737.06	–	40.85/764.70
mzzv11	0	0.13/41.75	0	0.02/41.60	0	0.15/41.85
mzzv42z	0	0.19/9.44	0	0.10/9.39	0	0.24/9.42
n3div36	2.5e+06	0.67/3.96	1.8e+05	0.38/3.70	1.6e+05	1.09/4.40
n3seq24	9.7e+07	8.09/46.82	–	0.22/40.31	1.6e+05	8.96/48.01
n4-3	–	0.06/0.34	14275.0	0.00/0.06	14275.0	0.06/0.19
neos-1109824	–	0.02/0.86	–	0.00/0.86	–	0.01/0.98
neos-1337307	–	0.03/2.50	–	0.00/2.49	–	0.04/2.48
neos-1396125	–	0.01/1.39	–	0.00/1.47	–	0.01/1.34
neos13	-28	0.95/3.15	–	0.03/2.31	-28	0.99/3.17
neos-1601936	–	0.08/4.98	–	0.00/5.05	–	0.08/4.85
neos18	19	0.01/0.16	57	0.01/0.26	19	0.03/0.26
neos-476283	–	21.91/84.89	–	0.02/63.30	–	21.51/84.03
neos-686190	–	0.09/0.33	–	0.01/0.26	–	0.08/0.33
neos-849702	–	0.01/1.10	–	0.01/1.16	–	0.02/1.15
neos-916792	–	0.17/0.87	–	0.00/0.80	–	0.17/0.92
neos-934278	4.7e+05	0.40/19.17	3.4e+10	0.04/18.93	4.7e+05	0.47/19.32
net12	–	0.04/3.94	–	0.02/3.96	–	0.05/4.05
netdiversion	4.9e+06	22.09/675.36	–	0.13/648.13	4.9e+06	28.70/674.29
newdano	–	0.01/0.03	–	0.00/0.02	–	0.00/0.04
noswot	-5	0.00/0.01	–	0.00/0.01	-6	0.00/0.02
ns1208400	–	0.07/2.30	–	0.00/2.33	–	0.06/2.23
ns1688347	–	0.02/5.13	–	0.01/5.23	–	0.02/5.27
ns1758913	–	1.36/131.32	–	0.04/130.64	–	1.47/134.89
ns1830653	–	0.01/0.66	–	0.00/0.58	–	0.02/0.82
nsrand-ipx	1.3e+06	0.05/1.50	–	0.01/1.27	70720.0	0.09/1.42
nw04	29430.0	0.26/11.70	–	0.08/11.61	29430.0	0.93/12.37
opm2-z7-s2	0	0.23/13.68	-3685.0	0.14/13.61	-3685.0	0.33/13.88
opt1217	0	0.00/0.02	–	0.00/0.03	0	0.01/0.04

continued on next page

Table B.2 continued

	SandP		RandI		Both	
Problem Name	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)	$c^T\tilde{x}$	t (s)
p0033	–	0.00/0.01	–	0.00/0.00	–	0.00/0.01
p0201	12855.0	0.00/0.06	–	0.00/0.06	11295.0	0.02/0.06
p0282	9.1e+05	0.01/0.03	2.7e+05	0.00/0.02	2.7e+05	0.00/0.03
p0548	–	0.01/0.04	–	0.00/0.04	–	0.02/0.05
p2756	6595.0	0.05/0.34	–	0.01/0.29	5e+03	0.07/0.34
pg5_34	0	0.02/0.17	0	0.01/0.16	0	0.02/0.19
pigeon-10	0	0.01/0.05	0	0.01/0.04	0	0.01/0.05
pk1	731.0	0.00/0.01	–	0.00/0.01	731.0	0.00/0.01
pp08a	–	0.01/0.01	14600.0	0.00/0.01	14600.0	0.00/0.01
pp08aCUTS	–	0.00/0.02	16630.4	0.00/0.01	16630.4	0.02/0.02
protfold	–	0.02/1.44	–	0.00/1.49	–	0.03/1.43
pw-myciel4	22	0.01/1.02	–	0.00/1.07	22	0.01/0.98
qiu	–	0.00/0.10	1805.2	0.00/0.10	1805.2	0.01/0.11
qnet1	2.2e+05	0.01/0.24	–	0.01/0.24	26659.3	0.04/0.26
qnet1_o	78355.2	0.02/0.06	28462.1	0.01/0.05	17842.7	0.01/0.06
rail507	–	8.25/20.69	216.0	0.43/13.02	216.0	8.98/21.27
ran16x16	6e+03	0.01/0.02	4333.0	0.01/0.02	4333.0	0.03/0.04
reblock67	0	0.02/0.91	-2.6e+06	0.01/0.84	-2.6e+06	0.03/0.85
rd-rplusc-21	–	1.49/58.94	–	0.00/57.58	–	1.47/58.81
rentacar	–	0.58/1.13	–	0.02/1.16	–	0.39/0.96
rgn	445.0	0.01/0.02	–	0.00/0.00	445.0	0.01/0.02
rmatr100-p10	817.0	0.07/2.52	–	0.01/2.59	817.0	0.08/2.50
rmatr100-p5	1414.0	0.14/4.17	–	0.01/4.16	1414.0	0.15/4.23
rmine6	0	0.05/1.54	-90	0.01/1.35	-90	0.07/1.53
rocII-4-11	–	0.02/6.18	–	0.01/6.17	–	0.04/6.15
rococoC10-001000	2.1e+05	0.03/0.31	27042.0	0.03/0.33	24044.0	0.05/0.34
roll3000	–	0.03/0.61	–	0.01/0.62	–	0.04/0.60
rout	2375.2	0.01/0.09	2375.2	0.00/0.08	2375.2	0.01/0.09
satellites1-25	97	0.10/39.36	–	0.00/37.12	97	0.12/39.79
set1ch	1e+05	0.01/0.03	1.1e+05	0.02/0.04	1e+05	0.02/0.05
seymour	1269.0	0.03/1.19	496.0	0.03/1.04	500.0	0.06/1.18
sp97ar	2.6e+10	0.18/5.67	–	0.05/5.57	9.7e+08	0.39/5.66
sp98ic	1.1e+10	0.19/3.80	5.4e+08	0.22/3.94	8.3e+08	0.42/3.93
sp98ir	5.2e+08	0.03/1.19	–	0.00/1.18	3.1e+08	0.04/1.38
stein27	23	0.00/0.01	19	0.00/0.00	19	0.01/0.01
stein45	38	0.01/0.02	33	0.00/0.01	32	0.00/0.02
stp3d	–	0.81/2215.49	–	0.21/2206.18	–	1.19/2239.59
swath	713.2	0.06/0.24	–	0.02/0.20	713.2	0.33/0.52
t1717	–	0.08/6.82	–	0.06/6.86	–	0.14/6.77
tanglegram1	33625.0	0.25/2.27	7798.0	0.72/2.76	5406.0	0.85/2.79
tanglegram2	4172.0	0.05/0.36	2122.0	0.15/0.38	535.0	0.16/0.49
timtab1	–	0.00/0.03	–	0.00/0.02	–	0.00/0.02
timtab2	–	0.00/0.06	–	0.00/0.05	–	0.01/0.05
tr12-30	–	0.02/0.12	–	0.00/0.10	–	0.02/0.12
triptim1	–	0.17/111.54	–	0.03/112.41	–	0.21/132.15
unitcal_7	–	0.10/17.19	–	0.03/17.86	–	0.14/17.25
vpm1	23	0.00/0.00	24	0.01/0.02	23	0.01/0.01
vpm2	–	0.00/0.03	–	0.00/0.03	–	0.00/0.02
vpphard	–	0.27/11.66	–	0.05/11.54	–	0.34/11.67
zib54-UUE	–	0.02/0.21	1.8e+07	0.00/0.21	1.8e+07	0.02/0.21

Table B.3.: comparison of different Feasibility Pump versions

	default		cuts		hierarchy		hessian		objective		simple	
	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]
4stufen	–	3.5	139803.4	8.5	–	4.0	132723.1	3.6	–	4.1	–	1.6
alan	3.914297	0.1	3.914297	0.1	3.914297	0.1	3.914297	0.1	3.263545	0.1	3.914297	0.2
batchdes	181201.6	0.2	181201.6	0.1	181201.6	0.1	181201.6	0.2	167427.6	0.2	181201.6	1.5
batch	309205.3	0.9	309205.3	0.9	309205.3	0.9	309205.3	0.7	284739.1	1.0	–	0.7
beuster	–	2.3	–	2.3	–	2.2	–	2.9	–	2.9	–	3.2
cecil_13	-108059.7	8.0	-97072.09	8.0	-108174.6	7.9	-108154.3	12.0	-37413.88	54.7	–	9.2
contvar	3114960	12.1	5094589	15.1	5629578	22.0	3187022	27.4	1994767	15.1	2206609	7.5
csched1	-27781.88	0.4	-24663.95	0.4	-27781.02	0.2	-27781.87	0.3	-263216.6	0.5	–	0.9
csched2	-114433.1	3.7	-116907	3.5	-114250.2	3.5	-113832.3	4.7	-113541.9	4.7	–	5.7
deb10	–	17.7	–	16.8	–	19.6	–	36.6	–	37.0	–	4.2
deb6	251.673	35.8	251.673	34.7	251.673	38.0	251.673	116.9	218.3573	118.5	–	41.9
deb7	–	220.1	–	242.5	–	216.1	–	6156.9	–	6148.1	–	120.1
deb8	–	223.6	–	223.6	–	221.1	–	6384.7	–	6332.8	–	199.2
deb9	–	233.5	–	214.1	–	210.7	–	6245.9	–	6235.3	–	143.8
detf1	–	4700.1	–	4821.3	–	5377.3	–	7200.0	–	7200.0	16462.46	3006.2
du-opt5b	319.0127	3.1	676898.2	2.8	319.0127	3.0	214.7467	3.0	577.9	4.4	–	4.4
du-opt5	754.4937	2.9	2646.166	2.7	754.4937	2.7	92.84725	2.9	71.87055	3.0	5460.092	3.1
du-optb	50259.35	2.5	128925.3	2.2	50259.35	2.3	45664.47	2.3	38726.67	1.7	–	2.5
du-opt	53.22028	3.8	53.22028	3.7	53.22028	3.6	53.22028	3.7	23.61013	3.9	53.22028	3.0
eg_all_sb	8.016916	7200.0	8.145184	7200.0	8.016916	7200.0	8.016916	7200.0	8.016916	7200.0	–	7200.0
eg_all_s	10.76562	7200.0	10.76562	7200.0	10.76562	7200.0	10.76562	7200.0	10.76562	7200.0	10.76562	7200.0
eg_disc2_sb	2.770486	7200.0	9.366156	7200.0	2.770486	7200.0	2.770486	7200.0	2.770486	7200.0	11.02987	7200.0
eg_disc2_s	-7.20939	7195.3	-7.20939	7195.2	-7.20939	7194.9	-7.20939	7195.3	-7.20939	7195.3	3.772146	7200.0
eg_disc_sb	-6.642528	7196.7	13.7239	7200.0	-6.642528	7197.3	-6.642528	7196.7	-6.642528	7196.7	20.26977	7200.0
eg_disc_s	–	7200.0	–	7200.0	–	7200.0	–	7200.0	–	7200.0	–	7200.0
eg_int_sb	12.58941	7200.0	12.58941	7200.0	12.58941	7200.0	12.58941	7200.0	12.58941	7200.0	–	7200.0
eg_int_s	1.726767	7200.0	1.726767	7200.0	1.726767	7200.0	1.726767	7200.0	1.726767	7200.0	11.37772	7200.0
elf	0.45	0.3	0.45	0.3	0.45	0.3	0.45	0.3	0	0.3	–	0.6

Table B.3 continued

	default		cuts		hierarchy		hessian		objective		simple	
	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]
eniplac	-118842.4	0.7	-122984.2	0.7	-126473.6	0.6	-117465.2	0.8	-132057	1.0	–	2.9
enpro48	209070.4	1.0	1280053	1.0	209070.4	0.9	208672	1.3	199586.2	2.5	862597.2	0.9
enpro56	821142.9	1.1	–	1.6	–	1.4	–	2.5	265311.6	2.7	–	1.1
ex1221	7.66718	0.1	7.66718	0.1	7.66718	0.1	7.66718	0.1	7.66718	0.1	7.66718	0.1
ex1222	1.076543	0.1	1.076543	0.1	1.076543	0.1	1.076543	0.1	1.076543	0.1	1.076543	0.1
ex1223a	8.563597	0.1	8.563597	0.1	5.817607	0.1	5.585952	0.1	5.567105	0.1	4.579582	0.1
ex1223b	4.579582	0.1	4.579582	0.1	4.579582	0.1	4.579582	0.1	4.579582	0.2	4.579582	0.1
ex1223	4.579582	0.1	4.579582	0.1	4.579582	0.1	4.579582	0.1	4.579582	0.1	4.579582	0.1
ex1224	-0.9049615	0.1	-0.67915	0.1	-0.9049615	0.1	-0.9049615	0.2	-0.9049615	0.2	-0.5289633	0.1
ex1225	31	0.1	31	0.1	31	0.1	31	0.1	31	0.1	31	0.1
ex1226	-17	0.1	-17	0.5	-17	0.1	-17	0.2	-17	0.2	-17	0.1
ex1233	529393.5	0.7	529393.5	0.6	413989.8	0.6	529393.5	0.8	228363.2	0.9	–	0.8
ex1243	358172.6	0.4	457457.1	0.2	359373.9	0.4	364156.2	0.5	294180.7	0.6	–	0.8
ex1244	587384.5	0.8	571656.4	0.8	592957.4	0.8	598001.4	0.8	517323.8	1.0	–	2.2
ex1252ab	–	2.5	–	1.4	–	1.7	–	2.2	–	2.5	–	1.6
ex1252a	141260.7	1.0	–	0.7	139508.3	0.9	1872.139	0.8	–	1.4	151223	0.4
ex1252	204635.1	1.3	–	1.2	204635.1	1.4	83205.45	1.6	155113.1	1.1	105320.6	1.3
ex1263ab	44.3	0.6	54	1.0	44.3	0.5	44.3	0.7	44.3	0.7	–	0.6
ex1263a	–	1.1	35	0.4	–	0.8	–	1.1	–	1.1	–	0.1
ex1263	48.6	1.3	–	0.8	–	1.0	45.6	1.2	–	1.5	–	0.2
ex1264ab	20.3	0.9	22	0.6	–	1.0	20.3	0.7	20.3	0.8	–	0.6
ex1264a	–	1.0	36	0.2	11	0.7	11	0.7	11	0.6	–	0.1
ex1264	28.6	0.8	35	0.4	–	1.2	35	0.9	35	1.0	–	0.7
ex1265ab	22.3	0.8	22.5	1.2	21.3	1.0	48.5	0.9	20.3	0.9	–	0.8
ex1265a	–	0.9	–	0.6	–	1.0	–	1.0	–	0.9	–	0.5
ex1265	15.1	0.6	36.5	0.6	15.1	0.6	15.1	0.8	15.1	0.8	–	0.8
ex1266ab	–	1.7	24.5	3.4	23.3	1.0	20.3	1.4	21.3	1.2	–	0.8
ex1266a	–	1.1	–	2.5	–	1.2	–	1.3	–	1.4	–	0.6
ex1266	23.3	1.4	25.3	1.2	23.3	1.3	31.6	2.0	–	2.1	–	1.2
ex3	104.2945	0.1	104.2945	0.2	104.2945	0.1	95.16263	0.2	74.2163	0.2	104.2945	0.1

Table B.3 continued

	default		cuts		hierarchy		hessian		objective		simple	
	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]
ex4	-7.313347	1.6	-7.313347	1.7	-7.313347	1.5	-7.313347	1.7	-7.326188	1.6	212801.7	2.0
fac1	160912600	0.2	160912600	0.2	160912600	0.1	160912600	0.2	160912600	0.2	–	0.1
fac2	376262300	0.4	331837500	0.5	376262300	0.5	376262300	0.2	331837500	0.4	–	0.4
fac3	31982310	0.2	36423810	0.1	31982310	0.3	31982310	0.3	36423810	0.5	–	0.4
feedtray2	8.869614e-08	1.6	8.869614e-08	1.7	8.869614e-08	1.7	8.869614e-08	1.7	8.869614e-08	1.7	3.878215e-08	1.0
feedtray	–	8.0	–	8.0	–	8.0	–	8.1	–	8.3	–	8.1
fuel	12228.52	0.1	12228.52	0.1	12228.52	0.1	12228.52	0.2	12228.52	0.1	12228.52	0.1
fuzzy	—	—	—	—	—	—	—	—	—	—	—	—
gasnet	11631750	5.8	–	4.1	11696360	3.0	11631750	5.9	–	3.8	7623861	2.7
gastrans	89.08584	0.4	89.08584	0.2	89.08584	0.1	89.08584	0.5	89.08584	0.5	–	0.9
gbd	2.2	0.1	2.2	0.1	2.2	0.1	2.2	0.1	2.2	0.1	2.2	0.1
gear2	0.01087418	0.1	0.7322579	0.1	0.01087418	0.1	0.01087418	0.2	0.01087418	0.2	–	0.3
gear3b	3.534003e-06	0.1	0.0648276	0.1	3.534003e-06	0.1	3.534003e-06	0.3	3.534003e-06	0.3	0.0006090035	0.2
gear3	1.164845e-05	0.1	5.328449e-05	0.1	1.164845e-05	0.1	1.164845e-05	0.1	1.164845e-05	0.1	1.164845e-05	0.1
gear4b	2387.342	0.2	418220.7	0.1	4067.894	0.3	49217.05	0.3	49217.05	0.3	418220.7	0.2
gear4	120.6752	0.1	55720.68	0.1	120.6752	0.1	120.6752	0.1	120.6752	0.1	120.6752	0.1
gearb	7.102863e-06	0.2	0.002835726	0.1	7.102863e-06	0.1	7.102863e-06	0.2	7.102863e-06	0.2	0.002835726	0.2
gear	1.164845e-05	0.1	5.328449e-05	0.1	1.164845e-05	0.1	1.164845e-05	0.1	1.164845e-05	0.1	1.164845e-05	0.1
gkocis	-1.720972	0.1	-1.720972	0.1	-1.720972	0.1	-1.720972	0.1	-1.720972	0.1	0	0.1
hda	—	—	—	—	—	—	–	0.1	–	0.1	—	—
hmittelman	13	0.2	16	0.1	13	0.2	13	0.2	13	0.2	13	0.3
johnall	-224.7302	32.7	-224.7302	37.8	-224.7302	19.5	-224.7302	296.0	-224.7302	294.6	-224.7302	2.5
lop97icb	4225.063	2615.0	5455.97	676.9	4393.579	2209.1	4664.303	6161.4	4419.913	6050.0	–	742.5
lop97ic	4301.931	1233.4	4527.804	201.7	4327.707	666.4	4296.261	3282.5	4321.452	2139.5	–	223.3
lop97icxb	4160.434	91.5	4741.518	57.9	4729.826	173.4	4160.434	397.2	4160.297	376.6	–	100.7
lop97icx	4297.165	30.3	5341.814	7.8	4366.357	30.3	4297.165	42.3	4297.165	46.4	–	17.1
meanvarx	18.03777	0.2	18.03777	0.2	18.03777	0.2	15.67514	0.2	16.56943	0.2	14.19703	0.2
minlphix	348.1008	0.1	348.1008	0.2	648.3886	0.2	451.3111	0.2	451.3111	0.3	–	0.5
nous1	–	1.1	0.99782	0.7	–	1.0	–	1.2	–	10.4	–	0.9
nous2	0.23947	0.8	0.23947	0.6	0.23947	0.7	0.23947	0.8	0.23947	0.7	0.6259675	0.4

Table B.3 continued

	default		cuts		hierarchy		hessian		objective		simple	
	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]
nuclear104	–	7200.0	–	6869.9	–	7200.0	—	—	—	—	–	6870.5
nuclear10a	–	7200.0	–	7193.6	–	7200.0	–	7200.0	–	7200.0	–	7193.8
nuclear10b	-1.141958	7184.1	-1.141958	7185.5	-1.155471	7189.1	–	7200.0	–	7200.0	–	7145.4
nuclear14a	-1.129559	121.8	-1.129559	121.4	-1.129559	119.4	-1.129559	149.8	-12.23989	151.1	-1.129559	117.5
nuclear14b	–	80.8	–	82.7	-1.078322	76.0	–	334.7	–	341.4	–	62.7
nuclear14	–	76.9	–	89.5	–	77.2	–	888.5	–	882.6	–	68.2
nuclear24a	-1.129559	123.0	-1.129559	120.3	-1.129559	118.0	-1.129559	149.1	-12.23989	150.4	-1.129559	118.0
nuclear24b	–	79.9	–	82.8	-1.078322	75.7	–	336.6	–	341.3	–	62.3
nuclear24	–	77.2	–	89.4	–	76.9	–	886.8	–	876.3	–	68.1
nuclear25a	–	458.2	–	157.2	–	431.8	–	706.5	–	838.9	–	146.5
nuclear25b	–	100.7	–	114.5	-1.07108	67.0	–	414.1	–	414.6	–	76.9
nuclear25	–	74.8	–	104.1	–	94.0	–	1169.1	–	1202.5	–	83.3
nuclear49a	–	7200.0	–	1574.0	–	4191.4	–	7200.0	–	7200.0	–	1443.4
nuclear49b	–	853.8	–	842.1	–	867.8	–	7200.0	–	7200.0	–	706.2
nuclear49	–	993.1	–	1035.7	–	1041.4	–	7200.0	–	7200.0	–	1160.2
nuclearva	–	9.8	–	9.3	–	9.0	–	38.7	–	26.5	–	6.6
nuclearvb	-1.010788	6.6	-1.010788	6.7	-1.010788	6.8	-1.010788	8.3	–	29.8	–	5.8
nuclearvc	-0.992737	5.9	-0.992737	5.3	-0.992737	5.4	-0.992737	7.5	–	38.2	–	5.2
nuclearvd	-1.020933	7.4	-1.021023	6.7	-1.020746	6.8	-1.020168	8.5	–	40.5	–	6.9
nuclearve	-1.017918	9.8	-1.018722	10.3	-1.018826	10.1	-1.018722	11.4	–	43.4	–	9.9
nuclearvf	-1.003303	6.2	-1.003719	6.0	-1.003354	6.1	-1.003472	8.2	–	48.0	–	6.0
nvs01b	18.27142	0.2	18.27142	0.4	18.27142	0.1	18.27142	0.1	18.27142	0.1	18.27142	0.1
nvs01	179.6416	0.4	179.6416	0.1	179.6416	0.4	179.6416	0.5	179.6416	0.4	179.6416	0.1
nvs02b	–	0.8	–	0.6	–	0.6	–	0.7	–	0.7	–	0.2
nvs02	5.964185	0.1	5.964185	0.3	5.964185	0.1	5.964185	0.2	5.964185	0.2	5.964185	0.2
nvs03b	20	0.1	20	0.1	20	0.1	20	0.1	20	0.1	20	0.1
nvs03	16	0.1	25	0.1	16	0.1	16	0.1	16	0.1	36	0.1
nvs04b	2.12	0.1	2.12	0.1	2.12	0.1	2.12	0.2	2.12	0.2	0.01249441	0.1
nvs04	0.72	0.2	0.72	0.2	0.72	0.2	0.72	0.2	0.72	0.2	0.72	0.2
nvs05b	103.1209	0.6	2081.138	1.3	58.00012	0.5	10.19712	0.6	32.52952	0.7	–	1.2

Table B.3 continued

	default		cuts		hierarchy		hessian		objective		simple	
	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]
nvs05	13.45624	2.1	6.061651	1.3	13.45624	2.0	6.911338	2.8	42.01389	3.0	9.130255	1.0
nvs06b	1.770313	0.2	1.770313	0.1	1.770313	0.1	1.770313	0.3	1.770313	0.3	1.770313	0.1
nvs06	1.770313	0.2	1.860494	0.2	1.770313	0.2	1.770313	0.2	1.770313	0.2	1.770313	0.2
nvs07b	4	0.2	4	0.1	4	0.1	4	0.2	4	0.2	4	0.2
nvs07	4	0.1	4	0.1	4	0.1	4	0.1	4	0.1	4	0.1
nvs08b	31.8762	0.2	1580.286	0.2	31.8762	0.2	20.28699	0.3	20.28699	0.3	41629	0.3
nvs08	34.8762	0.1	1662.49	0.5	34.8762	0.1	45.632	0.1	23.44973	0.1	1662.49	0.2
nvs09b	2.436241	0.6	2.436241	0.5	2.436241	0.5	2.436241	0.6	2.436241	0.7	2.436241	0.5
nvs09	-16.97739	0.5	-16.97739	0.5	-16.97739	0.5	-16.97739	0.5	-16.97739	0.6	-16.97739	0.9
nvs10b	-7124.2	0.1	-7124.2	0.1	-7124.2	0.1	-7089.2	0.2	-7124.2	0.1	-6809.2	0.1
nvs10	-310.8	0.2	-310.8	0.1	-310.8	0.1	-310.8	0.5	-310.8	0.2	-310.8	0.1
nvs11b	-3568	0.4	-3568	0.2	-3568	0.1	-3568	0.2	-3568	0.3	–	0.1
nvs11	-431	0.2	-427.8	0.1	-431	0.1	-431	0.3	-427.8	0.5	-431	0.1
nvs12b	–	0.8	–	0.7	-4488.4	0.4	–	0.7	–	0.8	–	0.2
nvs12	-481.2	0.3	-448.4	0.1	-481.2	0.2	-481.2	0.4	-448.4	0.9	-481.2	0.1
nvs13b	-582.8	0.8	0	0.2	-582.8	0.6	-582.8	1.0	–	1.8	-13.2	0.1
nvs13	-166.4	1.0	-577.6	0.7	-166.4	0.4	-166.4	0.9	-166.4	1.6	-166.4	0.2
nvs14b	–	1.0	–	0.5	–	1.1	–	1.1	–	1.0	–	0.2
nvs14	-40358.15	0.1	-40358.15	0.5	-40358.15	0.1	-40358.15	0.2	-40358.15	0.1	-40358.15	0.3
nvs15b	-15	0.1	-15	0.1	-15	0.1	-15	0.1	-15	0.1	-15	0.1
nvs15	1	0.1	1	0.1	1	0.1	1	0.1	1	0.1	1	0.1
nvs16b	14.20312	0.2	14.20312	0.1	14.20312	0.1	14.20312	0.2	14.20312	0.2	14.20312	0.1
nvs16	0.703125	0.1	0.703125	0.1	0.703125	0.1	0.703125	0.1	0.703125	0.1	0.703125	0.1
nvs17b	–	5.5	0	0.3	–	1.3	–	6.8	–	5.2	-456.2	0.2
nvs17	-279	3.9	-1091	3.6	-279	1.7	-279	5.0	-279	5.3	-279	0.4
nvs18b	-697.6	1.5	-193.6	0.3	-697.6	1.2	-706	3.2	–	3.6	0	0.1
nvs18	-603	3.5	-688.8	3.1	-603	1.3	-769.2	3.6	-603	4.3	-603	0.3
nvs19b	-1427.6	5.8	-1382	5.8	-1406.8	2.9	-1427.6	6.4	–	7.7	0	0.4
nvs19	-282.4	4.4	-1089.4	5.1	-282.4	1.2	-1091.8	6.1	-282.4	5.8	-282.4	0.4
nvs20b	911805.2	0.5	911805.2	0.5	911805.2	0.4	911805.2	0.6	261.7265	0.7	271.1631	0.6

Table B.3 continued

	default		cuts		hierarchy		hessian		objective		simple	
	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]
nvs20	261.7265	0.4	261.7265	0.3	261.7265	0.3	261.7265	0.4	250.3245	0.3	251.2101	0.4
nvs21b	-2.634547	0.2	-2.634547	0.1	-2.634547	0.1	-2.634547	0.2	-2.634547	0.2	-2.634547	0.2
nvs21	-2.01e-05	0.4	-2.01e-05	0.2	-2.01e-05	0.5	-2.01e-05	0.4	-2.01e-05	0.5	-2.01e-05	0.3
nvs22b	–	1.5	–	0.9	–	1.6	–	1.0	–	1.3	–	1.2
nvs22	–	1.1	–	1.0	–	1.0	–	1.1	–	1.1	–	2.4
nvs23b	–	8.1	-1189.2	6.6	0	2.0	–	10.3	–	8.8	0	0.4
nvs23	–	5.4	-1111	5.6	0	1.5	–	8.5	–	6.2	–	0.5
nvs24b	–	8.4	–	12.1	–	3.0	–	11.2	–	12.0	0	0.2
nvs24	–	6.8	–	6.9	0	2.2	–	7.9	–	10.9	–	0.6
oaer	-1.923099	0.1	-1.923099	0.1	-1.923099	0.1	-1.923099	0.1	-1.923099	0.2	-1.923098	0.1
oil2	—	—	—	—	—	—	—	—	—	—	—	—
oil	–	213.8	–	277.5	–	229.2	–	3318.4	–	3297.4	–	47.2
ortez	-4865.809	0.3	-4865.809	0.4	-4847.359	0.4	-4865.809	0.5	-9534.538	0.7	–	0.6
parallel	2673.508	1.3	2673.508	1.5	2673.508	1.4	2673.508	1.5	372.2467	1.9	2673.508	1.8
procsel	-1.720972	0.1	-1.720972	0.1	-1.720972	0.1	-1.720972	0.1	-1.720972	0.1	0	0.1
product2	-2079.547	17.9	-2079.547	17.7	-2079.549	17.3	-2090.353	1578.9	-2092.767	1454.0	–	17.8
product	-1901.474	25.6	-1819.21	25.6	-1851.484	25.3	-1852.691	42.7	-1854.925	43.3	–	25.0
pumpb	159373.1	0.9	–	1.0	159373.1	0.9	0	1.0	–	1.5	–	0.8
pump	141260.7	0.9	–	1.1	139508.3	0.8	1872.139	0.9	–	1.3	151223	0.3
qap	406236	4458.4	409896	7200.0	412968	3102.4	–	7200.0	–	7200.0	—	—
qapw	396172	85.9	430274	50.6	396172	94.8	396172	90.0	396172	87.8	–	67.5
ravem	–	1.9	–	3.1	–	1.7	–	2.3	269590.1	1.9	–	1.2
risk2b	-55.73617	3.1	-55.73617	3.1	-29.71307	3.2	-54.89331	3.8	-55.73617	4.0	-55.73617	3.1
saa_2	–	4687.6	–	4802.7	–	5390.1	–	7200.0	–	7200.0	16462.46	2993.7
sep1	-470.1301	0.1	-470.1301	0.1	-470.1301	0.1	-470.1301	0.1	-470.1301	0.1	-524.5425	0.1
space25a	–	4.2	848.9524	2.9	–	4.1	–	7.0	–	7.2	638.8238	3.1
space25	638.8276	10.5	–	13.0	638.8276	10.1	638.8276	36.7	–	55.5	–	12.6
space960b	–	7200.0	40470000	7087.4	–	7200.0	–	7200.0	–	7200.0	–	7200.0
space960	–	3590.2	–	2120.8	–	3069.6	–	7200.0	–	7200.0	–	2007.6
spectra2	168.8805	0.8	168.8805	0.8	162.0836	0.6	162.0836	0.8	27.48579	0.7	306.3343	0.5

Table B.3 continued

	default		cuts		hierarchy		hessian		objective		simple	
	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]
springb	–	0.4	–	0.6	–	0.4	–	0.6	–	0.6	–	0.3
spring	–	0.5	–	0.5	–	0.4	–	0.5	1.602034	0.3	–	0.4
stockcycle	436419.1	2.0	357714.3	2.1	436419.1	2.0	436419.1	3.3	158543.2	3.5	–	3.3
synheat	413984.1	1.7	413984.1	1.6	415853.6	0.7	413984.1	1.6	232786.1	1.0	–	0.8
synthes1	7.906644	0.1	7.906644	0.1	7.906644	0.1	7.906644	0.1	6.008898	0.2	7.906644	0.1
synthes2	124.2945	0.1	124.2945	0.1	124.2945	0.1	84.29449	0.1	73.03531	0.1	124.2945	0.1
synthes3	127	0.2	127	0.1	127	0.1	127	0.2	81.91229	0.2	18.52498	0.1
tln12b	–	11.7	32.2	8.6	16.2	4.9	32.2	11.3	16.2	10.7	–	3.6
tln12	–	12.9	–	5.2	–	7.9	–	18.6	–	35.3	–	1.7
tln2b	2.2	0.1	2.2	0.1	2.2	0.1	2.2	0.1	2.2	0.1	–	0.1
tln2	5.3	0.2	5.3	0.1	5.3	0.1	5.3	0.1	5.3	0.1	5.3	0.2
tln4b	2.2	0.8	6.9	0.5	8.2	0.6	5.2	1.3	2.2	0.7	–	0.4
tln4	–	0.8	–	0.3	–	0.6	–	0.7	–	0.9	–	0.3
tln5b	8.2	3.2	8.2	3.2	16.5	1.1	8.2	4.1	8.2	3.5	–	0.4
tln5	–	1.3	–	0.8	–	0.8	–	1.6	–	1.4	–	0.3
tln6b	8.2	5.9	8.2	4.0	16.2	1.5	8.2	5.3	16.2	5.5	–	0.8
tln6	–	3.1	–	0.8	–	2.4	–	3.9	–	3.1	–	0.5
tln7b	8.2	7.7	8.2	5.8	8.7	2.3	8.2	6.9	4.2	7.4	–	0.9
tln7	–	4.7	–	0.8	–	2.4	–	7.2	–	6.3	–	0.6
tlossb	–	1.7	25.1	3.0	22.3	0.9	23.3	1.2	19.3	1.1	–	0.8
tloss	–	1.1	–	1.7	–	1.5	–	1.3	–	1.2	–	0.7
tls12b	–	57.3	–	57.4	–	62.2	–	193.7	–	206.4	–	31.3
tls12	–	55.8	–	31.3	–	31.3	–	387.4	–	293.1	–	22.9
tls2b	7.2	0.6	14.3	0.3	–	0.8	7.2	0.7	–	1.0	–	0.4
tls2	15.3	0.5	15.3	0.3	–	0.6	15.3	0.7	–	0.7	–	0.4
tls4b	–	2.5	–	1.8	–	2.8	–	2.9	–	3.8	–	1.2
tls4	–	1.9	–	1.1	19.6	1.3	19.6	1.8	19.6	2.3	–	0.8
tls5b	–	10.6	–	2.3	–	4.4	–	23.4	–	20.3	–	2.1
tls5	–	4.6	27.5	1.8	–	2.6	–	20.9	–	20.0	–	1.9
tls6b	–	14.0	–	4.8	–	8.9	–	38.0	–	37.0	–	3.5

Table B.3 continued

	default		cuts		hierarchy		hessian		objective		simple	
	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]	solution	t[s]
tls6	–	10.1	–	3.5	20.2	6.1	–	39.3	–	32.4	–	2.7
tls7b	–	20.3	–	8.3	–	10.8	–	62.3	–	54.8	–	5.9
tls7	–	17.4	–	7.1	–	9.1	–	63.5	–	61.6	–	5.7
tltrb	–	1.1	129.4125	0.6	–	1.0	–	1.2	–	1.4	–	0.8
tltr	48.27083	0.6	129.4125	0.2	48.27083	0.6	48.27083	0.5	48.27083	0.6	–	0.5
util	1054.043	0.4	1054.043	0.4	1054.043	0.4	1054.043	0.4	999.7546	0.5	1054.043	0.3
var_con10	0	40.0	0	37.3	0	36.7	0	162.8	0	163.1	–	39.9
var_con5	0	37.3	285.8743	36.4	0	38.9	0	172.0	0	172.1	–	38.9
waste	147740.5	23.9	147740.5	24.1	147740.5	23.8	147740.5	136.1	147740.5	140.6	148063.8	23.5
water4	816162.3	1.6	–	1.7	935488.8	1.3	553793.9	1.6	–	4.5	–	2.1
waterx	940735.9	2.7	940735.9	2.8	1083.044	3.1	940735.9	2.9	940735.9	2.8	1664.264	1.6
waterz	–	2.9	–	2.1	–	2.5	–	3.2	–	4.2	–	1.7
windfacb	–	1.2	–	0.4	–	1.1	–	1.2	–	0.7	–	0.3
windfac	0.7499969	0.1	0.7499969	0.2	0.7499969	0.2	0.7499969	0.2	0.453673	0.2	0.7499969	0.2

Table B.4.: Impact of LNS heuristic on root node performance for MIQCP instances

	no LNS		aux. MIP		sub-MINLP	
Instance	Primal Bound	Time	Primal Bound	Time	Primal Bound	Time
108bar	–	2.6	–	2.8	–	3.0
10bar2	–	0.1	–	0.2	–	0.1
200bar	–	4.5	–	7.0	–	25.7
25bar	–	0.2	–	0.4	–	0.3
classical_200_0	-0.01	2.2	-0.01	4.7	-0.05	6.9
classical_200_1	–	2.5	–	3.2	–	2.5
classical_20_0	–	0.1	–	0.1	–	0.1
classical_20_1	–	0.1	–	0.1	–	0.1
classical_50_0	–	0.4	-0.07	1.0	–	0.6
classical_50_1	–	0.4	–	0.3	-0.09	1.6
clay0203m	–	0.1	–	0.1	–	0.1
clay0205m	–	0.1	–	0.1	–	0.1
clay0303m	–	0.1	–	0.1	31302.87	0.1
clay0305m	–	0.1	–	0.1	–	0.1
du-opt5	195.55	0.1	195.55	0.1	195.55	0.1
du-opt	1314.42	0.1	1314.42	0.1	1314.42	0.1
ex1263	35.30	0.1	32.30	0.3	21.30	0.4
ex1264	24.60	0.1	24.60	0.1	9.30	0.2
ex1265	20.30	0.1	18.30	0.1	11.60	0.2
ex1266	27.30	0.1	22.30	0.2	22.30	0.4
fac3	38335620.70	0.2	38335620.70	0.2	38335620.70	0.2
feedtray2	–	0.1	–	0.1	–	0.1
iair04	–	55.0	66475.31	100.5	65707.73	126.4
iair05	–	45.3	28249.89	52.5	27665.26	57.2
ibc1	5.17	6.0	3.54	20.1	3.54	23.6
ibell3a	897678.56	0.1	878785.03	0.1	879009.26	0.1
ibienst1	53.03	1.7	53.03	2.0	53.03	1.9
icap6000	-2446601.00	2.5	-2447103.00	3.2	-2448165.00	7.6
icvxqp1	4604249.00	26.9	1674226.00	28.1	3811872.00	30.8
ieilD76	1008.64	22.2	892.69	39.9	888.69	42.1
ilaser0	–	1.0	–	1.0	–	1.0
imas284	102638.21	0.4	93699.30	1.4	92194.04	1.7
imisc07	–	0.7	–	0.6	–	0.6
imod011	0.00	19.2	0.00	19.7	0.00	18.8
inug08	7213.00	10.5	7213.00	10.6	7213.00	10.3
iportfolio	0.00	53.8	0.00	71.5	0.00	70.9
iqap10	492.62	116.5	492.62	134.4	403.06	251.2
iqiu	864.10	1.7	864.10	1.5	-29.49	2.8
iran13x13	3956.10	1.5	3386.36	3.2	3373.67	3.7
iran8x32	5849.29	1.1	5483.62	3.0	5442.58	3.4
isqp0	–	0.1	–	0.2	–	0.2
isqp1	–	0.8	–	0.7	–	0.7
isqp	–	0.1	–	0.1	–	0.1
itointqor	0.00	0.1	0.00	0.1	0.00	0.1
ivalues	0.00	0.7	0.00	1.3	0.00	0.8
meanvarx	16.06	0.1	16.06	0.1	14.52	0.1
netmod_dol1	-0.00	2.2	-0.14	8.3	-0.49	12.5
netmod_dol2	-0.00	4.4	-0.20	8.0	-0.51	12.7
netmod_kar1	-0.00	0.2	-0.00	0.3	-0.14	0.6
netmod_kar2	-0.00	0.4	-0.00	0.4	-0.14	0.6

Table B.4 continued

	no LNS		aux. MIP		sub-MINLP	
Instance	Primal Bound	Time	Primal Bound	Time	Primal Bound	Time
nous1	–	0.1	–	0.1	–	0.1
nous2	–	0.1	–	0.1	–	0.1
nuclear14a	–	2.4	–	3.7	–	4.2
nuclear14b	–	2.9	–	3.0	–	68.6
nuclear14	–	0.1	–	0.2	–	0.1
nuclearva	–	0.1	–	0.1	–	0.1
nvs19	0.00	0.1	-208.20	0.1	-208.20	0.1
nvs23	0.00	0.1	0.00	0.1	0.00	0.1
product2	–	0.3	–	0.4	–	0.5
product	–	0.8	–	1.1	–	1.2
robust_100_0	–	1.3	–	1.5	-0.09	8.6
robust_100_1	–	1.1	–	1.4	–	2.3
robust_200_0	–	3.2	–	4.3	-0.14	26.2
robust_20_0	–	0.1	–	0.1	-0.08	0.5
robust_50_0	–	0.6	–	0.4	-0.09	1.1
robust_50_1	–	0.7	–	0.6	-0.09	7.3
sep1	–	0.1	-510.08	0.1	-510.08	0.1
shortfall_100_0	-1.00	1.3	-1.07	2.7	-1.07	2.8
shortfall_100_1	-1.00	1.2	-1.09	3.5	-1.06	3.0
shortfall_200_0	-1.00	3.2	-1.06	7.2	-1.06	10.0
shortfall_20_0	-1.00	0.1	-1.09	0.1	-1.09	0.1
shortfall_50_0	-1.00	0.6	-1.08	1.9	-1.08	2.4
shortfall_50_1	-1.00	0.3	-1.08	1.1	-1.09	2.2
SLay05H	110249.88	0.4	39218.64	0.4	22664.68	1.4
SLay05M	110272.14	0.1	46038.93	0.1	25589.68	0.2
SLay07M	265752.24	0.1	108756.71	0.2	65668.81	1.7
SLay10H	577942.50	3.7	577942.50	3.1	382507.50	10.6
SLay10M	705952.50	0.2	387120.35	0.4	156495.23	2.9
space25a	–	0.3	–	0.3	–	0.3
space25	–	0.5	–	0.5	–	0.5
spectra2	306.33	1.2	306.33	1.1	31.98	1.0
tln12	–	0.2	–	0.2	–	0.2
tln5	24.00	0.1	15.10	0.1	12.50	0.2
tln6	24.70	0.1	16.30	0.2	16.30	0.2
tln7	26.70	0.1	20.40	0.2	16.40	0.3
tloss	–	0.1	–	0.1	–	0.1
tltr	48.07	0.1	48.07	0.2	48.07	0.2
uflquad-15-60	1440.87	4.2	1440.87	3.8	1440.87	3.8
uflquad-20-50	1254.25	22.0	1254.25	22.1	1254.25	22.3
uflquad-30-100	2952.74	6.9	1734.79	40.9	1734.79	50.4
uflquad-40-80	2111.07	1.8	996.31	159.5	973.16	538.3
util	–	0.1	–	0.1	1000.70	0.1
waste	1056.36	0.2	688.24	0.4	688.24	0.7

Table B.5.: Impact of LNS heuristic on overall solving process for MIQCP instances

	nolns		lp		cip	
Instance	Nodes	Time	Nodes	Time	Nodes	Time
108bar	85 133	timeout	64 110	timeout	64 110	timeout
10bar2	1 778	2.4	908	2.8	904	2.8
25bar	33 520	102.0	17 561	59.8	17 517	59.7
200bar	10 728	timeout	941	timeout	941	timeout
classical_200_0	84 895	timeout	83 922	timeout	84 780	timeout
classical_200_1	102 737	timeout	99 478	timeout	99 904	timeout
classical_20_0	74	0.4	71	1.0	72	0.8
classical_20_1	452	1.0	414	0.9	416	1.1
classical_50_0	119 275	357.9	119 917	391.5	118 453	369.6
classical_50_1	24 991	66.8	24 991	77.1	14 857	44.4
clay0203m	36	0.1	36	0.1	36	0.1
clay0205m	9 205	2.9	12 500	6.3	9 422	3.5
clay0303m	67	0.1	67	0.1	71	0.2
clay0305m	9 528	3.2	9 109	6.0	8 178	3.5
du-opt5	2 196 266	timeout	2 180 074	timeout	2 180 164	timeout
du-opt	2 039 144	timeout	2 034 078	timeout	2 017 499	timeout
ex1263	716	0.7	405	0.6	415	0.8
ex1264	111	0.1	111	0.2	122	0.2
ex1265	112	0.2	390	0.5	101	0.3
ex1266	84	0.3	219	0.4	219	0.8
fac3	121 419	timeout	120 915	timeout	120 748	timeout
feedtray2	26	0.4	26	0.3	26	0.3
iair04	197	127.6	23	163.8	31	195.3
iair05	87	106.2	49	98.2	50	108.1
ibc1	239	22.6	168	42.8	168	45.4
ibell3a	41 020	14.1	1	0.1	1	0.1
ibienst1	24 391	71.4	24 377	74.9	24 432	75.6
icap6000	3 920	8.1	2 613	6.8	1 437	8.6
icvxqp1	7 619	timeout	7 383	timeout	8 839	timeout
ieilD76	6	44.2	4	42.0	3	43.2
ilaser0	107 300	timeout	107 948	timeout	107 629	timeout
imas284	16 908	9.8	16 358	16.9	16 024	13.8
imisc07	33 979	29.8	33 179	38.7	35 006	41.2
imod011	1	18.8	1	19.6	1	18.8
inug08	1	10.5	1	10.5	1	10.5
iportfolio	59 021	timeout	52 504	timeout	51 071	timeout
iqap10	11	227.7	7	221.3	4	339.1
iqiu	12 600	73.5	13 258	80.9	11 812	84.5
iran13x13	43 494	45.8	34 019	47.6	33 938	45.3
iran8x32	13 064	21.4	12 567	26.6	11 082	22.2
isqp0	68 763	timeout	68 520	timeout	68 740	timeout
isqp1	26 228	timeout	26 163	timeout	26 354	timeout
isqp	58 315	timeout	58 439	timeout	58 531	timeout
itointqor	–	–	–	–	–	–
ivalues	138 886	timeout	141 110	timeout	140 415	timeout
meanvarx	118	0.2	118	0.3	144	0.4
netmod_dol1	41 422	timeout	38 181	timeout	40 258	timeout
netmod_dol2	139	39.7	138	44.8	75	46.8
netmod_kar1	221	3.3	221	3.5	274	3.9
netmod_kar2	221	3.5	221	3.4	274	3.8

Table B.5 continued

	noIns		lp		cip	
Instance	Nodes	Time	Nodes	Time	Nodes	Time
nous1	2 480 666	timeout	2 282 507	timeout	2 179 770	timeout
nous2	3 507	2.5	2 401	2.1	2 401	2.0
nuclear14a	9 096	timeout	22 728	timeout	20 588	timeout
nuclear14b	21 176	timeout	21 029	timeout	14 253	timeout
nuclear14	1 410 156	timeout	1 419 852	timeout	1 402 513	timeout
nuclearva	4 152 523	timeout	4 151 128	timeout	4 111 046	timeout
nvs19	9 251	6.5	9 751	7.7	9 742	8.0
nvs23	165 393	139.6	156 175	136.1	156 175	136.1
product2	2 326 759	timeout	2 333 174	timeout	2 339 046	timeout
product	135 445	205.2	104 486	189.1	104 486	189.8
robust_100_0	44 014	607.9	44 014	664.1	38 983	607.1
robust_100_1	22 126	306.6	26 568	423.0	24 232	396.1
robust_200_0	60 145	timeout	56 147	timeout	60 670	timeout
robust_20_0	18	0.2	16	0.4	8	0.6
robust_50_0	61	1.0	59	1.1	12	1.4
robust_50_1	65	1.0	65	4.2	62	9.0
sep1	34	0.1	19	0.1	19	0.1
shortfall_100_0	345 564	timeout	298 497	timeout	310 145	timeout
shortfall_100_1	100 790	898.7	127 195	1256.2	110 472	1075.8
shortfall_200_0	67 319	timeout	67 601	timeout	68 137	timeout
shortfall_20_0	50	0.3	32	0.5	32	0.6
shortfall_50_0	133 538	506.9	116 971	526.3	83 573	340.0
shortfall_50_1	4 435	15.7	2 305	11.4	3 883	19.1
SLay05H	363	2.0	384	7.4	61	1.7
SLay05M	378	1.1	195	0.9	117	0.9
SLay07M	946	3.5	1 007	9.0	653	6.8
SLay10H	224 840	timeout	217 178	timeout	207 060	timeout
SLay10M	154 448	1013.0	127 022	859.7	41 750	344.1
space25a	184 707	timeout	53 678	timeout	53 678	timeout
space25	27 357	timeout	21 694	timeout	21 694	timeout
spectra2	10 525	timeout	9 633	timeout	13 073	timeout
tln12	967 545	timeout	851 244	timeout	835 407	timeout
tln5	41 913	26.9	27 572	19.8	21 763	16.3
tln6	5 741 445	timeout	5 508 841	timeout	5 914 657	timeout
tln7	3 016 385	timeout	3 066 726	timeout	2 761 957	timeout
tloss	68	0.1	46	0.1	34	0.1
tltr	3	0.1	3	0.2	3	0.2
uflquad-15-60	793	1457.5	793	1458.2	793	1463.2
uflquad-20-50	101	timeout	101	timeout	101	timeout
uflquad-30-100	77	timeout	43	timeout	43	timeout
uflquad-40-80	258	timeout	68	timeout	69	timeout
util	465	0.1	465	0.2	412	0.2
waste	1 756 578	timeout	1 608 003	timeout	1 740 261	timeout

Table B.6.: RENS: computing optimal roundings for MMM instances, after cuts

	% Vars Fixed			RENS			
Instance	Int	All	TimeS	Time	Nodes	Solution	Found At
10teams	86.9	92.6	0.8	0.0	0	–	–
30n20b8	97.3	98.1	42.0	0.0	0	–	–
a1c1s1	18.8	7.9	6.6	limit	404552	13209.1836	271570
acc-tight5	58.8	78.1	6.0	0.7	0	–	–
aflow30a	78.9	80.4	4.4	3.6	3777	1158*	357
aflow40b	91.8	92.6	13.1	43.1	67215	1179	19497
air04	96.0	99.6	7.0	0.0	0	–	–
air05	96.1	98.9	2.4	0.0	0	–	–
app1-2	96.1	48.5	52.8	115.7	598	–	–
arki001	85.7	68.5	1.5	0.2	1	–	–
ash608gpia-3col	28.0	53.5	22.1	0.0	0	–	–
atlanta-ip	88.6	97.0	59.8	2.6	27	98.0096	22
bab5	97.2	99.4	56.3	0.1	0	–	–
beasleyC3	63.7	73.4	3.5	1.5	779	789	428
bell3a	96.2	90.2	0.0	0.0	1	878430.316*	1
bell5	72.3	77.5	0.1	0.0	0	–	–
biella1	90.5	92.0	5.6	limit	1439186	3278480.58	904043
bienst2	0.0	0.0	1.1	1634.4	459071	54.6*	49778
binkar10_1	48.8	48.7	0.9	270.9	407041	6746.64	89429
blend2	90.6	90.8	0.3	0.1	35	7.599*	22
bley_xl1	27.5	64.2	226.5	3.9	18	190*	18
bnatt350	50.4	66.3	4.2	0.0	0	–	–
cap6000	99.9	100.0	1.8	0.0	1	-2443599	1
core2536-691	94.6	94.8	11.5	3289.1	544659	695	10446
cov1075	25.0	25.0	0.9	35.0	10410	20*	506
csched010	88.7	84.3	2.9	1.0	38	–	–
dano3mip	67.4	64.6	30.6	limit	14384	762.75	2737
danoint	5.4	0.6	1.2	450.3	109479	65.6667*	5463
dcmulti	29.7	21.9	0.7	0.4	180	188186.5	68
dfn-gwin-UUM	38.9	13.4	0.5	819.3	307149	39920	4343
disctom	97.5	99.5	2.1	0.0	0	–	–
ds	99.0	99.4	105.5	0.6	0	–	–
dsbmip	84.5	21.5	0.8	0.2	34	-305.1982*	34
egout	85.7	85.7	0.0	0.0	1	568.1007*	1
eil33-2	98.5	99.7	6.3	0.0	0	–	–
eilB101	88.9	99.0	13.5	0.1	0	–	–
enigma	83.0	92.0	0.0	0.0	0	–	–
enlight13	66.6	96.2	0.2	0.0	0	–	–
enlight14	68.4	95.9	0.3	0.0	0	–	–
fast0507	99.5	99.5	14.3	14.4	10302	177	4218
fiber	91.9	95.0	0.9	0.1	78	411151.82	48
fixnet6	88.6	82.3	1.1	0.4	32	3997	26
flugpl	11.1	35.7	0.0	0.0	0	–	–
gesa2	88.0	82.4	1.1	0.0	5	25780031.4*	3
gesa2-o	92.9	88.9	1.0	0.0	5	25780031.4*	3
gesa3	78.6	82.0	1.3	0.0	36	27991430.1	33
gesa3_o	85.0	85.6	1.2	0.0	19	27991430.1	17
glass4	70.8	74.4	0.3	1.6	2622	2.2666856e+09	2491
gmu-35-40	93.5	93.7	0.5	0.1	61	-2399398.21	57
gt2	90.8	100.0	0.0	0.0	1	21166*	1
harp2	91.1	98.3	0.8	0.0	0	–	–
iis-100-0-cov	0.0	0.0	2.6	1700.5	120842	29*	30
iis-bupa-cov	57.8	57.8	8.8	3819.8	537082	36*	1634
iis-pima-cov	82.3	82.3	17.9	54.6	12823	33*	4545
khb05250	66.7	32.6	0.3	0.1	7	106940226*	4
l152lav	97.2	99.4	0.1	0.0	0	–	–

Table B.6 continued

	% Vars Fixed			RENS			
Instance	Int	All	TimeS	TimeR	NodesR	Solution	Found At
lectsched-4-obj	28.7	31.0	6.8	0.0	0	–	–
liu	49.0	46.2	10.8	limit	6040599	3418	4613091
lseu	74.4	77.9	0.1	0.1	22	1148	18
m100n500k4r1	73.2	73.2	0.4	0.8	848	-22	180
macrophage	43.5	45.8	2.3	0.0	0	–	–
map18	63.6	76.6	48.8	128.7	2896	-847*	711
map20	63.6	75.7	38.9	104.5	2408	-922*	888
markshare1	76.0	76.0	0.0	0.0	107	142	59
markshare2	78.3	78.3	0.1	0.0	101	131	94
mas74	91.3	90.7	0.2	0.4	90	14343.468	67
mas76	91.9	91.3	0.2	0.3	42	40560.0541	35
mcsched	15.9	18.4	3.0	limit	1721772	213768	54512
mik-250-1-100-1	62.4	62.2	0.2	0.2	172	-66729*	172
mine-90-10	20.6	27.8	4.2	limit	2667271	-784302338*	2445697
misc03	78.3	99.3	0.2	0.0	0	–	–
misc06	90.2	38.4	0.2	0.1	19	12850.8607*	17
misc07	82.8	94.0	0.3	0.0	0	–	–
mitre	99.6	100.0	4.8	0.0	1	115155*	1
mkc	92.6	93.5	2.9	0.3	389	-539.866	160
mod008	94.4	94.4	0.7	0.1	19	309	4
mod010	98.6	100.0	0.3	0.0	0	–	–
mod011	53.1	12.6	7.2	64.1	387	-54219145.9	129
modglob	60.2	56.8	0.2	1.3	5795	20799458.8	4360
momentum1	76.7	73.0	11.8	0.2	0	–	–
momentum2	74.8	76.5	50.9	0.7	0	–	–
momentum3	78.4	77.1	1034.8	0.5	0	–	–
msc98-ip	82.0	85.5	145.8	0.1	0	–	–
mspp16	99.0	99.1	1202.2	13.1	0	–	–
mzzv11	83.4	82.9	74.8	0.0	0	–	–
mzzv42z	86.5	86.1	75.4	0.1	0	–	–
n3div36	99.9	99.9	6.7	0.1	1	151600	1
n3seq24	99.6	99.7	82.8	63.1	24054	68000	3536
n4-3	56.9	10.0	2.8	limit	415575	9010	112840
neos-1109824	94.5	97.0	3.2	0.0	0	–	–
neos-1337307	45.1	45.2	4.7	limit	767115	-202133	4868
neos-1396125	45.0	48.0	3.5	11.4	2026	3000.0553*	1867
neos-1601936	80.8	77.1	7.9	limit	252812	–	–
neos-476283	99.0	93.0	147.4	3.0	130	406.8123	71
neos-686190	96.0	98.3	1.3	0.0	0	–	–
neos-849702	70.8	80.0	1.6	0.1	0	–	–
neos-916792	87.1	89.3	13.1	0.1	0	–	–
neos-934278	76.8	75.1	49.9	limit	105271	1332	9576
neos13	78.6	78.1	26.8	limit	75103	-65.6552	51090
neos18	70.8	78.1	0.9	0.0	0	–	–
net12	41.8	56.3	31.7	0.1	0	–	–
netdiversion	96.1	99.9	301.9	1.1	0	–	–
newdano	0.0	0.0	2.9	limit	1160686	66.5	774340
noswot	47.4	64.2	0.1	0.0	0	–	–
ns1208400	78.2	82.5	6.2	0.1	0	–	–
ns1688347	99.4	99.9	20.1	0.0	0	–	–
ns1758913	91.4	92.1	5385.0	6.4	5	-457.7183	5
ns1766074	77.8	87.0	0.1	0.0	1	–	–
ns1830653	57.8	72.8	4.6	0.1	0	–	–
nsrand-ipx	98.3	98.4	19.7	569.0	2061551	55360	31084
nw04	100.0	100.0	14.1	0.4	0	–	–
opm2-z7-s2	9.9	10.1	10.9	limit	52398	-10271	50719
opt1217	95.2	96.9	0.3	0.0	1	-16*	1
p0201	63.1	92.8	0.4	0.0	1	7805	1
p0282	71.5	72.5	0.3	0.1	1	258411*	1

Table B.6 continued

	% Vars Fixed			RENS			
Instance	Int	All	TimeS	TimeR	NodesR	Solution	Found At
p0548	96.6	100.0	0.2	0.0	1	8763	1
p2756	98.9	99.6	1.1	0.0	1	3152	1
pg5_34	97.0	46.0	3.8	1.5	7	-14287.7021	4
pigeon-10	44.6	77.2	1.3	7.7	27538	–	–
pk1	72.7	46.5	0.0	0.2	460	29	376
pp08a	46.9	33.8	0.3	0.6	319	7360	148
pp08aCUTS	48.4	32.1	0.2	0.6	434	7370	405
protfold	65.8	88.4	3.8	0.2	0	–	–
pw-myciel4	58.4	60.4	7.6	0.0	0	–	–
qiu	25.0	25.0	0.2	47.6	23791	-132.8731*	1149
qnet1	92.0	95.1	0.9	0.0	1	21237.6552	1
qnet1_o	91.7	95.0	1.1	0.1	261	22600.83	168
rail507	99.5	99.5	14.8	41.2	23871	178	230
ran16x16	71.1	71.5	1.1	138.7	464014	3846	4332
rd-rplusc-21	55.5	66.0	57.7	4.3	415	–	–
reblock67	17.6	26.6	3.7	limit	5552244	-34629815.5	700261
rentacar	75.0	7.6	1.2	0.6	9	30356761*	6
rgn	96.0	54.9	0.2	0.0	1	82.2*	1
rmatr100-p10	49.0	49.2	2.8	16.5	686	424	322
rmatr100-p5	63.0	63.6	4.1	13.0	258	976*	118
rmine6	65.5	67.0	8.2	5266.1	4687190	-457.1727	811719
rocII-4-11	81.6	88.4	18.4	0.0	0	–	–
rococoC10-001000	82.1	85.8	2.7	33.5	42970	12067	4679
roll3000	65.2	78.3	2.7	0.4	94	14193	12
rout	83.2	93.7	0.6	0.0	0	–	–
satellites1-25	89.2	99.4	68.2	0.0	0	–	–
set1ch	96.2	90.2	0.7	0.0	3	54537.75*	2
seymour	52.7	55.5	15.1	limit	1067621	427	917345
sp98ic	99.3	99.3	4.8	12.2	37885	469766019	12687
sp98ir	93.9	96.0	3.2	0.0	0	–	–
stein27	11.1	11.1	0.0	0.3	1202	18*	50
stein45	17.8	17.8	0.2	0.9	3597	30*	313
swath	99.2	98.3	3.4	0.0	0	–	–
t1717	99.2	99.4	27.3	0.4	0	–	–
tanglegram1	99.1	99.1	14.4	0.2	0	–	–
tanglegram2	96.4	96.7	1.3	0.0	0	–	–
timtab1	14.4	15.9	0.8	8.3	16082	827609	4701
timtab2	12.6	14.7	1.7	2.4	151	–	–
tr12-30	73.6	50.7	1.5	408.3	909211	131438	17370
triptim1	87.0	99.5	127.3	0.2	0	–	–
unitcal_7	81.6	59.7	63.9	2.8	1	–	–
vpm2	55.4	50.8	0.4	0.4	336	13.75*	301
vpphard	97.6	98.1	28.4	0.6	0	–	–
zib54-UUE	17.5	21.5	2.6	limit	1126102	10334015.8*	392023

Table B.7.: RENS: computing optimal roundings for MMM instances, before cuts

	% Vars Fixed			RENS			
Instance	Int	All	TimeS	Time	Nodes	Solution	Found At
10teams	90.1	92.5	0.4	0.0	0	–	–
30n20b8	98.1	98.8	2.2	0.0	0	–	–
a1c1s1	15.6	10.4	3.6	limit	2174731	–	–
acc-tight5	56.8	84.4	2.0	0.1	0	–	–
aflow30a	92.6	97.9	0.2	0.0	0	–	–
aflow40b	97.2	98.0	1.0	0.0	0	–	–
air04	96.1	98.2	6.2	0.0	0	–	–
air05	96.4	98.7	1.8	0.0	0	–	–
app1-2	96.7	48.9	14.9	60.3	492	-23	492
arki001	84.9	72.6	0.5	0.0	0	–	–
ash608gpia-3col	33.4	67.3	3.3	0.0	0	–	–
atlanta-ip	88.9	97.3	30.2	1.8	196	99.0098	195
bab5	98.8	99.1	29.0	0.1	0	–	–
beasleyC3	82.4	100.0	0.1	0.0	1	945	1
bell3a	84.6	80.4	0.0	0.0	13	878651.068	12
bell5	70.2	87.5	0.0	0.0	1	9082700.02	1
biella1	90.5	92.0	5.2	limit	1251065	3253217.92	682395
bienst2	0.0	0.0	0.3	1133.6	514667	54.6*	248177
binkar10_1	77.6	77.6	0.1	1.1	2688	6796.71	1565
blend2	97.4	99.3	0.0	0.0	0	–	–
bley_xl1	47.4	82.2	171.8	0.4	11	210	11
bnatt350	58.9	59.4	1.3	0.0	0	–	–
cap6000	100.0	100.0	0.6	0.0	1	-2442801	1
core2536-691	94.6	94.8	11.2	5427.0	951274	695	30373
cov1075	0.0	0.0	0.6	limit	1622177	20*	184
csched010	94.2	91.6	0.3	0.0	1	–	–
dano3mip	77.5	74.4	22.1	limit	25874	761.9286	118
danoint	7.1	0.8	0.6	152.0	50018	65.6667*	40513
dcmulti	35.1	41.4	0.1	9.4	40800	188182*	12687
dfn-gwin-UUM	50.0	4.8	0.1	54.1	108721	41040	20493
disctom	97.5	99.5	1.8	0.0	0	–	–
ds	99.2	99.5	27.0	0.5	0	–	–
dsbmip	74.1	20.6	0.5	0.2	7	–	–
egout	71.4	100.0	0.0	0.0	1	625.3192	1
eil33-2	99.3	99.9	3.6	0.0	0	–	–
eilB101	96.8	97.8	1.6	0.0	0	–	–
enigma	88.0	99.0	0.0	0.0	0	–	–
enlight13	99.1	99.1	0.0	0.0	0	–	–
enlight14	99.2	99.2	0.0	0.0	0	–	–
fast0507	99.5	99.5	13.2	15.6	12207	177	5197
fiber	96.2	100.0	0.0	0.0	0	–	–
fixnet6	96.8	90.6	0.0	0.0	3	4435	3
flugpl	11.1	21.4	0.0	0.0	0	–	–
gesa2	89.7	91.6	0.2	0.0	5	26038337.6	5
gesa2-o	89.9	96.0	0.2	0.0	6	26038337.6	5
gesa3	81.5	88.1	0.2	0.0	29	27991430.1	24
gesa3_o	84.6	90.6	0.2	0.0	29	27991430.1	24
glass4	75.8	83.9	0.1	0.1	49	–	–
gmu-35-40	98.3	98.6	0.3	0.0	0	–	–
gt2	91.3	96.5	0.0	0.0	0	–	–
harp2	97.8	99.3	0.1	0.0	0	–	–
iis-100-0-cov	0.0	0.0	0.6	2902.6	186105	29*	47
iis-bupa-cov	55.1	55.1	1.3	6150.5	745491	36*	1989
iis-pima-cov	82.1	82.1	1.8	50.7	11558	33*	1363
khb05250	20.8	4.3	0.0	1.6	3364	106940226*	87
l152lav	97.2	99.9	0.2	0.0	0	–	–

Table B.7 continued

Instance	% Vars Fixed Int	All	TimeS	RENS TimeR	NodesR	Solution	Found At
lectsched-4-obj	78.2	78.8	1.5	0.0	0	–	–
liu	51.4	48.4	36.3	limit	8755631	4762	2865
lseu	90.7	100.0	0.0	0.0	0	–	–
m100n500k4r1	80.0	80.0	-0.0	0.5	650	-21	102
macrophage	70.0	70.5	0.1	0.0	0	–	–
map18	58.5	71.5	33.3	3299.9	61952	-847*	52
map20	66.1	80.4	27.0	404.2	17845	-922*	617
markshare1	88.0	92.0	0.0	0.0	1	204	1
markshare2	88.3	88.3	0.0	0.0	3	131	2
mas74	91.9	91.3	0.0	0.0	58	14372.8713	20
mas76	92.6	92.0	0.0	0.0	21	40560.0541	12
mcsched	15.8	18.2	0.9	limit	1966420	214792	1088285
mik-250-1-100-1	60.0	59.8	0.1	0.0	32	0	31
mine-90-10	20.6	27.8	3.8	limit	2556662	-782117611	1315502
misc03	87.0	97.1	0.1	0.0	0	–	–
misc06	92.9	39.0	0.1	0.1	43	12854.0023	33
misc07	91.4	98.3	0.2	0.0	0	–	–
mitre	99.6	100.0	4.5	0.0	1	116745	1
mkc	97.6	99.1	1.1	0.0	1	-284.55	1
mod008	98.4	100.0	0.0	0.0	1	308	1
mod010	98.4	99.8	0.4	0.0	0	–	–
mod011	83.3	21.2	0.6	1.6	153	-53656254.1	50
modglob	69.4	76.6	0.1	0.0	174	20784597.9	174
momentum1	80.3	78.6	5.1	155.3	76443	109169.397	19330
momentum2	78.9	83.1	26.2	0.3	0	–	–
momentum3	77.3	78.4	497.4	0.6	0	–	–
msc98-ip	86.8	89.4	6.6	0.1	0	–	–
mspp16	99.9	100.0	1001.5	13.0	0	–	–
mzzv11	86.4	85.7	51.5	0.0	0	–	–
mzzv42z	88.2	87.8	51.9	0.0	0	–	–
n3div36	99.9	99.9	2.0	0.1	3	149800	2
n3seq24	99.8	99.9	23.0	1.5	0	–	–
n4-3	74.1	31.8	0.1	359.6	215073	9395	12131
neos-1109824	96.8	99.9	1.1	0.0	0	–	–
neos-1337307	50.0	50.1	2.4	742.6	154344	-202143	12623
neos-1396125	47.3	53.1	1.1	2.2	510	3000.0556*	489
neos-1601936	83.7	77.2	6.5	0.0	0	–	–
neos-476283	99.0	93.0	141.6	2.7	121	406.8123	74
neos-686190	96.9	99.0	0.2	0.0	0	–	–
neos-849702	74.5	80.9	1.2	0.0	0	–	–
neos-916792	87.1	89.3	1.1	0.1	0	–	–
neos-934278	79.5	78.0	18.6	limit	215616	346	201165
neos13	78.2	77.7	12.2	39.6	267	-66.8793	267
neos18	71.4	71.7	0.3	0.0	0	–	–
net12	60.4	79.8	7.9	0.1	0	–	–
netdiversion	96.5	100.0	199.6	1.0	0	–	–
newdano	3.6	0.4	0.4	1900.2	1332691	66.8333	800380
noswot	50.5	47.5	0.0	0.0	0	–	–
ns1208400	84.1	87.6	2.3	0.0	0	–	–
ns1688347	65.8	77.8	8.0	0.1	0	–	–
ns1758913	97.4	98.7	1437.2	1.3	41	-862.2649	37
ns1766074	77.8	86.0	0.0	0.0	7	–	–
ns1830653	57.8	50.3	1.2	0.0	0	–	–
nsrand-ipx	99.0	99.2	1.1	0.2	1381	61760	109
nw04	100.0	100.0	10.4	0.4	0	–	–
opm2-z7-s2	9.9	10.1	7.6	limit	52408	-10271	50719
opt1217	96.2	98.8	0.1	0.0	13	-16*	13
p0201	78.5	98.5	0.1	0.0	0	–	–
p0282	96.0	100.0	0.0	0.0	1	320465	1

Table B.7 continued

Instance	% Vars Fixed Int	All	TimeS	RENS TimeR	NodesR	Solution	Found At
p0548	91.7	97.8	0.1	0.0	0	–	–
p2756	95.6	98.8	0.3	0.0	0	–	–
pg5_34	12.0	0.5	28.3	limit	5836732	-14232.4589	1862706
pigeon-10	69.5	99.3	0.1	0.0	0	–	–
pk1	72.7	46.5	0.0	0.1	402	29	247
pp08a	17.2	9.4	125.6	limit	33411470	7360	395800
pp08aCUTS	28.1	16.5	0.1	134.2	557900	7350*	29979
protfold	72.2	90.0	2.0	0.1	0	–	–
pw-myciel4	46.0	56.9	1.0	0.0	0	–	–
qiu	25.0	25.0	0.2	47.5	23791	-132.8731*	1149
qnet1	96.3	99.3	0.2	0.0	1	21396.52	1
qnet1_o	99.2	100.0	0.1	0.0	1	28462.14	1
rail507	99.5	99.5	13.7	100.5	66744	178	341
ran16x16	92.2	100.0	0.0	0.0	1	4333	1
rd-rplusc-21	77.9	78.8	46.0	0.2	0	–	–
reblock67	17.6	26.6	3.1	limit	6367150	-34629815.5	540746
rentacar	70.8	8.4	0.8	0.6	15	30356761*	13
rgn	85.0	48.6	0.1	0.0	109	82.2*	9
rmatr100-p10	49.0	49.2	2.6	16.4	686	424	322
rmatr100-p5	63.0	63.6	3.9	13.0	258	976*	118
rmine6	64.7	66.9	2.3	2730.2	2638869	-457.1727	1416590
rocII-4-11	85.9	89.6	14.3	0.0	0	–	–
rococoC10-001000	93.4	100.0	0.3	0.0	1	23730	1
roll3000	67.5	72.3	1.1	0.0	0	–	–
rout	88.9	93.9	0.2	0.0	0	–	–
satellites1-25	90.2	98.8	35.6	0.0	0	–	–
set1ch	45.1	44.6	1077.1	limit	41287842	–	–
seymour	48.3	48.9	3.4	limit	700335	428	607424
sp98ic	99.3	99.3	2.1	19.2	70178	469766019	478
sp98ir	94.3	97.1	2.3	0.0	0	–	–
stein27	22.2	22.2	0.1	0.0	224	18*	18
stein45	22.2	22.2	0.0	0.4	1507	30*	510
swath	99.3	98.7	2.6	0.0	0	–	–
t1717	99.2	99.4	7.4	0.4	0	–	–
tanglegram1	99.2	99.2	2.0	0.2	0	–	–
tanglegram2	96.9	97.1	0.3	0.0	0	–	–
timtab1	36.3	51.7	0.1	0.0	1	–	–
timtab2	22.8	42.8	0.0	0.1	1	–	–
tr12-30	7.4	6.2	70.5	limit	11643684	–	–
triptim1	87.1	99.7	103.8	0.2	0	–	–
unitcal_7	80.1	48.3	29.3	0.1	0	–	–
vpm2	63.9	77.3	0.0	0.0	14	15.25	12
vpphard	97.8	98.0	17.1	0.2	0	–	–
zib54-UUE	26.3	25.4	3.4	limit	1588278	10334015.8*	64873

Table B.8.: RENS: computing optimal roundings for MIQCP instances, using LP solution, after cuts

	% Vars Fixed			RENS			
Instance	Int	All	TimeS	Time	Nodes	Solution	Found At
10bar2	77.3	76.0	0.2	0.0	14	2691.7039	13
25bar	83.9	49.2	0.1	0.1	20	–	–
classical_200_0	92.0	59.8	1.6	1.7	31	-0.0848	26
classical_200_1	90.5	59.0	1.5	6.3	313	-0.097	195
classical_20_0	60.0	25.0	0.1	0.1	15	-0.0686	11
classical_20_1	55.0	23.3	0.0	0.3	40	-0.0698	16
classical_50_0	72.0	42.7	0.4	0.9	116	-0.0818	99
classical_50_1	82.0	49.3	0.2	0.6	69	-0.0737	6
clay0203m	20.0	14.8	0.1	0.0	55	41573.0147*	10
clay0205m	35.6	32.0	0.1	0.4	341	8672.5	184
clay0303m	21.1	22.6	0.1	0.1	61	41573.0276	53
clay0305m	29.4	25.9	0.2	0.5	724	8488.3117	716
du-opt5	54.5	5.3	0.1	0.2	29	–	–
du-opt	30.8	0.0	0.1	1.8	335	–	–
ex1263	69.0	69.2	0.2	0.0	1	28.3	1
ex1266	81.7	97.6	0.2	0.0	1	21.3	1
fac3	0.0	0.0	0.1	0.1	25	31982309.8*	13
feedtray2	41.7	29.7	24.4	limit	2358782	–	–
ibell3a	88.3	85.2	0.1	0.0	1	878785.031*	1
icvxqp1	99.7	100.0	454.9	0.6	1	914601	1
ilaser0	0.0	5.7	1.2	limit	237295	–	–
imod011	71.1	23.4	233.9	6341.0	345627	362636789	333111
iportfolio	80.1	64.5	4.3	283.9	26983	–	–
isqp	62.0	2.4	3.9	limit	97291	–	–
itointqor	86.0	94.1	0.0	0.0	1	53624064.4	1
ivalues	68.8	40.9	0.8	0.0	1	9026.4463	1
meanvarx	83.3	66.7	0.0	0.0	5	14.3692*	4
netmod_dol1	16.7	16.7	1.3	4622.7	82905	-0.5562	99
netmod_dol2	47.4	36.1	1.9	774.4	24560	-0.545	3448
netmod_kar1	0.0	0.0	0.3	1.9	327	-0.4198*	8
netmod_kar2	0.0	0.0	0.3	1.8	327	-0.4198*	8
nous1	0.0	0.0	290.7	limit	6203637	–	–
nous2	0.0	0.0	393.2	limit	5777698	–	–
nuclear14a	83.5	63.6	16.7	limit	94439	–	–
nuclear14b	92.7	71.7	2.0	3.7	111	–	–
nvs19	0.0	0.0	0.0	0.0	9	-1098.4*	9
nvs23	0.0	0.0	0.1	0.0	1	-1124.2	1
product2	81.2	26.9	162.5	limit	5890550	–	–
product	67.4	50.4	0.7	389.8	650612	-2130.6323	255299
robust_100_0	88.1	41.9	1.1	0.7	23	-0.0888	12
robust_100_1	86.1	41.2	0.9	1.6	123	-0.0525	63
robust_200_0	89.6	43.8	2.0	1.9	121	-0.0944	20
robust_20_0	85.7	32.5	0.1	0.0	5	-0.0759	2
robust_50_0	82.4	37.4	0.5	0.3	38	-0.0671	16
robust_50_1	82.4	37.4	0.5	0.4	50	-0.0714	34
shortfall_100_0	76.2	35.9	0.9	0.9	45	-1.0737	36
shortfall_100_1	83.2	39.4	1.1	0.8	33	-1.0657	32
shortfall_200_0	88.6	43.2	2.5	3.0	45	-1.0803	45
shortfall_20_0	71.4	25.0	0.0	0.1	11	-1.0811	10
shortfall_50_0	72.5	31.9	0.4	0.7	27	-1.0799	21
shortfall_50_1	78.4	34.8	0.3	0.5	21	-1.0806	18
SLay05H	67.5	60.6	0.3	0.3	33	24809.6753	31
SLay05M	55.0	43.7	0.1	0.1	33	33732.8607	9
SLay07M	71.4	48.9	0.1	0.4	63	73105.8847	33
SLay10H	41.1	38.2	19.6	limit	754162	131656.989	105106
SLay10M	68.3	51.9	0.4	1.9	415	185502.124	392

Table B.8 continued

	% Vars Fixed			RENS			
Instance	Int	All	TimeS	TimeR	NodesR	Solution	Found At
space25a	96.7	82.5	0.2	0.0	5	–	–
space25	94.6	80.1	0.4	0.7	8407	–	–
spectra2	80.0	70.6	0.4	0.1	26	13.9783*	14
tln12	48.2	52.2	0.2	0.0	0	–	–
tln5	74.3	77.1	0.0	0.0	0	–	–
tln6	64.6	68.8	0.1	0.0	0	–	–
tln7	42.9	49.2	0.1	0.0	0	–	–
tloss	69.6	82.6	0.0	0.0	0	–	–
tltr	25.5	39.3	0.1	0.0	0	–	–
uflquad-15-60	0.0	0.0	2.8	2679.7	1052	1063.1929*	237
uflquad-20-50	0.0	0.0	25.1	limit	128	474.9019	64
uflquad-40-80	97.5	85.1	1.7	limit	2	–	–
util	91.7	46.9	0.0	0.0	10	1000.9676	10
waste	97.5	91.5	0.4	0.1	426	692.7824	291

Table B.9.: RENS: computing optimal roundings for MIQCP instances, using NLP solution, after cuts

	% Vars Fixed			RENS			
Instance	Int	All	TimeS	Time	Nodes	Solution	Found At
10bar2	0.0	0.0	0.2	5.1	2678	1960.4104	2571
25bar	23.0	12.5	0.2	5.2	1199	400.3246	1192
classical_200_0	0.0	0.0	21.5	limit	157452	-0.1042	19694
classical_200_1	0.0	0.0	24.2	limit	184429	-0.1092	67634
classical_20_0	0.0	0.0	0.1	1.0	1354	-0.0823*	834
classical_20_1	0.0	0.0	-0.0	2.4	1835	-0.0757*	1747
classical_50_0	0.0	0.0	0.4	784.7	199803	-0.0907*	133471
classical_50_1	0.0	0.0	0.2	61.8	20511	-0.0948*	17026
clay0203m	0.0	0.0	0.1	0.1	110	41573.0265*	95
clay0205m	0.0	0.0	0.2	3.0	10442	8092.5*	1759
clay0303m	0.0	0.0	0.1	0.2	167	26669.0752	156
clay0305m	0.0	0.0	0.2	6.1	17597	8092.5*	1579
du-opt5	45.5	5.3	0.1	0.1	25	–	–
du-opt	0.0	0.0	0.1	34.0	6827	–	–
ex1263	45.1	52.7	0.3	0.2	70	20.3	49
ex1266	65.9	69.6	0.2	0.1	40	16.3*	40
fac3	8.3	1.5	1.0	0.0	23	31982309.8*	13
feedtray2	0.0	0.0	0.1	247.5	96287	0*	96287
ibell3a	60.0	82.8	0.1	0.0	1	879009.262	1
icvxqp1	97.6	98.1	580.3	0.6	1	375878	1
ilaser0	0.0	7.7	1.0	0.0	0	–	–
imod011	–	–	1346.6	–	–	–	–
iportfolio	0.0	0.0	6.9	limit	276015	–	–
isqp	0.0	0.0	331.7	limit	800472	–	–
itointqor	0.0	0.0	60.4	limit	31848641	-1145.95	30734174
ivalues	51.5	6.4	0.7	45.2	262102	-1.1657*	20497
meanvarx	58.3	56.7	0.1	0.0	5	14.3692*	4
netmod_dol1	0.0	0.0	13.8	limit	70283	-0.56*	197
netmod_dol2	24.4	24.1	4.7	12.3	365	-0.5208	216
netmod_kar1	0.0	0.0	0.4	1.9	327	-0.4198*	8
netmod_kar2	0.0	0.0	0.2	1.9	327	-0.4198*	8
nous1	0.0	0.0	295.1	limit	6189939	–	–
nous2	0.0	0.0	401.9	limit	5775976	–	–
nuclear14a	0.0	0.0	18.4	limit	98876	–	–
nuclear14b	0.0	0.0	39.2	limit	122109	–	–
nvs19	0.0	0.0	0.0	0.1	53	-1098.2	52
nvs23	0.0	0.0	0.0	0.2	75	-1124.8	73
product2	9.4	11.5	226.2	limit	5344387	–	–
product	67.4	41.6	159.1	limit	3714246	–	–
robust_100_0	0.0	0.0	36.0	limit	643608	-0.0964	432103
robust_100_1	0.0	0.0	28.5	limit	749339	-0.0716	500948
robust_200_0	0.0	0.0	20.3	limit	194374	-0.1359	57193
robust_20_0	0.0	0.0	0.1	0.1	11	-0.0798*	6
robust_50_0	0.0	0.0	0.6	1.2	270	-0.0861*	156
robust_50_1	0.0	0.0	0.3	12.3	3064	-0.0857*	754
shortfall_100_0	0.0	0.0	51.8	limit	418270	-1.1023	57765
shortfall_100_1	0.0	0.0	60.5	limit	459850	-1.094	168978
shortfall_200_0	0.0	0.0	32.9	limit	130390	-1.1096	10874
shortfall_20_0	0.0	0.0	0.1	0.6	624	-1.0905*	157
shortfall_50_0	0.0	0.0	74.0	limit	1248837	-1.095	930028
shortfall_50_1	0.0	0.0	0.5	1975.5	520190	-1.1018*	427638
SLay05H	0.0	0.0	0.2	7.0	3094	22664.678*	1400
SLay05M	0.0	0.0	0.1	1.7	878	22664.6781*	536
SLay07M	0.0	0.0	0.0	59.6	29886	64748.8243*	9877
SLay10H	0.0	0.0	18.5	limit	468624	130031.675	129100
SLay10M	0.0	0.0	16.4	limit	834497	129771.879	740342

Table B.9 continued

Instance	% Vars Fixed Int	All	TimeS	RENS TimeR	NodesR	Solution	Found At
space25a	41.7	32.5	0.3	limit	6254	–	–
space25	41.7	34.4	0.5	limit	894	–	–
spectra2	80.0	70.6	0.5	0.1	26	13.9783*	14
tln12	2.4	0.0	0.5	0.0	0	–	–
tln5	22.9	40.0	0.1	0.0	0	–	–
tln6	18.8	35.4	0.1	0.0	0	–	–
tln7	19.0	31.7	0.1	0.0	0	–	–
tloss	69.6	82.6	0.1	0.0	0	–	–
tltr	27.7	73.2	0.1	0.0	0	–	–
uflquad-15-60	0.0	0.0	2.9	2701.2	1052	1063.1929*	237
uflquad-20-50	0.0	0.0	25.1	limit	128	474.9019	64
uflquad-40-80	0.0	0.0	3.0	limit	1083	–	–
util	0.0	0.0	0.0	0.1	455	999.5788*	224
waste	86.3	75.1	401.5	limit	13736796	–	–

Table B.10.: RENS: computing optimal roundings for MINLP instances, using LP solution, after cuts

	% Vars Fixed			RENS			
Instance	Int	All	TimeS	Time	Nodes	Solution	Found At
beuster	76.5	40.2	118.1	7049.7	15735321	–	–
cecil_13	25.0	19.4	18.5	7010.4	6032779	-115599.148	6497
chp_partload	35.7	2.8	4.5	7158.7	13536	–	–
contvar	89.7	13.6	2.6	limit	89028	–	–
csched1	95.0	78.7	0.1	0.0	45	-29775.9885	45
csched2a	60.0	38.5	76.1	7059.7	5436390	-94800.4303	2043936
eg_all_s	28.6	53.0	589.3	6598.6	80557	–	–
eg_disc2_s	0.0	13.4	286.4	6970.9	22	–	–
eg_disc_s	50.0	36.6	316.1	6886.8	858	–	–
eg_int_s	0.0	14.3	501.2	6723.7	5	–	–
eniplac	30.4	26.2	0.1	0.1	151	-132117.083*	37
enpro48	80.4	77.3	0.1	15.5	111594	241150.752	111594
enpro48pb	79.3	71.4	0.0	1.1	4634	264032.12	4634
enpro56	67.1	56.0	0.2	17.7	147897	279702.866	147897
enpro56pb	65.7	53.6	0.1	5.4	41063	279704.1	41063
ex1233	20.0	7.2	1.6	limit	189292	–	–
ex1244	40.0	36.7	0.2	0.0	28	84035.1235	23
ex1252a	77.8	57.8	0.0	0.0	0	–	–
ex1252	71.4	55.4	0.1	1.2	5342	–	–
feedtray	42.9	1.2	68.9	7115.3	874824	–	–
fo7_2	19.0	9.8	0.1	3.3	14805	17.7493*	627
fo7_ar2_1	24.4	12.3	0.1	291.9	2381312	26.9425	2381312
fo7_ar25_1	36.6	18.5	0.1	0.4	755	25.6421	326
fo7_ar3_1	43.9	22.2	0.0	0.5	976	25.6421	316
fo7_ar4_1	29.3	14.8	0.1	2.3	9622	24.3794	4178
fo7_ar5_1	34.1	17.3	0.0	0.9	2147	19.6229	566
fo7	16.7	8.5	0.0	120.9	558800	30.6572	382347
fo8_ar2_1	36.4	20.8	0.2	2.1	6262	41.8507	3493
fo8_ar25_1	16.4	8.9	0.2	108.1	453566	28.0452*	84041
fo8_ar3_1	38.2	20.8	0.1	2.9	8133	–	–
fo8_ar4_1	30.9	16.8	0.1	146.9	975930	32.5005	968495
fo8_ar5_1	30.9	16.8	0.2	6.1	21065	24.4077	3434
fo8	21.4	11.8	0.2	592.2	2279417	37.2612	216937
fo9_ar2_1	23.9	13.8	0.1	1.7	5290	45.8141	3577
fo9_ar25_1	35.2	20.3	0.2	15.5	46324	32.6795	23480
fo9_ar3_1	22.5	13.0	0.1	598.5	1658625	37.5937	8325
fo9_ar4_1	25.4	14.6	0.2	879.3	2599259	37.1576	29588
fo9_ar5_1	28.2	16.3	0.2	65.9	196069	26.9217	134598
fo9	19.4	11.3	34.7	7053.4	20841677	34.6228	6480181
fuzzy	71.8	42.6	86.8	7126.7	4547053	–	–
gasnet	50.0	23.6	0.1	limit	3063	–	–
ghg_1veh	0.0	0.0	386.4	7108.6	6426635	–	–
ghg_2veh	18.8	7.6	109.5	7130.6	1936310	–	–
ghg_3veh	51.4	21.3	37.6	7163.3	1587865	–	–
hda	28.6	18.0	6.6	limit	588838	–	–
m6	3.3	1.6	0.0	2.2	11390	82.2569*	3883
m7_ar2_1	13.3	5.9	0.1	1.5	10467	195.035	9794
m7_ar25_1	18.8	8.6	0.1	0.2	443	143.585*	204
m7_ar3_1	34.2	17.1	0.1	0.5	772	152.5792	330
m7_ar4_1	34.1	17.7	0.2	0.3	730	130.46	287
m7_ar5_1	26.8	13.9	0.1	1.0	4354	148.6199	1740
m7	33.3	17.5	0.1	0.1	341	126.4312	196
mbtd	–	–	limit	–	–	–	–
no7_ar2_1	36.6	17.2	0.2	0.8	1772	150.7814	740
no7_ar25_1	26.8	12.6	0.0	2.7	9032	107.8663	7186
no7_ar3_1	26.8	12.6	0.2	1.2	3223	119.3432	2131

Table B.10 continued

Instance	% Vars Fixed Int	% Vars Fixed All	TimeS	RENS TimeR	RENS NodesR	Solution	Found At
no7_ar4_1	43.9	20.7	0.0	1.1	3492	117.8947	2278
no7_ar5_1	24.4	11.5	0.1	28.6	104622	100.8113	10082
nvs09	60.0	55.0	534.1	6969.9	77479321	-11.1518	15924294
nvs20	20.0	6.1	0.0	1.2	1948	230.9221*	1580
o7_2	31.0	14.4	0.1	8.6	33559	129.4105	2060
o7_ar2_1	31.7	14.6	0.2	2.8	10741	140.4119*	188
o7_ar25_1	36.6	16.9	0.1	29.0	182612	143.1372	182612
o7_ar3_1	26.8	12.4	0.1	10.9	34069	–	–
o7_ar4_1	26.8	12.4	0.1	7.2	27844	143.8912	24195
o7_ar5_1	46.3	21.3	0.1	31.0	213317	135.7148	213317
o7	19.0	8.9	0.1	428.8	1812739	139.4551	207218
o8_ar4_1	32.7	15.4	0.2	28.0	65139	–	–
o9_ar4_1	39.4	20.4	0.1	119.5	311859	–	–
oil2	50.0	0.5	1.6	limit	1205253	–	–
oil	57.9	8.3	24.3	7177.6	165976	–	–
parallel	20.0	14.7	8.1	7184.6	899801	924.225	834864
pump	77.8	57.8	0.0	0.0	0	–	–
risk2b	66.7	5.6	0.2	0.0	11	-55.8761*	9
spring	91.7	67.9	0.0	0.0	0	–	–
st_e32	88.9	29.7	0.1	0.0	3	–	–
stockcycle	86.8	91.3	0.8	0.0	51	334280.188	46
super1	83.9	10.0	1.2	0.0	0	–	–
super2	71.0	8.4	1.1	0.0	0	–	–
super3	67.6	8.6	1.2	0.0	0	–	–
super3t	35.1	6.2	8.8	7157.6	76873	–	–
synheat	20.0	8.0	17.7	limit	3475310	–	–
synthes1	0.0	0.0	0.0	0.0	5	6.0098*	4
synthes2	50.0	36.4	0.0	0.0	6	73.0353*	6
synthes3	42.9	29.4	0.1	0.0	11	68.0097*	10
tls12	93.7	81.0	1.7	0.0	0	–	–
tls4	55.3	53.2	0.2	0.2	417	11.5	338
tls5	64.1	64.0	0.5	0.4	2073	12.5	2043
tls6	86.1	83.1	0.3	0.0	0	–	–
tls7	90.7	64.9	0.5	0.0	0	–	–
water3	67.9	35.3	0.1	292.7	972217	907.0153	779595
waterful2	92.9	76.4	0.2	4.8	14332	944.0185	13167
watersbp	25.0	19.8	0.3	695.4	2039425	925.5489	1871298
watersym1	71.4	57.1	0.1	13.6	53787	914.5702	48361
watersym2	83.3	55.6	0.1	10.8	28608	1056.1449	25709
waterx	78.6	24.0	0.1	limit	91	–	–
detf1	81.5	1.2	1579.0	5733.5	367	–	–
gear2	70.8	57.6	0.0	0.0	20	0*	13
gear3	50.0	11.1	0.0	0.0	2	0.0164	2
gear4	50.0	22.2	0.0	0.0	4	495720.675	4
gear	50.0	11.1	0.0	0.0	2	0.0164	2
johnall	98.9	9.0	63.2	13.0	18	-224.7302*	16
saa_2	81.5	1.2	1579.0	5733.3	367	–	–
water4	65.1	48.3	0.8	5.5	12624	926.9473	10394
waterz	75.4	58.0	0.2	0.1	63	–	–

Table B.11.: RENS: computing optimal roundings for MINLP instances, using NLP solution, after cuts

	% Vars Fixed			RENS			
Instance	Int	All	TimeS	Time	Nodes	Solution	Found At
beuster	–	–	0.1	–	–	–	–
cecil_13	37.5	30.8	1.2	775.6	1225350	-115630.852	720438
chp_partload	21.4	1.5	17.6	7146.9	9859	–	–
contvar	–	–	1.9	–	–	–	–
csched1	26.7	20.0	0.1	6864.6	51911752	-30639.353*	510093
csched2a	60.0	52.2	3.6	limit	58208	–	–
eg_all_s	85.7	83.1	682.2	6529.9	1220160	–	–
eg_disc2_s	–	–	798.3	–	–	–	–
eg_disc_s	–	–	546.0	–	–	–	–
eg_int_s	–	–	1011.3	–	–	–	–
eniplac	47.8	42.6	0.2	0.0	28	-130450.77	22
enpro48	82.6	73.4	0.1	3.5	28731	198547.396	28731
enpro48pb	82.6	73.4	0.2	2.3	17748	198547.384	17748
enpro56	68.6	56.8	0.2	8.0	75178	271493.619	75178
enpro56pb	68.6	56.8	0.1	4.7	41949	271496.644	41949
ex1233	0.0	0.0	436.5	7042.5	8215104	–	–
ex1244	0.0	0.0	0.2	0.4	562	82042.2724*	307
ex1252a	0.0	60.0	4.0	0.0	0	–	–
ex1252	28.6	33.9	1.6	1.4	317	131123.771	292
feedtray	14.3	0.4	25.3	limit	406512	–	–
fo7_2	0.0	0.0	0.1	135.9	704358	17.7493*	2293
fo7_ar2_1	0.0	0.0	0.2	46.8	247054	24.8398*	19889
fo7_ar25_1	0.0	0.0	0.1	24.6	115558	23.0936*	105003
fo7_ar3_1	0.0	0.0	0.1	136.1	668929	22.5175*	17122
fo7_ar4_1	0.0	0.0	0.1	155.0	733240	20.7298*	350369
fo7_ar5_1	0.0	0.0	0.1	151.7	767719	17.7493*	68937
fo7	0.0	0.0	0.1	497.1	2372596	20.7298*	240205
fo8_ar2_1	0.0	0.0	0.2	934.6	3788852	30.3406*	1263812
fo8_ar25_1	0.0	0.0	0.2	1106.3	4787074	28.0452*	1555470
fo8_ar3_1	0.0	0.0	0.2	231.3	898814	23.9101*	126001
fo8_ar4_1	0.0	0.0	0.2	234.6	969121	22.3819*	214458
fo8_ar5_1	0.0	0.0	0.1	1432.4	5813287	22.3819*	1898654
fo8	0.0	0.0	6.5	7001.8	26796040	22.3819*	316351
fo9_ar2_1	0.0	0.0	12.1	7024.9	22193275	32.625*	1452885
fo9_ar25_1	0.0	0.0	24.3	7023.6	22803832	32.25	20506093
fo9_ar3_1	0.0	0.0	0.2	1052.7	3352680	24.8155*	336767
fo9_ar4_1	0.0	0.0	16.7	7033.6	28964871	23.4643*	1012573
fo9_ar5_1	0.0	0.0	13.7	7024.4	20112356	23.4643*	1774865
fo9	0.0	0.0	30.3	7040.1	22676841	26.4643	15213281
fuzzy	16.4	6.3	7.8	0.0	3	–	–
gasnet	90.0	39.9	6.3	limit	191339	–	–
ghg_1veh	0.0	0.0	382.9	7085.1	6381143	–	–
ghg_2veh	0.0	0.0	56.0	7146.3	1083873	–	–
ghg_3veh	17.1	21.3	33.2	7164.7	1775681	–	–
hda	14.3	7.1	32.8	7149.4	1682642	–	–
m6	0.0	0.0	0.1	4.1	24562	82.2569*	6680
m7_ar2_1	0.0	0.0	0.2	2.1	10276	190.235*	3930
m7_ar25_1	0.0	0.0	0.2	1.1	3726	143.585*	138
m7_ar3_1	0.0	0.0	0.0	6.3	28008	143.585*	1817
m7_ar4_1	0.0	0.0	0.1	9.3	44016	106.7569*	15850
m7_ar5_1	0.0	0.0	0.0	32.8	173785	106.46*	53909
m7	0.0	0.0	0.1	8.2	48013	106.7569*	20018
mbtd	–	–	limit	–	–	–	–
no7_ar2_1	0.0	0.0	0.2	219.9	1033148	107.8153*	325747
no7_ar25_1	0.0	0.0	0.1	379.1	1545736	107.8153*	548721
no7_ar3_1	0.0	0.0	0.1	506.6	1988914	107.8153*	118955

Table B.11 continued

Instance	% Vars Fixed Int	All	TimeS	RENS TimeR	NodesR	Solution	Found At
no7_ar4_1	0.0	0.0	0.1	2571.1	13791699	98.5184*	9316640
no7_ar5_1	0.0	0.0	0.2	3548.9	14641250	90.6227*	2261480
nvs09	–	–	0.2	–	–	–	–
nvs20	0.0	0.0	0.0	1.2	1668	230.9221*	1585
o7_2	0.0	0.0	42.7	7032.6	26786207	116.9459*	19601790
o7_ar2_1	0.0	0.0	0.2	403.4	1959250	140.4119*	360093
o7_ar25_1	0.0	0.0	0.2	1184.8	4608236	140.7327	293836
o7_ar3_1	0.0	0.0	0.2	2486.6	9747119	137.9318*	3672646
o7_ar4_1	0.0	0.0	4.4	7040.1	26611055	131.6531*	3627436
o7_ar5_1	0.0	0.0	0.9	6992.7	30028960	116.9458*	3480829
o7	0.0	0.0	27.8	7014.7	26516141	131.6531*	544651
o8_ar4_1	0.0	0.0	23.1	7088.4	18402307	245.4744	8887518
o9_ar4_1	0.0	0.0	46.7	7025.3	19840728	250.1082	9730833
oil2	0.0	0.0	35.8	7133.9	1001767	–	–
oil	0.0	0.1	33.7	7149.0	119377	–	–
parallel	20.0	14.7	14.4	limit	900521	924.225	834864
pump	33.3	40.0	0.9	5.7	146	131123.769	143
risk2b	0.0	0.0	0.1	0.1	53	-55.8761*	25
spring	0.0	0.0	0.1	0.0	44	0.9876	34
st_e32	83.3	40.6	0.1	0.0	1	–	–
stockcycle	24.3	21.8	2.6	7159.6	6417875	128864.597	3237213
super1	16.1	1.1	13.5	0.0	1	–	–
super2	16.1	1.2	10.8	0.0	1	–	–
super3	21.6	2.7	16.6	0.0	1	–	–
super3t	0.0	0.0	7.4	7196.9	48370	–	–
synheat	0.0	0.0	3.6	limit	512341	–	–
synthes1	0.0	0.0	0.0	0.0	5	6.0098*	4
synthes2	0.0	0.0	0.0	0.0	16	73.0353*	12
synthes3	0.0	0.0	0.0	11.4	172409	68.0098*	172409
tls12	29.6	67.2	45.9	7125.8	8583097	–	–
tls4	27.1	28.2	0.3	14.0	85190	11.5	4135
tls5	34.4	36.4	0.4	136.8	663149	12.1	49484
tls6	45.5	50.7	0.3	275.6	1106419	–	–
tls7	72.4	78.5	0.5	0.3	965	–	–
water3	3.6	6.3	51.4	6929.7	20651925	908.5771	11154642
waterful2	64.3	58.0	233.9	6956.7	21237400	1727.7383	12114
watersbp	3.6	6.3	139.1	6965.6	21701297	926.9473	1393039
watersym1	42.9	38.0	41.6	6934.4	24032000	945.8494	823376
watersym2	50.0	41.2	0.6	1649.9	5395774	955.728	1697926
waterx	0.0	0.0	7.4	6967.9	983262	–	–
detf1	41.0	0.6	1599.0	5649.0	608	–	–
gear2	0.0	0.0	0.0	0.2	896	-0*	896
gear3	0.0	0.0	0.0	0.0	5	0*	4
gear4	0.0	0.0	0.0	0.0	5	333.1514	4
gear	0.0	0.0	0.0	0.0	5	0*	4
johnall	0.0	0.0	63.6	8.2	1	-224.7302*	1
saa_2	41.0	0.6	1601.4	5649.3	608	–	–
water4	64.3	54.6	0.7	0.8	2430	1008.4471	1819
waterz	65.1	44.4	0.6	36.6	98729	2600.6081	98389

Table B.12.: RENS: analyzing rounding heuristics for MMM instances

Instance	RENS	ZI Round	Rounding	Simple Rounding
10teams	–	–	–	–
30n20b8	–	–	–	–
a1c1s1	13209.184	–	–	–
acc-tight5	–	–	–	–
aflow30a	1158	–	–	–
aflow40b	1179	–	–	–
air04	–	–	–	–
air05	–	–	–	–
app1-2	–	–	–	–
arki001	–	–	–	–
ash608gpia-3col	–	–	–	–
atlanta-ip	98.009586	–	–	–
beasleyC3	789	1690	1730	1730
bab5	–	–	–	–
bell3a	878430.32	880414.28	–	–
bell5	–	–	–	–
biella1	3278480.6	–	–	–
bienst2	54.6	–	–	–
binkar10_1	6746.64	–	–	–
blend2	7.598985	–	–	–
bley_xl1	190	–	–	–
bnatt350	–	–	–	–
cap6000	-2443599	-2443599	-2441736	-2441736
core2536-691	695	1103	1651	–
cov1075	20	43	90	90
csched010	–	–	–	–
dano3mip	762.75	–	–	–
danoint	65.666667	–	–	–
dcmulti	188186.5	–	–	–
dfn-gwin-UUM	39920	199352	209984	209984
disctom	–	–	–	–
ds	–	–	–	–
dsbmip	-305.19817	–	–	–
egout	568.1007	597.46403	597.46403	597.46403
eil33-2	–	–	–	–
eilB101	–	–	–	–
enigma	–	–	–	–
enlight13	–	–	–	–
enlight14	–	–	–	–
fast0507	177	315	540	540
fiber	411151.82	–	–	–
fixnet6	3997	10723.928	10723.928	10723.928
flugpl	–	–	–	–
gesa2-o	25780031	–	–	–
gesa2	25780031	–	–	–
gesa3	27991430	–	–	–
gesa3_o	27991430	–	–	–
glass4	2.2666856e+09	–	–	–
gmu-35-40	-2399398.2	–	–	–
gt2	21166	21166	–	–
harp2	–	–	–	–
iis-100-0-cov	29	55	100	100
iis-bupa-cov	36	71	144	144
iis-pima-cov	33	66	130	130
khb05250	1.0694023e+08	1.1688827e+08	1.1688827e+08	1.1688827e+08
l152lav	–	–	–	–
lectsched-4-obj	–	–	–	–
liu	3418	–	–	–

Table B.12 continued

Instance	RENS	ZI Round	Rounding	Simple Rounding
lseu	1148	–	–	–
m100n500k4r1	-22	-9	0	0
macrophage	–	–	–	–
map18	-847	–	–	–
map20	-922	–	–	–
markshare1	142	584	2108	2108
markshare2	131	531	2288	2288
mas74	14343.468	–	–	–
mas76	40560.054	–	–	–
mcsched	213768	–	–	–
mik-250-1-100-1	-66729	-66409	-66409	-66409
mine-90-10	-7.8430234e+08	–	–	–
misc03	–	–	–	–
misc06	12850.861	12920.927	12920.927	12920.927
misc07	–	–	–	–
mitre	115155	–	–	–
mkc	-539.866	–	–	–
mod008	309	452	1212	1212
mod010	–	–	–	–
mod011	-54219146	–	–	–
modglob	20799459	21051934	21051934	21051934
momentum1	–	–	–	–
momentum2	–	–	–	–
momentum3	–	–	–	–
msc98-ip	–	–	–	–
mspp16	–	–	–	–
mzzv11	–	–	–	–
mzzv42z	–	–	–	–
n3div36	151600	230600	562600	–
n3seq24	68000	–	–	–
n4-3	9010	20686.357	23686.357	23686.357
neos-1109824	–	–	–	–
neos-1337307	-202133	–	–	–
neos-1396125	3000.0553	–	–	–
neos13	-65.655161	–	–	–
neos-1601936	–	–	–	–
neos18	–	–	–	–
neos-476283	406.81233	–	–	–
neos-686190	–	–	–	–
neos-849702	–	–	–	–
neos-916792	–	–	–	–
neos-934278	1332	–	–	–
net12	–	–	–	–
netdiversion	–	–	–	–
newdano	66.5	–	–	–
noswot	–	–	–	–
ns1208400	–	–	–	–
ns1688347	–	–	–	–
ns1758913	-457.71835	–	–	–
ns1766074	–	–	–	–
ns1830653	–	–	–	–
nsrand-ipx	55360	–	114560	–
nw04	–	–	–	–
opm2-z7-s2	-10271	-3937	–	–
opt1217	-16	–	–	–
p0201	7805	–	–	–
p0282	258411	400676	373318	–
p0548	8763	–	–	–
p2756	3152	–	–	–
pg5_34	-14287.702	–	–	–

Table B.12 continued

Instance	RENS	ZI Round	Rounding	Simple Rounding
pigeon-10	–	–	–	–
pk1	29	–	–	–
pp08a	7360	12657.971	12657.971	12657.971
pp08aCUTS	7370	13128.015	13128.015	13128.015
protfold	–	–	–	–
pw-myciel4	–	–	–	–
qiu	-132.87314	1805.1771	1805.1771	1805.1771
qnet1	21237.655	–	–	–
qnet1_o	22600.83	–	45561.556	–
rail507	178	319	550	–
ran16x16	3846	10305.599	10305.599	10305.599
reblock67	-34629816	–	–	–
rd-rplusc-21	–	–	–	–
rentacar	30356761	–	–	–
rgn	82.199998	–	–	–
rmatr100-p10	424	–	–	–
rmatr100-p5	976	–	–	–
rmine6	-457.17275	-435.70014	–	–
rocII-4-11	–	–	–	–
rococoC10-001000	12067	–	87872	–
roll3000	14193	–	–	–
rout	–	–	–	–
satellites1-25	–	–	–	–
set1ch	54537.75	59480.277	59480.277	59480.277
seymour	427	590	757	757
sp98ic	4.6976602e+08	6.9404931e+08	1.3685495e+09	–
sp98ir	–	–	–	–
stein27	18	20	27	27
stein45	30	37	45	45
swath	–	–	–	–
t1717	–	–	–	–
tanglegram1	–	–	–	–
tanglegram2	–	–	–	–
timtab1	827609	–	–	–
timtab2	–	–	–	–
tr12-30	131438	–	–	–
triptim1	–	–	–	–
unitcal_7	–	–	–	–
vpm2	13.75	–	–	–
vpphard	–	–	–	–
zib54-UUE	10334016	19016948	19016948	19016948

Table B.13.: Impact of RENS on overall solving process for MMM instances

	No RENS		Root RENS		Tree RENS	
Instance	Nodes	Time	Nodes	Time	Nodes	Time
10teams	2 766	33.8	2 766	33.8	2 766	33.8
30n20b8	>13 609	limit	>13 098	limit	>13 480	limit
a1c1s1	>444 580	limit	>445 106	limit	>355 340	limit
acc-tight5	2 414	388.9	2 414	389.5	2 414	389.5
aflow30a	3 617	20.8	1 931	13.2	1 931	13.3
aflow40b	366 800	3221.7	230 705	1087.1	230 705	1085.1
air04	272	77.8	272	77.5	272	77.8
air05	478	45.8	478	44.5	478	44.5
app1-2	76	1139.8	76	1300.6	76	1302.4
arki001	2 703 497	4529.0	2 703 497	4527.6	2 703 497	4526.7
ash608gpia-3col	10	69.7	10	70.0	10	69.9
atlanta-ip	>8 841	limit	>8 520	limit	>8 520	limit
beasleyC3	>1 897 819	limit	>1 890 444	limit	>1 767 779	limit
bab5	>21 663	limit	>21 663	limit	>21 636	limit
bell3a	47 240	13.2	46 910	11.2	46 910	11.1
bell5	1 069	0.6	1 069	0.7	1 069	0.5
biella1	10 546	2284.0	2 607	939.9	2 607	953.5
bienst2	73 759	394.5	73 759	396.7	82 826	454.9
binkar10_1	105 531	158.8	105 531	159.3	129 286	204.9
blend2	2 135	1.9	164	0.7	164	0.9
bley_xl1	18	372.2	1	214.1	1	206.8
bnatt350	7 866	972.6	7 866	970.9	7 866	972.6
cap6000	3 005	2.5	3 005	2.6	3 005	2.8
core2536-691	204	383.3	281	652.9	281	653.5
cov1075	>1 719 951	limit	>1 721 430	limit	>1 697 293	limit
csched010	940 018	6394.7	940 018	6395.9	940 018	6397.6
dano3mip	>2 838	limit	>3 064	limit	>2 384	limit
danoint	1 063 562	5251.8	1 063 562	5237.1	1 063 562	5256.0
dcmulti	130	1.8	130	1.8	130	1.7
dfn-gwin-UUM	77 613	148.8	77 613	146.7	77 613	148.1
disctom	1	3.5	1	3.6	1	3.5
ds	>465	limit	>460	limit	>460	limit
dsbmip	1	0.7	1	0.6	1	0.6
egout	1	0.5	1	0.5	1	0.5
eil33-2	10 571	98.0	10 571	99.3	10 571	99.7
eilB101	9 239	773.3	9 239	777.1	9 239	776.3
enigma	1 289	0.6	1 289	0.6	1 289	0.7
enlight13	1 099 066	655.3	1 099 066	658.3	1 099 066	658.8
enlight14	156 998	108.9	156 998	108.3	156 998	108.1
fast0507	1 477	1474.5	2 774	3501.8	2 774	3509.4
fiber	78	1.9	32	1.3	32	1.2
fixnet6	54	1.8	14	1.8	14	1.9
flugpl	121	0.5	121	0.5	121	0.5
gesa2-o	55	1.8	4	1.5	4	1.5
gesa2	42	1.7	7	1.4	7	1.3
gesa3	147	2.3	16	1.7	16	1.6
gesa3_o	119	3.1	12	2.1	12	2.0
glass4	>10 167 913	limit	1 795 478	1454.2	1 795 478	1459.6
gmu-35-40	>5 151 788	limit	>11 990 260	limit	>13 431 923	limit
gt2	1	0.5	1	0.5	1	0.5
harp2	360 980	301.6	360 980	301.2	364 890	308.2
iis-100-0-cov	106 874	1706.4	106 874	1705.5	106 389	1828.4
iis-bupa-cov	183 185	6723.2	189 467	6655.7	189 467	6690.3
iis-pima-cov	13 766	952.6	13 011	953.7	13 011	966.1
khb05250	11	0.5	11	0.5	11	0.5
l152lav	52	3.0	52	2.9	52	3.1
lectsched-4-obj	11 988	246.4	11 988	246.4	11 988	247.4

Table B.13 continued

Instance	No RENS Nodes	No RENS Time	Root RENS Nodes	Root RENS Time	Tree RENS Nodes	Tree RENS Time
liu	>1 835 353	limit	>1 832 824	limit	>1 965 400	limit
lseu	329	0.5	552	0.5	552	0.5
m100n500k4r1	5 272 016	4732.9	>8 222 511	limit	>8 183 822	limit
macrophage	>929 901	limit	>925 398	limit	>928 739	limit
map18	607	649.6	293	463.1	293	463.8
map20	1 180	496.4	353	549.0	353	548.5
markshare1	>75 355 137	limit	>78 655 002	limit	>78 886 991	limit
markshare2	>63 825 711	limit	>62 613 242	limit	>62 433 221	limit
mas74	2 955 765	500.1	2 955 765	499.8	2 955 765	502.1
mas76	243 004	43.5	281 857	42.2	281 857	42.3
mcsched	16 113	222.9	16 113	222.2	20 712	256.3
mik-250-1-100-1	1 920 723	373.9	1 021 375	205.6	1 021 375	206.1
mine-90-10	469 802	1753.4	359 569	1156.5	359 569	1157.1
misc03	131	1.1	131	1.2	131	1.1
misc06	18	0.5	6	0.5	6	0.5
misc07	38 363	20.5	38 363	20.5	38 363	20.9
mitre	1	4.5	1	4.6	1	4.7
mkc	>3 288 146	limit	>3 186 952	limit	>3 223 059	limit
mod008	192	0.9	192	0.9	192	0.9
mod010	4	0.9	4	0.7	4	0.8
mod011	1 596	206.1	1 596	206.0	1 596	205.8
modglob	1 408	1.3	1 408	1.5	1 408	1.6
momentum1	>21 781	limit	>21 733	limit	>21 781	limit
momentum2	>63 180	limit	>61 812	limit	>62 495	limit
momentum3	>44	limit	>43	limit	>44	limit
msc98-ip	>756	limit	>756	limit	>756	limit
mspp16	>750	limit	>382	limit	>736	limit
mzzv11	2 734	341.8	2 734	343.5	2 734	342.3
mzzv42z	1 557	364.5	1 557	364.2	1 557	365.0
n3div36	>200 784	limit	>257 302	limit	>264 668	limit
n3seq24	>2 290	limit	>2 094	limit	>2 114	limit
n4-3	53 959	835.6	53 959	835.3	53 959	844.5
neos-1109824	24 162	185.9	24 162	185.4	24 162	186.1
neos-1337307	>415 472	limit	>416 447	limit	>413 169	limit
neos-1396125	54 219	3981.6	54 219	3981.4	54 219	3982.6
neos13	>28 166	limit	>26 778	limit	>25 527	limit
neos-1601936	>31 161	limit	>30 882	limit	>30 831	limit
neos18	9 133	41.4	9 133	41.4	9 133	41.5
neos-476283	466	326.9	609	323.2	609	327.1
neos-686190	9 894	114.1	9 894	114.7	9 894	114.3
neos-849702	137 579	1652.0	137 579	1651.7	137 579	1653.2
neos-916792	57 471	228.0	57 471	227.3	57 471	227.3
neos-934278	>2 951	limit	>4 825	limit	>4 708	limit
net12	3 838	2650.2	3 838	2647.9	3 838	2649.5
netdiversion	>72	limit	>72	limit	>72	limit
newdano	>1 570 960	limit	>1 574 108	limit	>1 138 936	limit
noswot	525 460	148.2	525 460	147.8	525 460	147.4
ns1208400	15 050	1960.2	15 050	1957.1	15 050	1956.6
ns1688347	17 807	1979.0	17 807	1978.5	17 807	1979.6
ns1758913	>23	limit	>17	limit	>5	limit
ns1766074	946 987	514.1	946 987	515.2	946 987	516.1
ns1830653	57 234	584.3	57 234	585.5	57 234	585.9
nsrand-ipx	>1 097 182	limit	>1 154 058	limit	>1 158 945	limit
nw04	5	51.1	5	52.0	5	51.9
opm2-z7-s2	4 401	1154.7	4 401	1153.8	4 401	1154.5
opt1217	>16 012 029	limit	>12 726 890	limit	>12 478 488	limit
p0201	169	1.9	65	1.6	65	1.8
p0282	26	0.8	3	0.6	3	0.5
p0548	96	0.8	14	0.5	14	0.5

Table B.13 continued

Instance	No RENS Nodes	No RENS Time	Root RENS Nodes	Root RENS Time	Tree RENS Nodes	Tree RENS Time
p2756	403	3.2	153	2.6	153	2.5
pg5_34	348 765	1717.1	318 742	1501.1	306 428	1374.3
pigeon-10	>7 056 792	limit	>7 034 031	limit	>6 972 773	limit
pk1	213 670	46.8	226 780	50.0	206 727	44.4
pp08a	590	1.5	590	1.5	670	1.7
pp08aCUTS	403	1.5	403	1.4	480	1.6
protfold	>6 866	limit	>6 865	limit	>6 862	limit
pw-myciel4	647 355	5306.6	647 355	5310.9	647 355	5311.9
qiu	11 012	56.2	11 012	56.3	10 301	55.9
qnet1	7	2.4	7	2.5	7	2.3
qnet1_o	29	3.9	29	4.0	29	3.9
rail507	1 704	1494.8	1 472	1269.2	1 472	1268.4
ran16x16	348 556	196.6	331 635	195.2	331 635	195.3
reblock67	111 964	279.5	111 964	279.1	111 964	279.7
rd-rplusc-21	>58 623	limit	>58 592	limit	>58 592	limit
rentacar	14	3.0	14	3.0	14	3.1
rgn	62	0.5	62	0.5	62	0.5
rmatr100-p10	901	197.3	901	197.7	864	201.0
rmatr100-p5	420	668.8	385	553.4	385	553.4
rmine6	541 456	2814.6	727 632	4044.6	523 315	2760.6
rocII-4-11	40 353	544.4	40 353	545.6	40 353	545.7
rococoC10-001000	662 755	3313.2	488 147	2372.7	495 582	2404.2
roll3000	>1 390 052	limit	>1 479 602	limit	>1 482 101	limit
rout	29 656	39.7	29 656	39.9	19 937	33.3
satellites1-25	9 089	2148.3	9 089	2146.1	9 089	2148.0
set1ch	28	0.9	6	0.8	6	0.9
seymour	>122 156	limit	>130 095	limit	>116 911	limit
sp98ic	>135 751	limit	>209 889	limit	>208 547	limit
sp98ir	4 912	64.8	4 912	64.9	4 912	65.1
stein27	4 045	0.9	4 045	1.1	4 045	1.0
stein45	52 523	13.1	52 523	13.1	52 523	13.3
swath	>1 448 548	limit	>1 460 957	limit	>1 433 029	limit
t1717	>734	limit	>720	limit	>734	limit
tanglegram1	27	867.6	27	866.3	27	860.5
tanglegram2	3	7.0	3	7.0	3	6.9
timtab1	925 706	412.1	925 706	413.2	925 706	414.5
timtab2	>8 939 001	limit	>8 943 388	limit	>8 926 669	limit
tr12-30	1 518 459	1986.3	1 685 757	2280.3	1 532 831	2052.5
triptim1	30	2002.7	30	1984.3	30	1993.2
unitcal_7	11 624	1173.8	10 569	1137.6	10 569	1138.7
vpm2	945	1.2	143	1.1	143	1.1
vpphard	>5 521	limit	>5 524	limit	>5 525	limit
zib54-UUE	951 366	5701.2	951 366	5708.5	865 298	4910.0
sh. geom. mean	11 248	377.2	10 390	366.3	10 346	365.8
arithm. mean	1 446 078	2461.4	1 442 400	2427.0	1 443 404	2414.3

Table B.14.: Impact of RENS on overall solving process for MIQCP instances

	No RENS		Root RENS		Tree RENS	
Instance	Nodes	Time	Nodes	Time	Nodes	Time
108bar	>1	>7200.0	>1	>7200.0	>1	>7200.0
10bar2	369	2.3	653	2.8	653	2.9
25bar	>7 936	>7200.0	>3 402	>7200.0	>3 402	>7200.0
classical_200_0	>100 675	>7200.0	>109 742	>7200.0	>109 204	>7200.0
classical_200_1	>152 012	>7200.0	>134 651	>7200.0	>131 226	>7200.0
classical_20_0	172	0.7	127	0.9	127	0.9
classical_20_1	866	1.7	897	1.9	897	2.1
classical_50_0	243 420	1068.1	1 260 971	5287.2	940 699	3782.0
classical_50_1	20 929	74.4	29 760	106.3	29 760	107.9
clay0203m	55	0.5	55	0.5	55	0.5
clay0205m	10 494	4.0	10 494	4.1	10 492	4.5
clay0303m	99	0.5	99	0.5	99	0.5
clay0305m	9 361	4.5	9 361	4.5	9 361	4.5
du-opt5	86	0.5	86	0.5	86	0.5
du-opt	322	0.7	322	0.7	322	0.8
ex1263	199	0.7	199	0.8	199	0.8
ex1266	37	0.7	255	1.1	255	1.1
fac3	6	0.5	6	0.5	6	0.5
feedtray2	1	0.5	1	0.5	1	0.5
ibell3a	44 048	12.9	42 066	13.8	42 066	13.8
icvxqp1	>1 897	>7200.0	>1 893	>7200.0	>1 903	>7200.0
ilaser0	169	3.2	169	3.0	169	3.2
imod011	1	319.2	1	319.4	1	319.4
iportfolio	>21 555	>7200.0	>21 527	>7200.0	>21 279	>7200.0
isqp0	>1 479 285	>7200.0	>1 483 123	>7200.0	>1 481 383	>7200.0
isqp1	>1 362 277	>7200.0	>1 362 255	>7200.0	>1 360 580	>7200.0
isqp	>1 706 210	>7200.0	>1 706 576	>7200.0	>1 706 619	>7200.0
ivalues	>153 470	>7200.0	>153 572	>7200.0	>153 088	>7200.0
meanvarx	7	0.5	3	0.5	3	0.5
netmod_dol1	62 794	6077.4	62 794	6049.4	62 028	6115.3
netmod_dol2	192	49.6	192	49.7	150	47.8
netmod_kar1	288	5.9	288	5.9	288	5.8
netmod_kar2	288	6.0	288	5.9	288	5.9
nous1	>5 156 737	>7200.0	>5 154 877	>7200.0	>5 149 665	>7200.0
nous2	2 821	2.2	2 821	2.0	2 821	2.2
nuclear14a	>36 917	>7200.0	>36 932	>7200.0	>53 127	>7200.0
nuclear14b	>73 331	>7200.0	>73 976	>7200.0	>73 751	>7200.0
nvs19	105	0.5	105	0.5	105	0.5
nvs23	96	0.5	96	0.5	96	0.5
product2	>6 014 234	>7200.0	>6 225 476	>7200.0	>5 740 865	>7200.0
product	5 562	11.7	7 747	15.7	7 853	15.9
robust_100_0	86 362	1307.3	79 523	1234.3	79 523	1245.8
robust_100_1	13 780	207.9	16 517	235.9	16 517	239.9
robust_200_0	>139 784	>7200.0	>74 872	>7200.0	>73 339	>7200.0
robust_20_0	8	0.5	8	0.5	8	0.5
robust_50_0	91	1.4	91	1.8	91	1.8
robust_50_1	228	3.0	200	2.8	200	2.8
shortfall_100_0	>495 750	>7200.0	>497 757	>7200.0	>503 010	>7200.0
shortfall_100_1	356 687	3926.5	311 239	3382.3	226 505	2414.0
shortfall_200_0	>104 110	>7200.0	>103 692	>7200.0	>103 523	>7200.0
shortfall_20_0	102	0.8	120	0.9	120	0.8
shortfall_50_0	343 829	1738.6	695 205	3628.8	690 262	3615.6
shortfall_50_1	9 259	43.2	11 106	46.0	11 106	47.4
SLay05H	254	2.1	75	1.6	75	1.6
SLay05M	79	0.6	150	1.0	150	1.0
SLay07M	1 930	6.9	377	3.0	377	3.1

Table B.14 continued

	No RENS		Root RENS		Tree RENS	
Instance	Nodes	Time	Nodes	Time	Nodes	Time
SLay10H	>532 368	>7200.0	>532 759	>7200.0	>498 710	>7200.0
SLay10M	229 809	1828.4	28 848	233.2	28 856	241.4
space25a	>21 026	>7200.0	>21 026	>7200.0	>21 026	>7200.0
space25	>8 751	>7200.0	>8 751	>7200.0	>8 751	>7200.0
spectra2	33	0.7	23	0.7	23	0.8
tln12	>2 590 652	>7200.0	>2 587 580	>7200.0	>2 589 049	>7200.0
tln5	44 527	26.2	44 527	26.1	44 527	26.3
tln6	>12 370 474	>7200.0	>12 372 692	>7200.0	>12 367 087	>7200.0
tln7	>9 474 819	>7200.0	>9 482 513	>7200.0	>9 493 095	>7200.0
tloss	60	0.5	60	0.5	60	0.5
tltr	24	0.5	24	0.5	24	0.5
uflquad-15-60	904	2857.7	904	2862.1	827	2491.9
uflquad-20-50	>201	>7200.0	>201	>7200.0	>34	>7200.0
uflquad-40-80	>105	>7200.0	>105	>7200.0	>39	>7200.0
util	371	0.5	375	0.5	375	0.5
waste	>4 005 594	>7200.0	>3 983 731	>7200.0	>3 964 173	>7200.0
arithm. mean	659 740	2872.3	677 123	2927.0	664 117	2888.6
sh. geom. mean	6 457	229.9	6 361	232.0	6 193	229.9

Table B.15.: Impact of RENS on overall solving process for MINLP instances

	No RENS		Root RENS		Tree RENS	
Instance	Nodes	Time	Nodes	Time	Nodes	Time
beuster	>243	>7200.0	>243	>7200.0	>243	>7200.0
cecil_13	>2 557 284	>7200.0	>2 553 413	>7200.0	>2 568 736	>7200.0
contvar	>10 024	>7200.0	>10 024	>7200.0	>10 024	>7200.0
csched1	44 649	17.2	44 649	17.5	44 649	17.6
csched2a	>26 250	>7200.0	>26 250	>7200.0	>26 250	>7200.0
detf1	>331	>7200.0	>330	>7200.0	>331	>7200.0
eg_all_s	>446	>7200.0	>440	>7200.0	>440	>7200.0
eg_disc2_s	>83	>7200.0	>83	>7200.0	>48	>7200.0
eg_disc_s	>136	>7200.0	>136	>7200.0	>34	>7200.0
eg_int_s	>5	>7200.0	>5	>7200.0	>5	>7200.0
eniplac	172	0.7	172	0.6	98	0.6
enpro48	84	0.8	54 982	11.9	12 571	4.3
enpro48pb	249 160	42.9	36	0.9	36	0.8
enpro56pb	4 048	1.8	85 265	17.6	85 265	17.6
ex1233	>11 127 294	>7200.0	>11 141 457	>7200.0	>11 144 945	>7200.0
ex1244	492	1.0	492	1.1	504	1.4
ex1252	>88	>7200.0	>88	>7200.0	>88	>7200.0
ex1252a	>204	>7200.0	>204	>7200.0	>204	>7200.0
feedtray	>640 421	>7200.0	>638 931	>7200.0	>639 220	>7200.0
fo7	163 542	68.1	163 542	67.8	163 542	68.6
fo7_2	45 627	22.2	45 627	22.2	48 697	23.8
fo7_ar25_1	43 715	16.9	43 715	17.3	49 960	19.5
fo7_ar2_1	39 986	17.3	39 986	17.3	39 986	17.6
fo7_ar3_1	47 741	17.9	47 741	17.9	50 563	19.5
fo7_ar4_1	58 884	28.5	58 884	28.2	58 884	29.2
fo7_ar5_1	20 509	9.1	20 509	9.0	20 509	9.1
fo8	538 828	277.3	538 828	277.3	538 828	279.3
fo8_ar25_1	337 708	141.8	337 708	141.4	149 658	59.9
fo8_ar2_1	643 114	168.7	643 114	168.6	192 277	75.0
fo8_ar3_1	75 943	43.8	75 943	43.8	75 943	44.6
fo8_ar4_1	>46 231 801	>7200.0	>46 093 488	>7200.0	86 646	43.3
fo8_ar5_1	55 953	27.9	55 953	28.4	55 953	29.2
fo9	2 155 434	1140.4	2 155 434	1143.8	10 127 873	2879.5
fo9_ar25_1	4 702 715	1731.2	4 702 715	1733.8	4 881 081	1843.4
fo9_ar2_1	2 615 019	1089.9	2 615 019	1092.5	2 615 019	1092.2
fo9_ar3_1	532 025	284.5	532 025	284.9	331 077	172.6
fo9_ar4_1	284 985	133.1	284 985	134.6	284 985	133.7
fo9_ar5_1	729 300	405.2	729 300	408.4	729 300	409.3
fuzzy	>2 161 178	>7200.0	>2 156 389	>7200.0	408 344	1883.2
gasnet	>1 382	>7200.0	>1 382	>7200.0	>1 382	>7200.0
gear	2 828	2.0	2 828	2.0	2 828	2.0
gear2	591	0.5	506	0.5	506	0.5
gear3	2 828	2.2	2 828	2.1	2 828	2.0
gear4	105	0.5	105	0.5	105	0.5
ghg_1veh	>18 013 454	>7200.0	>18 137 988	>7200.0	>18 188 182	>7200.0
ghg_2veh	>737 048	>7200.0	>87 992	>7200.0	>853 625	>7200.0
ghg_3veh	>420 745	>7200.0	>420 693	>7200.0	>211 106	>7200.0
hda	>848 500	>7200.0	>847 241	>7200.0	>824 623	>7200.0
johnall	1	64.0	1	72.3	1	63.8
m6	955	1.1	955	1.0	955	1.2
m7	14 053	6.5	14 053	6.4	14 053	6.6
m7_ar25_1	2 848	2.0	2 848	2.1	2 055	1.4
m7_ar2_1	22 707	5.7	22 707	5.6	22 707	5.8
m7_ar3_1	9 390	4.6	9 390	4.5	9 390	4.6
m7_ar4_1	2 134	1.8	2 134	1.8	2 134	2.1
m7_ar5_1	25 814	6.8	25 814	6.9	25 814	7.2
no7_ar25_1	107 048	51.4	107 048	50.7	87 297	42.4

Table B.15 continued

	No RENS		Root RENS		Tree RENS	
Instance	Nodes	Time	Nodes	Time	Nodes	Time
no7_ar2_1	27 667	14.9	27 667	14.8	27 667	14.9
no7_ar3_1	423 874	187.2	423 874	185.8	423 874	186.9
no7_ar4_1	228 710	108.6	228 710	108.3	252 173	120.5
no7_ar5_1	103 053	52.5	103 053	52.2	103 053	52.0
nvs09	>4 697 821	>7200.0	>6 241 826	>7200.0	>6 342 072	>7200.0
nvs20	355	0.8	355	0.8	355	1.0
o7	4 566 673	2343.0	4 566 673	2345.9	4 566 673	2357.2
o7_2	1 730 061	756.5	1 730 061	754.7	1 708 453	755.9
o7_ar25_1	489 625	241.3	489 625	239.7	489 625	244.1
o7_ar2_1	176 585	88.0	176 585	86.2	151 581	69.9
o7_ar3_1	1 230 419	616.6	1 230 419	616.7	1 230 419	618.9
o7_ar4_1	1 854 132	991.8	1 854 132	994.0	1 854 132	994.5
o7_ar5_1	795 136	371.7	795 136	372.3	613 092	282.3
o8_ar4_1	11 782 816	6666.4	11 782 816	6688.3	12 722 339	6984.3
o9_ar4_1	>12 507 230	>7200.0	>12 514 424	>7200.0	>12 415 746	>7200.0
oil	>589 974	>7200.0	>589 231	>7200.0	>589 208	>7200.0
oil2	>1 027 176	>7200.0	>1 028 096	>7200.0	>1 024 608	>7200.0
parallel	735 814	2599.6	735 814	2592.5	735 814	2591.3
pump	>47	>7200.0	>47	>7200.0	>47	>7200.0
risk2b	2	0.6	2	0.6	2	0.6
saa_2	>331	>7200.0	>331	>7200.0	>331	>7200.0
spring	90	0.5	90	0.5	90	0.5
st_e32	12 153	13.6	12 153	13.7	12 153	13.6
stockcycle	32 340	222.0	32 340	222.2	32 340	223.2
super1	>88 353	>7200.0	>88 400	>7200.0	>88 430	>7200.0
super2	>90 554	>7200.0	>89 681	>7200.0	>90 164	>7200.0
super3	>102 297	>7200.0	>100 310	>7200.0	>102 024	>7200.0
super3t	>71 449	>7200.0	>71 272	>7200.0	>68 820	>7200.0
synheat	>68 710	>7200.0	>68 710	>7200.0	>68 710	>7200.0
synthes1	4	0.5	4	0.5	4	0.5
synthes2	5	0.5	4	0.5	4	0.5
synthes3	>56 469 781	>7200.0	>54 499 711	>7200.0	>57 219 056	>7200.0
tls12	>622 812	>7200.0	>629 179	>7200.0	>628 973	>7200.0
tls4	9 520	11.7	12 723	13.4	12 723	13.5
tls5	>3 950 998	>7200.0	>3 941 413	>7200.0	>3 943 467	>7200.0
tls6	>2 741 985	>7200.0	>2 729 799	>7200.0	>2 732 632	>7200.0
tls7	>1 805 765	>7200.0	>1 797 325	>7200.0	>1 804 162	>7200.0
water3	>6 706 261	>7200.0	>6 698 169	>7200.0	>6 578 939	>7200.0
water4	1 692 444	1860.5	1 692 444	1863.9	1 642 038	1816.3
waterful2	>4 169 416	>7200.0	>4 164 237	>7200.0	>4 148 024	>7200.0
watersbp	>4 032 620	>7200.0	>4 032 620	>7200.0	>155 142	>7200.0
watersym1	>6 705 227	>7200.0	>6 453 837	>7200.0	>6 730 378	>7200.0
watersym2	>8 127 217	>7200.0	>8 123 253	>7200.0	>8 059 966	>7200.0
waterx	>1 425	>7200.0	>1 425	>7200.0	>1 425	>7200.0
waterz	>1 094 883	>7200.0	>1 094 883	>7200.0	>1 094 883	>7200.0
arithm. mean	2 338 903	3274.5	2 324 208	3274.7	1 925 902	3168.7
sh. geom. mean	58 758	466.5	58 406	467.1	51 066	431.3

Table B.16.: Cover sizes and primal solution values attained by UC and MINLP solvers on MIQCP test set.

	% cov	% nlcov	UC	SCIP 2.1.1	COUENNE 0.3	BARON 9.3.1	BONMIN 1.6
CLay0203M	22.22	100.00	–	41573.262	–	54581.749	41986.253
CLay0204M	16.67	100.00	–	9199.9953	–	–	55601.563
CLay0205M	13.33	100.00	–	81612.088	–	–	8688.4286
CLay0303M	19.35	100.00	–	–	–	–	56141.526
CLay0304M	14.81	100.00	–	78552.626	–	–	–
CLay0305M	12.35	100.00	–	70332.768	–	–	11136.861
SLay04H	5.67	100.00	14395.62	9975.6616	13338.483	12013.906	12013.906
SLay04M	17.78	100.00	14395.62	12544.861	13241.081	9859.6597	12013.906
SLay05H	4.33	100.00	56836.232	24998.521	30419.804	–	30158.377
SLay05M	14.08	100.00	56836.232	27119.518	30286.966	–	35512.884
SLay06H	3.50	100.00	78418.037	135525.52	40761.755	–	44660.49
SLay06M	11.65	100.00	99393.821	42920.398	40225.601	–	46473.124
SLay07H	2.94	100.00	–	266528.05	105472.68	–	100329.71
SLay07M	9.93	100.00	157243.3	99366.754	105417.85	96289.867	112362.73
SLay08H	2.53	100.00	–	370075	131525.48	–	178522.98
SLay08M	8.65	100.00	458483.51	102746.46	143095.48	–	190574.08
SLay09H	2.22	100.00	–	152428.01	162397.71	–	188685.63
SLay09M	7.66	100.00	–	135783.77	184450.78	344702.34	205678.96
SLay10H	1.98	100.00	–	577942.5	216914.82	–	462524.34
SLay10M	6.87	100.00	–	144233.42	259159.88	–	352953.66
LeeCrudeOil1_05	8.25	25.00	–	–	–	–	-79.75
LeeCrudeOil1_06	7.97	25.00	–	–	–	–	-78.75
LeeCrudeOil1_07	7.79	25.00	–	–	–	–	–
LeeCrudeOil1_08	7.67	25.00	–	–	–	–	–
LeeCrudeOil1_09	7.58	25.00	–	–	–	–	-79.75
LeeCrudeOil1_10	7.52	25.00	–	–	–	–	–
LeeCrudeOil2_05	7.81	22.41	–	–	–	–	–
LeeCrudeOil2_07	7.29	22.47	–	–	–	–	–
LeeCrudeOil2_09	7.10	22.50	–	–	–	–	–
LeeCrudeOil2_10	7.04	22.51	–	–	–	–	–
LeeCrudeOil3_05	11.79	23.85	–	–	–	–	-81.314018

Table B.16 continued

	% cov	% nlcov	UC	SCIP 2.1.1	COUENNE 0.3	BARON 9.3.1	BONMIN 1.6
LeeCrudeOil3_06	11.66	23.37	–	–	–	–	–
LeeCrudeOil3_07	11.57	23.08	–	–	–	–	–
LeeCrudeOil3_08	11.51	22.88	–	–	–	–	-78.5
LeeCrudeOil3_09	11.45	22.75	–	–	–	–	–
LeeCrudeOil3_10	11.41	22.64	–	–	–	–	–
LeeCrudeOil4_06	6.57	21.62	–	-132.53069	–	–	–
LeeCrudeOil4_07	6.37	21.52	–	-132.16	–	–	–
LeeCrudeOil4_08	6.24	21.46	–	-132.47831	–	–	–
LeeCrudeOil4_09	6.15	21.40	–	-131.96016	–	–	–
LeeCrudeOil4_10	6.08	21.36	–	–	–	–	–
LiCrudeOil_ex01	9.69	46.67	–	–	–	–	5102.1808
LiCrudeOil_ex02	0.95	29.41	–	64752228	–	–	32923042
LiCrudeOil_ex03	5.72	33.33	–	–	–	–	–
LiCrudeOil_ex05	6.73	33.33	–	–	–	–	–
LiCrudeOil_ex06	5.72	33.33	–	–	–	–	2979.375
LiCrudeOil_ex11	5.24	33.33	–	–	–	–	–
LiCrudeOil_ex21	4.70	33.33	–	–	–	–	–
alan	33.33	100.00	3.6	3	2.925	2.925	3.6
du-opt5	94.74	100.00	546.27998	15.621774	157.08224	–	1531.95
du-opt	95.24	100.00	632.89142	4.9046426	3.9051553	3725.6208	1758.1078
elf	5.56	50.00	1.675	0.32799999	–	1.675	1.9771427
ex1223a	33.33	100.00	6.5574613	4.5795817	4.5795824	4.5795824	4.5795824
ex1263a	17.39	20.00	30.1	29.6	–	–	–
ex1263	4.40	20.00	30.1	–	–	–	–
ex1264a	17.39	20.00	11.1	11.1	–	–	–
ex1264	4.88	20.00	11.1	–	–	–	–
ex1265a	14.71	16.67	15.1	15.1	–	–	–
ex1265	4.10	16.67	15.1	–	–	–	–
ex1266a	13.04	14.29	16.3	–	–	–	–
ex1266	3.57	14.29	16.3	–	–	–	–
ex4	13.51	100.00	-7.7891291	-8.0641362	289199.88	-7.7891291	-7.3132691
fac3	80.60	100.00	31995144	32039523	–	–	1.3065386e+08
feedtray2	4.67	50.00	–	0	–	0	1.5447167e-09
fuel	46.15	100.00	10286.143	11925	–	8566.119	8566.119

Table B.16 continued

	% cov	% nlcov	UC	SCIP 2.1.1	COUENNE 0.3	BARON 9.3.1	BONMIN 1.6
meanvarx	23.33	100.00	14.824808	14.369212	14.404062	14.369232	21.110398
netmod_dol1	0.30	100.00	0	-0.372303	–	-4.4408921e-15	4.9805856e-09
netmod_dol2	0.38	100.00	0	-1.012723e-09	–	0.057164114	4.6888223e-08
netmod_kar1	0.88	100.00	0	-1.0012411e-09	–	1.6653345e-15	2.6219098e-09
netmod_kar2	0.88	100.00	0	-1.0012411e-09	–	1.6653345e-15	2.6219098e-09
nous1	29.79	36.84	–	–	1.567072	1.567072	1.567072
nous2	30.43	36.84	–	1.3843163	0.62596741	0.62596741	0.62596741
nuclear14a	12.24	48.98	–	-1.1007655	–	–	-1.1286438
nuclear14b	12.24	48.98	–	-1.0976863	–	–	–
nuclear24a	12.24	48.98	–	-1.1007655	–	–	-1.1286438
nuclear24b	12.24	48.98	–	-1.0976863	–	–	–
nuclear25a	11.88	49.02	–	-1.0571308	–	–	-1.0890967
nuclear25b	11.88	49.02	–	–	–	–	–
nvs03	66.67	100.00	16	16	16	68	–
nvs10	66.67	100.00	-252	-310.8	–	-310.8	-310.8
nvs11	75.00	100.00	-270.8	-431	-431	-431	-431
nvs12	80.00	100.00	-358.4	-481.2	–	-481.2	-481.2
nvs13	83.33	100.00	-172	-580.4	–	-582.8	-585.2
nvs14	33.33	60.00	-39886.664	-40358.155	-40358.155	–	–
nvs15	40.00	100.00	1	1	1	1	3
nvs17	87.50	100.00	0	-1098.6	–	-1098.6	-1100.4
nvs18	85.71	100.00	-106.8	-777	-678.4	-776.6	-778.4
nvs19	88.89	100.00	0	-1097.8	–	-1098	–
nvs23	90.00	100.00	484.2	-1122.2	–	-1118	-1113.8
nvs24	90.91	100.00	–	-1028.8	–	-1025.8	-1031.8
prob02	16.67	16.67	792000	112235	112235	112235	112235
prob03	50.00	50.00	10	10	10	10	10
product2	23.80	100.00	–	-2102.3771	–	–	-2093.4479
product	36.22	100.00	–	-2094.688	–	–	-1868.5446
eniplac_reformulated	19.51	100.00	–	–	–	-128473.32	-129658.81
fo7_2_reformulated	8.54	50.00	–	–	–	–	31.980873
fo7_reformulated	8.54	50.00	–	–	–	–	28.335082
fo8_reformulated	7.84	50.00	–	–	–	–	40.789936
fo9_reformulated	7.26	50.00	–	–	–	–	59.305615

Table B.16 continued

	% cov	% nlcov	UC	SCIP 2.1.1	COUENNE 0.3	BARON 9.3.1	BONMIN 1.6
m3_reformulated	13.64	50.00	37.8	46.306314	67.8	37.8	49.8
m6_reformulated	9.68	50.00	–	–	–	–	106.25688
m7_reformulated	8.75	50.00	–	–	–	–	130.75688
o7_2_reformulated	7.78	50.00	–	–	–	–	167.68058
o7_reformulated	7.78	50.00	–	–	–	–	189.66429
sep1	10.53	40.00	-510.08098	-470.13009	-510.08098	-510.08098	-510.08098
space25a	5.84	41.86	–	–	–	–	487.07433
space25	1.04	30.77	–	–	–	–	487.07433
space960	27.74	43.43	–	17130000	–	42155000	–
spectra2	44.12	100.00	306.3343	13.978303	–	28.143456	306.3343
st_e13	50.00	100.00	2	2	2	2	2
st_e27	40.00	100.00	9	2	2	2	2
st_e31	3.39	40.00	-2.0000015	–	–	–	-2
st_miqp2	40.00	100.00	2	2	–	2	–
st_miqp3	50.00	100.00	-6	-6	-6	-6	-6
st_miqp4	50.00	100.00	-4574	-4574	-4574	-4574	-4574
st_miqp5	25.00	100.00	-333.88891	-333.88891	-333.88889	-333.88889	-333.88889
st_test4	28.57	100.00	-7	-7	-7	-7	–
st_test8	96.00	100.00	-26041	-29605	-29575	-29605	-29605
st_testgr1	90.91	100.00	-6.688	-12.79955	-12.7842	-12.7392	-7.713
st_testgr3	95.24	100.00	-20.27475	-20.5795	-20.49105	-20.47635	-20.0796
st_testph4	75.00	100.00	-56	-80.5	-80.5	-80.5	-80.5
tln12	6.67	8.33	–	–	–	–	–
tln2	33.33	33.33	17.3	5.3	5.3	–	–
tln4	16.67	20.00	11.1	12.4	–	–	–
tln5	14.29	16.67	15.1	15.5	–	–	11
tln6	12.50	14.29	32.3	–	–	–	–
tln7	11.11	12.50	30.3	–	–	–	–
tloss	13.04	14.29	16.3	–	–	–	–
tltr	16.07	33.33	61.133333	83.475	–	–	–
util	3.12	16.67	999.69056	1005.2681	–	1000.0498	999.57875
waste	2.50	13.78	626.89124	692.98377	–	710.352	1011.5257
Sarawak_Scenario16	4.97	19.05	-31479.405	-31868.099	-31921.57	-31409.405	–
Sarawak_Scenario1	4.04	19.05	-32435.405	-31115.543	-27840.759	-32399.405	-22605.074

Table B.16 continued

	% cov	% nlcov	UC	SCIP 2.1.1	COUENNE 0.3	BARON 9.3.1	BONMIN 1.6
Sarawak_Scenario81	5.03	19.05	-31479.405	-31865.355	-30515.443	-31409.405	–
lee1	16.33	40.00	–	–	–	-4640.0824	-4296.9511
lee2	22.64	50.00	–	–	–	–	–
meyer04	10.17	42.86	–	–	–	–	1422175.2
meyer10	7.61	23.08	–	–	–	3698168.3	–
meyer15	6.13	16.67	–	–	–	–	1008046.4
ahmetovic1_pw4	2.40	35.00	–	–	–	–	606466.42
ahmetovic2_pw4	2.09	28.57	–	–	–	–	1217509.3
karuppiah1	27.59	34.78	–	139.32508	–	117.05263	117.45263
karuppiah2_pw4	14.74	26.42	–	381396.6	490999.74	480435.29	–
karuppiah3_pw4	17.72	28.57	–	1753698.3	1753698.3	–	–
karuppiah4_pw4	18.57	27.96	–	1430067.5	2376436.2	–	1046406.2
ruiz_concbased_pw4	13.33	36.36	–	414748.31	–	–	–
ruiz_flowbased_pw4	8.33	45.45	–	–	–	346345.39	–

Table B.17.: Comparison of overall performance of SCIP 2.1.1 with and without Undercover on MIQCP test set. Columns nodes and time show the number of branch-and-bound nodes and the running time needed to solve an instance to proven optimality, respectively. Column pb root depicts the primal bound after the root node.

	SCIP + UC			SCIP − UC		
	nodes	time [s]	pb root	nodes	time [s]	pb root
CLay0203M	48	0.1	41572.98	48	0.2	41572.98
CLay0204M	661	0.7	9199.995	721	0.7	9199.995
CLay0205M	10 690	4.2	81611.33	9 655	3.9	81611.33
CLay0303M	87	0.1	–	87	0.1	–
CLay0304M	316	0.6	78552.09	298	0.6	78552.09
CLay0305M	9 205	3.9	70332.47	8 969	4.1	70332.47
SLay04H	31	0.6	14395.62	31	0.3	9975.662
SLay04M	71	0.6	11676.06	132	0.8	12544.86
SLay05H	288	2.5	24998.52	286	2.3	24998.52
SLay05M	24	0.6	25589.95	56	0.7	27119.52
SLay06H	992	5.1	135525.5	1 670	8.6	135525.5
SLay06M	266	1.4	41921.2	618	2.0	42920.4
SLay07H	5 406	66.9	266528.1	5 895	71.9	266528.1
SLay07M	730	3.8	71077.43	1 430	9.6	99366.75
SLay08H	4 769	61.2	370075	32 232	310.3	370075
SLay08M	1 079	5.8	102746.5	1 493	7.0	102746.5
SLay09H	6 971	107.8	152428	31 680	383.1	152428
SLay09M	3 561	24.8	136774.1	1 453	22.4	135783.8
SLay10H	>212 055	>3600.0	577942.5	144 350	2144.1	577942.5
SLay10M	27 922	181.9	144233.4	170 975	1034.2	144233.4
LeeCrudeOil1_05	25	1.0	–	13	0.8	–
LeeCrudeOil1_06	14	1.3	–	27	1.5	–
LeeCrudeOil1_07	29	1.5	–	29	1.4	–
LeeCrudeOil1_08	40	4.3	–	39	4.3	–
LeeCrudeOil1_09	62	3.9	–	108	4.4	–
LeeCrudeOil1_10	141	7.4	–	179	8.6	–
LeeCrudeOil2_05	32	2.1	–	80	2.1	–
LeeCrudeOil2_06	21	3.5	-101.1746	46	3.3	-101.1746
LeeCrudeOil2_07	397	6.5	–	384	6.2	–
LeeCrudeOil2_08	261	7.3	-101.1738	371	8.0	-101.1738
LeeCrudeOil2_09	713	17.5	–	517	16.5	–
LeeCrudeOil2_10	672	20.5	–	682	30.0	–
LeeCrudeOil3_05	2 141	7.3	–	6 821	18.3	–
LeeCrudeOil3_06	14 851	53.1	–	21 515	65.0	–
LeeCrudeOil3_07	20 341	77.2	–	28 851	95.2	–
LeeCrudeOil3_08	52 781	259.0	–	32 411	160.6	–
LeeCrudeOil3_09	48 118	289.9	–	51 121	270.9	–
LeeCrudeOil3_10	41 941	308.5	–	37 141	264.0	–
LeeCrudeOil4_05	106	2.5	–	23	3.3	–
LeeCrudeOil4_06	20	3.5	-132.5307	16	4.5	-132.5307
LeeCrudeOil4_07	118	5.7	-132.16	21	5.8	-132.16
LeeCrudeOil4_08	67	8.8	-132.4783	212	15.0	-132.4783
LeeCrudeOil4_09	43	15.5	-131.9602	28	11.9	-131.9602
LeeCrudeOil4_10	419	20.8	–	157	21.9	–
LiCrudeOil_ex01	>1 318 676	>3600.0	–	>1 178 319	>3600.0	–
LiCrudeOil_ex02	>1 096 681	>3600.0	64752230	>1 074 998	>3600.0	64752230
LiCrudeOil_ex03	>285 396	>3600.0	–	>307 358	>3600.0	–
LiCrudeOil_ex05	>375 180	>3600.0	–	>408 032	>3600.0	–

Table B.17 continued

	SCIP + UC			SCIP − UC		
	nodes	time [s]	pb root	nodes	time [s]	pb root
LiCrudeOil_ex06	19 790	313.8	–	60 296	805.1	–
LiCrudeOil_ex11	>269 067	>3600.0	–	>271 844	>3600.0	–
LiCrudeOil_ex21	>232 969	>3600.0	–	>242 431	>3600.0	–
alan	6	0.1	2.924996	6	0.1	2.924996
du-opt5	80	0.5	13.60875	58	0.4	15.62177
du-opt	238	0.7	7.246512	162	0.6	4.904643
elf	293	0.3	0.328	293	0.4	0.328
ex1223a	1	0.1	4.579582	1	0.0	4.579582
ex1263a	229	0.2	29.3	126	0.2	29.6
ex1263	596	0.7	30.1	194	0.4	–
ex1264a	176	0.2	10.3	128	0.1	11.1
ex1264	86	0.2	11.1	179	0.2	–
ex1265a	72	0.1	14.3	70	0.1	15.1
ex1265	69	0.3	11.3	186	0.4	–
ex1266a	1	0.0	16.3	397	0.5	–
ex1266	1	0.1	16.3	209	0.6	–
ex4	11	0.7	-8.064135	11	0.8	-8.064136
fac3	8	0.2	31995140	12	0.1	32039520
feedtray2	1	0.1	0	1	0.1	0
fuel	3	0.1	10286.14	5	0.1	11925
gbd	1	0.0	2.2	1	0.0	2.2
meanvarx	4	0.1	14.36921	4	0.1	14.36921
netmod_dol1	>42 355	>3600.0	-0.3740157	>40 552	>3600.0	-0.372303
netmod_dol2	793	71.7	0	80	33.4	0
netmod_kar1	315	4.5	-0.3717949	279	3.7	0
netmod_kar2	315	4.5	-0.3717949	279	3.7	0
nous1	>2 196 718	>3600.0	–	>2 174 442	>3600.0	–
nous2	3 311	2.9	1.384316	4 764	3.4	1.384316
nuclear104	>62 874	>3600.0	–	>66 256	>3600.0	–
nuclear10a	>49	>3600.0	–	>43	>3600.0	–
nuclear10b	>1	>3600.0	–	>1	>3600.0	–
nuclear14a	>62 466	>3600.0	-1.111458	>57 760	>3600.0	-1.100766
nuclear14b	>47 549	>3600.0	-1.097686	>47 568	>3600.0	-1.097686
nuclear14	>1 473 004	>3600.0	–	>1 471 465	>3600.0	–
nuclear24a	>62 343	>3600.0	-1.111458	>57 760	>3600.0	-1.100766
nuclear24b	>47 556	>3600.0	-1.097686	>47 482	>3600.0	-1.097686
nuclear24	>1 474 515	>3600.0	–	>1 463 971	>3600.0	–
nuclear25a	>55 186	>3600.0	-1.057131	>49 835	>3600.0	-1.057131
nuclear25b	>33 525	>3600.0	–	>35 924	>3600.0	–
nuclear25	>1 380 060	>3600.0	–	>1 374 983	>3600.0	–
nuclear49a	>5 883	>3600.0	–	>6 729	>3600.0	–
nuclear49b	>2 920	>3600.0	–	>3 032	>3600.0	–
nuclear49	>379 308	>3600.0	–	>378 649	>3600.0	–
nuclearva	>2 988 726	>3600.0	–	>2 978 098	>3600.0	–
nuclearvb	>3 004 258	>3600.0	–	>3 005 785	>3600.0	–
nuclearvc	>2 983 650	>3600.0	–	>3 002 087	>3600.0	–
nuclearvd	>2 719 871	>3600.0	–	>2 715 102	>3600.0	–
nuclearve	>2 735 734	>3600.0	–	>2 732 253	>3600.0	–
nuclearvf	>2 740 055	>3600.0	–	>2 743 820	>3600.0	–
nvs03	1	0.0	16	1	0.0	16
nvs10	1	0.0	-310.8	1	0.0	-310.8
nvs11	3	0.0	-431	3	0.0	-431
nvs12	6	0.1	-481.2	5	0.0	-481.2
nvs13	12	0.1	-580.4	9	0.1	-580.4
nvs14	1	0.0	-40358.15	1	0.0	-40358.15
nvs15	4	0.1	1	5	0.0	1
nvs17	51	0.1	-1098.6	45	0.1	-1098.6
nvs18	23	0.1	-777	20	0.1	-777
nvs19	89	0.2	-1097.8	84	0.2	-1097.8

Table B.17 continued

	SCIP + UC			SCIP − UC		
	nodes	time [s]	pb root	nodes	time [s]	pb root
nvs23	106	0.3	-1124.2	103	0.3	-1122.2
nvs24	104	0.3	-1028.8	103	0.3	-1028.8
prob02	1	0.0	112235	1	0.0	112235
prob03	1	0.0	10	1	0.0	10
product2	>3 258 756	>3600.0	-2099.124	>3 311 596	>3600.0	-2102.377
product	7 317	21.1	-2094.688	7 989	23.3	-2094.688
iplac_reformulated	282	0.7	–	282	0.8	–
fo7_2_reformulated	57 203	33.5	–	62 818	35.6	–
fo7_reformulated	185 976	108.2	–	210 033	125.2	–
fo8_reformulated	364 808	230.3	–	426 135	250.6	–
fo9_reformulated	1 689 569	1130.6	–	2 775 675	1831.0	–
m3_reformulated	14	0.2	37.8	21	0.1	46.30631
m6_reformulated	8 958	4.1	–	1 601	1.5	–
m7_reformulated	4 635	3.6	–	5 988	4.2	–
o7_2_reformulated	1 461 823	773.7	–	1 501 419	790.7	–
o7_reformulated	3 647 967	2124.1	–	3 838 657	2191.2	–
sep1	37	0.3	-510.081	47	0.2	-470.1301
space25a	>339 329	>3600.0	–	>188 127	>3600.0	–
space25	>6 611	>3600.0	–	>70 227	>3600.0	–
space960	>3 760	>3600.0	17130000	>3 479	>3600.0	17130000
spectra2	19	0.8	13.9783	23	0.6	13.9783
st_e13	1	0.0	0	1	0.0	0
st_e27	1	0.0	2	1	0.0	2
st_e31	1 647	1.0	-2.000001	2 038	1.0	–
st_miqp1	1	0.0	281	1	0.0	281
st_miqp2	1	0.0	2	1	0.0	2
st_miqp3	1	0.0	-6	1	0.0	-6
st_miqp4	1	0.1	-4574	1	0.0	-4574
st_miqp5	1	0.1	-333.8889	1	0.0	-333.8889
st_test1	1	0.0	0	1	0.0	0
st_test2	1	0.0	-9.25	1	0.0	-9.25
st_test3	1	0.0	-7	1	0.0	-7
st_test4	1	0.0	-7	1	0.0	-7
st_test5	1	0.0	-110	1	0.0	-110
st_test6	1	0.0	471	1	0.0	471
st_test8	1	0.0	-29605	1	0.0	-29605
st_testgr1	48	0.1	-12.79955	20	0.1	-12.79955
st_testgr3	28	0.1	-20.5795	23	0.1	-20.5795
st_testph4	1	0.0	-80.5	1	0.0	-80.5
tln12	>1 549 104	>3600.0	–	>1 481 337	>3600.0	–
tln2	1	0.0	5.3	1	0.0	5.3
tln4	2 658	1.5	11.1	2 784	1.5	12.4
tln5	171 037	105.1	15.1	104 002	63.1	15.5
tln6	>5 322 038	>3600.0	32.3	>5 432 291	>3600.0	–
tln7	>3 010 846	>3600.0	30.3	>3 179 741	>3600.0	–
tloss	1	0.0	16.3	145	0.2	–
tltr	38	0.2	61.13333	94	0.2	83.475
util	7	0.3	999.6906	213	0.4	1005.268
waste	>2 080 248	>3600.0	621.8648	>2 068 008	>3600.0	692.9838
Sarawak_Scenario16	>706 322	>3600.0	-31868.1	>668 385	>3600.0	-31868.1
Sarawak_Scenario1	502	1.1	-32435.4	541	1.4	-31115.54
Sarawak_Scenario81	>152 074	>3600.0	-31865.36	>153 286	>3600.0	-31865.36
lee1	2 828	2.2	–	21 451	19.3	–
lee2	37 584	44.8	–	31 580	36.9	–
meyer04	>3 106 891	>3600.0	–	>3 047 664	>3600.0	–
meyer10	>1 391 651	>3600.0	–	>1 338 247	>3600.0	–
meyer15	>183 963	>3600.0	–	>358 139	>3600.0	–
ahmetovic1_pw4	42 842	36.1	–	63 195	58.2	–
ahmetovic2_pw4	>684 192	>3600.0	–	>640 108	>3600.0	–

Table B.17 continued

	SCIP + UC			SCIP − UC		
	nodes	time [s]	pb root	nodes	time [s]	pb root
karuppiah1	1 941	1.8	139.3251	1 031	1.1	139.3251
karuppiah2_pw4	>4 650 797	>3600.0	381396.6	>4 299 998	>3600.0	381396.6
karuppiah3_pw4	34 191	23.0	1753698	61 191	33.9	1753698
karuppiah4_pw4	>1 198 193	>3600.0	1430067	>1 110 591	>3600.0	1430067
ruiz_concbased_pw4	8 511	8.3	414748.3	27 271	22.7	414748.3

Table B.18.: Rapid Learning results for pure BPs and IPs from Miplib 3, Miplib 2003 and Benchmark test set of Miplib 2010

	SCIP def		SCIP RL		Rapid Learning			
	Nodes	Time	Nodes	Time	RL Time	Ngds	Bds	Sol
10teams	512	18.1	725	16.6	1.28	412	–	
acc-tight5	562	121.2	790	123.5	2.14	2200	–	
air03	1	1.6	1	1.6	0.12	–	–	
air04	113	55.7	43	48.9	4.47	1277	11	
air05	190	35.7	320	46.1	4.60	1608	5	
cap6000	3568	2.7	3100	3.3	0.87	270	–	
cov1075	1635419	limit	1623039	limit	0.14	–	–	
eil33-2	10315	73.5	11608	77.0	0.50	–	–	✓
eilB101	20077	452.0	26896	511.1	0.46	–	–	
enigma	1697	0.7	45	0.3	0.05	34	–	
fiber	18	1.6	13	1.4	0.17	102	–	
harp2	12977499	4632.7	9652824	2915.6	0.16	201	–	
iis-100-0-cov	101535	1640.2	106329	1768.7	0.11	–	–	
iis-bupa-cov	192310	limit	190939	limit	0.21	–	–	
iis-pima-cov	7213	665.3	8408	732.0	0.26	–	–	
l152lav	64	3.7	90	4.1	0.81	229	–	
lseu	454	0.6	270	0.5	0.06	205	–	
m100n500k4r1	7977796	limit	7729744	6826.4	0.04	–	–	
macrophage	1634984	limit	1569262	limit	0.09	–	–	
markshare1	80628220	limit	65900646	limit	0.06	187	–	✓
markshare2	71284947	limit	68006924	limit	0.05	–	–	
mcsched	21927	267.7	19173	250.5	0.24	–	–	
mine-166-5	2596	42.6	2448	39.7	0.28	–	–	
mine-90-10	243075	1182.3	157461	662.0	0.59	5	4	✓
misc03	118	1.2	71	1.3	0.08	255	–	
misc07	30874	18.3	21375	13.2	0.10	221	–	
mitre	1	5.3	1	5.5	0.10	–	–	
mod008	43	0.7	43	0.9	0.15	–	–	
mod010	12	0.9	4	1.5	0.84	288	3	
neos-1109824	16175	120.7	23022	151.1	0.10	–	–	
neos18	11569	52.5	9468	45.5	0.09	–	–	
neos-849702	83053	1039.0	68072	1172.5	1.56	1636	–	
ns1208400	1786	278.5	5903	803.5	3.68	956	–	
ns1688347	4264	514.6	7047	467.6	0.29	780	–	
nsrand-ipx	1823524	limit	1412341	limit	1.09	240	–	
p0033	1	0.0	1	0.1	0.04	162	–	✓
p0201	267	1.9	111	1.9	0.08	187	–	✓
p0282	3	0.6	3	0.8	0.10	–	–	
p0548	5	0.3	5	0.5	0.13	435	–	✓
p2756	21	1.1	75	1.4	0.10	–	81	
protfold	9297	limit	6300	limit	1.69	6727	–	✓
reblock67	87898	213.9	114722	282.7	0.37	–	–	
rmine6	1075113	3223.1	592670	2058.5	0.22	–	–	
rococoC10-001000	1131839	limit	623971	3448.6	0.41	1370	–	
seymour	136912	limit	135458	limit	0.20	–	–	✓
stein27	3895	1.1	3683	1.1	0.04	–	–	
stein45	51942	13.3	48696	12.9	0.05	–	–	
tanglegram2	3	7.9	3	8.1	0.19	–	–	
30n20b8	490	841.6	7	113.5	0.65	332	–	
blend2	204	0.8	199	0.6	0.03	–	–	
bley_xl1	1	251.7	1	249.8	0.74	2372	–	✓
bnatt350	5432	503.1	3912	416.6	0.49	741	–	
csched010	562056	4016.4	950890	6157.4	0.34	47	–	
enlight13	2253556	935.7	6504809	3026.9	0.20	64	–	
enlight14	1471317	715.5	14565494	limit	0.21	67	–	
flugpl	278	0.1	275	0.1	0.02	–	–	✓

continued on next page

Table B.18 continued

	SCIP def		SCIP RL		Rapid Learning			
	Nodes	Time	Nodes	Time	RL Time	Ngds	Bds	Sol
gt2	1	0.1	1	0.1	0.05	199	–	✓
lectsched-4-obj	43645	516.9	22197	291.5	0.07	–	–	
manna81	1	0.7	1	0.9	0.19	–	–	
mzzv11	4618	297.0	1230	264.6	1.14	–	–	
mzzv42z	1232	140.2	532	179.6	4.17	–	–	
neos-1337307	379474	limit	416302	limit	3.82	–	35	
neos-1601936	26834	limit	28011	limit	2.13	691	–	
neos-686190	8144	100.7	7822	105.7	1.04	114	–	
neos-934278	4603	limit	3110	limit	19.71	6	–	
ns1830653	42280	545.5	53081	758.7	3.58	1736	5	✓
opt1217	1	0.7	1	0.8	0.08	1	–	
pw-myciel4	422214	3515.3	428238	3342.3	0.89	301	–	
qnet1	43	4.9	31	4.3	0.28	308	–	
qnet1_o	9	2.5	39	5.3	0.16	188	12	
sp98ir	5613	88.0	6224	96.1	0.80	908	–	

Table B.19.: Rapid Learning results for pure BPs and IPs from Infeasible and Primal test set of MIPLIB 2010

	SCIP def		SCIP RL		Rapid Learning			
	Nodes	Time	Nodes	Time	RL Time	Ngds	Bds	Sol
acc-tight4	1011	249.4	626	112.1	2.77	1090	–	
acc-tight5	562	121.1	790	123.0	2.13	2200	–	
acc-tight6	4761	489.2	228	52.3	2.26	2020	–	
bnatt350	5432	501.0	3912	416.6	0.48	741	–	
m100n500k4r1	7952424	limit	7729744	6820.0	0.05	–	–	
neos-1440225	12987	332.7	6353	125.2	1.32	871	–	
neos-738098	2053	limit	1231	limit	10.86	397	–	
neos-849702	83053	1038.8	68072	1169.4	1.56	1636	–	
neos-957389	10	12.4	22	21.2	8.12	335	8	
neos-785912	471	83.7	610	65.0	0.38	369	–	
neos788725	76260	211.9	74750	255.2	0.29	537	–	
neos-820146	5052473	limit	5497291	limit	0.15	275	–	
neos-820157	3044990	limit	2864395	limit	0.48	166	–	
neos858960	1458652	1483.2	1570385	1561.4	0.03	–	–	
neos-859770	1	101.6	1	115.2	12.91	679	–	
ns1686196	1767	39.6	7	20.1	0.47	985	13	
ns1745726	8	52.0	261	68.4	0.56	1133	22	
ns1769397	4430	251.4	328	69.0	0.27	852	12	
p2m2p1m1p0n100	161700567	limit	162771381	limit	0.02	–	–	
lectsched-2	1554	73.2	691	65.0	1.11	354	2	
neos-1224597	132	31.3	140	36.9	2.20	1560	35	
neos-555424	1384431	limit	977383	limit	4.95	187	7	
neos6	2105	262.9	2996	359.8	27.27	725	–	
neos-932816	27633	limit	32340	limit	2.74	–	–	
neos-933638	192	456.7	320	647.3	19.41	–	–	
neos-933966	18525	6298.8	1767	1659.1	2.01	–	–	
neos-935627	2625	limit	3374	limit	3.06	–	–	
neos-935769	4113	5657.5	931	1590.8	3.87	–	–	
neos-937511	27	760.6	39	1043.5	7.33	–	–	
enlight14	1471317	712.4	14519009	limit	0.20	67	–	
enlight16	620942	441.6	1438781	841.0	0.20	58	–	
enlight9	281704	106.9	172864	62.2	0.05	47	–	

Table B.20.: comparison of Cloud Branching and Full Strong Branching on MMM instances

	cloud statistics				SCIP cloud branch		SCIP strong branch	
instance	%Succ	Pts	LPs	%Sav	Nodes	Time (s)	Nodes	Time (s)
10teams	64.3	2.7	50.3	79.3	129	105.3	348	488.1
aflow30a	0.0	–	–	–	166	19.6	182	21.7
air04	91.3	2.0	9.1	32.3	55	2087.9	57	2074.6
air05	46.8	2.0	3.0	3.4	166	1597.0	153	1541.6
ash608gpia-3col	100.0	4.3	2240.1	86.7	5	1072.1	9	2406.8
bell3a	0.0	–	–	–	26 588	6.6	26 590	6.3
bell5	0.2	2.0	2.0	0.6	851	0.7	865	0.7
bienst2	25.5	2.4	6.6	34.1	21 729	1586.4	21 210	1707.6
binkar10_1	4.4	2.0	4.8	3.3	45 080	1715.7	48 835	1744.9
blend2	9.3	2.0	2.0	5.4	108	0.8	110	0.7
cap6000	0.0	–	–	–	1 601	3.3	1 545	3.1
dcmulti	0.0	–	–	–	120	2.3	120	2.5
dfn-gwin-UUM	0.0	–	–	–	5 897	435.1	5 918	431.6
eil33-2	0.0	–	–	–	484	739.8	480	734.2
enigma	5.2	2.0	9.2	14.6	27	0.5	249	0.6
fiber	0.0	–	–	–	16	1.1	16	1.3
fixnet6	0.0	–	–	–	9	2.3	9	2.2
flugpl	0.0	–	–	–	134	0.5	134	0.5
gesa2-o	0.0	–	–	–	5	1.4	5	1.5
gesa2	0.0	–	–	–	3	1.0	3	1.0
gesa3	0.0	–	–	–	11	1.4	15	1.5
gesa3_o	0.0	–	–	–	9	1.5	9	1.7
khb05250	0.0	–	–	–	4	0.5	4	0.5
l152lav	3.9	2.0	6.7	3.5	53	4.7	65	7.1
lseu	15.4	2.1	3.4	12.7	364	0.7	382	0.5
map18	0.0	–	–	–	103	1454.7	101	1701.6
map20	0.0	–	–	–	87	1129.0	91	1384.7
mas74	0.0	–	–	–	574 769	1389.5	574 769	1321.8
mas76	0.0	–	–	–	81 106	123.7	84 280	123.0
mik-250-1-100-1	0.0	–	–	–	290 018	1681.4	290 038	1628.3
mine-166-5	0.0	–	–	–	2 001	142.6	1 994	155.6
misc03	11.7	2.3	10.4	25.4	68	1.4	65	1.5
misc06	5.9	2.0	4.0	6.7	13	0.8	13	0.6
misc07	13.2	2.1	7.1	23.5	2 300	62.9	2 365	57.9
mod008	0.0	–	–	–	104	0.8	111	0.8
mod010	0.0	–	–	–	10	1.0	10	1.1
mod011	0.0	–	–	–	321	989.9	321	1069.9
modglob	0.0	–	–	–	299	2.9	299	2.8
neos-1109824	51.9	2.3	26.0	68.4	1 246	473.0	1 023	390.9
neos-1396125	69.4	2.2	8.3	55.1	2 714	2355.7	2 976	2653.2
neos-476283	0.0	–	–	–	445	887.5	323	680.2
neos-686190	3.7	2.0	9.7	4.8	1 451	540.6	2 085	774.5
noswot	86.9	2.4	16.5	74.2	337 012	957.6	210 056	869.0
ns1766074	0.0	2.1	5.5	0.1	241 641	492.3	241 801	470.2
nw04	0.0	–	–	–	5	54.8	5	46.4
p0033	0.0	–	–	–	5	0.5	5	0.5
p0201	47.0	2.3	23.2	64.6	52	2.6	51	3.0
p0282	0.0	–	–	–	3	0.5	3	0.5
p0548	0.0	–	–	–	5	0.5	5	0.5
p2756	2.6	2.0	4.0	2.5	82	1.9	146	2.0

Table B.20 continued

	cloud statistics				SCIP cloud branch		SCIP strong branch	
instance	%Succ	Pts	LPs	%Sav	Nodes	Time (s)	Nodes	Time (s)
pk1	0.1	2.8	16.4	0.6	76 569	257.8	77 616	233.1
pp08a	0.0	–	–	–	300	3.7	251	3.0
pp08aCUTS	0.2	2.0	2.0	0.1	213	3.2	284	4.2
qiu	10.3	2.1	10.3	17.9	14 858	1515.7	16 290	1895.5
qnet1	17.6	2.0	6.0	4.4	5	3.8	5	3.4
qnet1_o	20.0	2.0	3.8	2.8	22	9.2	22	10.3
ran16x16	4.4	2.0	2.3	2.6	28 684	1184.3	27 051	964.4
reblock67	0.0	–	–	–	28 052	1528.8	33 290	1773.3
rentacar	27.3	2.0	3.3	22.2	13	3.4	14	3.5
rmatr100-p10	0.1	2.0	2.0	0.0	163	952.8	164	950.2
rmatr100-p5	0.0	–	–	–	33	1327.1	33	1321.6
rout	32.6	2.2	11.8	46.0	1 561	79.3	1 712	85.8
set1ch	0.0	–	–	–	16	0.9	17	1.0
sp98ir	2.1	2.0	3.5	1.3	609	404.1	876	507.4
stein27	29.1	2.3	6.2	22.9	787	2.2	775	2.0
stein45	20.7	2.1	5.8	11.2	7 909	73.8	8 446	77.3
tanglegram2	0.0	–	–	–	2	27.3	2	34.3
vpm2	9.4	2.0	2.2	3.4	46	1.3	48	1.3

Table B.21.: comparison of Cloud Branching and Full Strong Branching on COR@L instances

	cloud statistics				cloud branching		full strong branching	
instance	%Succ	Pts	LPs	%Sav	Nodes	Time (s)	Nodes	Time (s)
22433	0.0	–	–	–	4	1.1	4	1.2
23588	0.2	2.0	14.0	0.3	148	6.0	174	6.3
aligninq	1.8	2.0	9.0	1.1	24	20.6	32	23.1
bc1	0.0	–	–	–	604	132.6	616	124.0
bc	0.0	–	–	–	1 985	2437.7	1 985	2323.3
bienst1	31.7	2.2	6.0	39.0	2 712	151.1	2 737	172.1
bienst2	25.5	2.4	6.6	34.1	21 729	1587.7	21 210	1703.3
binkar10_1	4.4	2.0	4.8	3.3	45 080	1714.4	48 835	1744.8
dano3_3	0.0	–	–	–	9	235.5	9	153.3
dano3_4	0.0	–	–	–	4	176.7	4	177.4
haprp	0.0	–	–	–	20 289	1696.4	19 844	1629.9
neos-1053591	94.5	2.3	8.7	70.0	1 794	19.2	46 259	367.4
neos-1109824	51.9	2.3	26.0	68.4	1 246	482.7	1 023	390.7
neos-1120495	38.7	2.2	19.4	49.7	102	18.7	75	17.6
neos-1122047	100.0	3.5	42.0	96.6	3	80.6	2	34.1
neos-1200887	86.2	2.5	13.8	63.5	981	234.4	1 465	381.7
neos-1211578	74.9	2.4	10.9	74.6	49 619	315.6	32 225	323.5
neos-1224597	98.6	6.7	631.1	95.0	70	406.8	80	864.5
neos-1228986	74.8	2.4	11.7	70.2	42 072	358.7	39 690	400.6
neos-1281048	91.8	5.8	133.8	88.1	59	49.8	80	170.7
neos-1337489	74.9	2.4	10.9	74.6	49 619	312.0	32 225	320.5
neos-1367061	0.0	–	–	–	16	1601.4	16	1683.2
neos-1396125	69.4	2.2	8.3	55.1	2 714	2359.8	2 976	2659.6
neos-1413153	81.8	2.9	292.5	88.6	192	219.9	462	2546.4
neos-1415183	90.9	2.6	252.4	85.7	19	6.6	56	43.6
neos-1420205	82.5	2.1	8.0	33.5	10 674	46.8	7 840	45.5
neos-1437164	74.7	2.2	11.0	47.1	80	1.8	47	1.9
neos-1440225	92.3	4.7	381.7	96.6	6	11.0	134	1387.4
neos-1440447	88.7	2.8	18.4	80.6	7 676	207.5	22 496	747.9
neos-1441553	78.0	2.3	26.0	58.1	133	16.2	215	86.7
neos-1445743	0.0	–	–	–	2	101.1	2	64.4
neos-1445755	5.0	2.0	4.0	16.7	3	75.0	3	57.6
neos-1445765	1.6	2.0	2.0	0.3	5	374.9	5	236.2
neos-1460265	99.8	4.0	216.2	80.2	6 997	1354.8	1 125	540.5
neos-1480121	23.0	2.0	3.5	25.0	1 288	3.3	1 961	4.0
neos-1489999	0.0	–	–	–	21	28.6	21	32.0
neos-476283	0.0	–	–	–	445	885.7	323	687.4
neos-480878	24.2	2.0	3.2	7.3	2 803	230.9	3 517	279.0
neos-494568	94.2	3.0	181.7	76.8	291	398.2	285	1082.3
neos-501474	48.2	2.0	4.0	46.9	158	1.3	104	0.7
neos-504674	50.5	2.0	5.9	16.6	1 256	399.7	1 230	426.9
neos-504815	35.6	2.1	6.1	16.5	510	75.4	502	83.3
neos-506422	16.2	2.1	3.7	20.9	1 451	540.7	959	337.3
neos-512201	48.7	2.0	6.0	15.5	665	175.7	436	149.2
neos-522351	0.0	–	–	–	3	1.1	3	1.0
neos-525149	55.3	3.0	88.7	45.8	46	17.9	187	193.4
neos-530627	0.0	–	–	–	2	0.5	2	0.5
neos-538867	72.8	3.0	21.4	76.2	6 697	318.1	4 358	208.9
neos-538916	77.5	3.2	23.9	77.4	4 642	371.5	3 496	294.6
neos-544324	99.9	2.0	15.2	92.3	7	301.4	7	149.9

Table B.21 continued

	cloud statistics				cloud branching		full strong branching	
instance	%Succ	Pts	LPs	%Sav	Nodes	Time (s)	Nodes	Time (s)
neos-547911	90.0	2.1	10.8	85.1	30	244.3	30	184.5
neos-555694	71.3	2.6	98.6	69.9	65	36.5	177	301.7
neos-555771	92.7	2.4	107.7	80.2	32	17.8	70	72.4
neos-570431	71.0	2.0	8.1	59.4	60	290.2	76	314.7
neos-584851	47.0	2.1	20.0	60.3	56	778.1	38	840.5
neos-585192	0.0	–	–	–	333	40.1	345	40.6
neos-585467	1.2	2.0	12.0	1.4	125	10.6	133	10.7
neos-593853	0.0	–	–	–	10 157	52.4	12 204	56.0
neos-595905	0.0	–	–	–	418	25.2	473	29.5
neos-595925	0.0	–	–	–	1 166	51.6	1 189	51.8
neos-598183	0.0	–	–	–	488	6.8	486	7.1
neos-611838	0.0	–	–	–	193	89.5	193	94.0
neos-612125	0.0	–	–	–	92	47.1	92	50.3
neos-612143	0.0	–	–	–	130	55.1	128	59.7
neos-612162	0.0	–	–	–	122	74.6	126	80.3
neos-631694	93.9	2.9	56.8	49.5	94	57.3	101	93.6
neos-686190	3.7	2.0	9.7	4.8	1 451	537.9	2 085	776.4
neos-709469	12.5	2.3	22.4	58.6	1 608	3.5	28	1.6
neos-717614	0.0	–	–	–	1 059	65.9	1 061	65.3
neos-775946	95.4	2.8	93.4	81.3	234	140.6	413	343.6
neos-785899	93.0	2.8	94.2	77.7	179	130.7	266	247.6
neos-785914	83.8	3.4	135.0	92.0	109	123.4	20	296.6
neos-801834	0.0	–	–	–	11	841.5	11	817.6
neos-803219	0.1	2.0	2.0	0.0	4 131	72.8	4 231	70.9
neos-803220	0.0	–	–	–	18 713	179.3	17 175	166.5
neos-806323	28.1	2.0	2.7	11.0	3 258	137.6	3 645	142.6
neos-807639	4.1	2.0	2.6	3.1	1 130	18.6	1 120	17.2
neos-807705	20.3	2.0	2.2	6.3	2 373	88.3	2 241	80.0
neos-808072	72.1	2.3	32.3	51.7	43	379.0	90	1905.3
neos-810326	34.9	2.0	4.0	6.2	267	2394.3	266	2431.7
neos-820879	45.1	2.0	3.8	9.6	127	281.1	114	210.3
neos-825075	84.6	3.9	60.0	80.5	18	3.0	49	7.8
neos-839859	0.1	2.0	12.0	0.1	1 084	773.1	1 628	938.6
neos-862348	35.8	2.1	19.8	21.9	99	33.9	70	38.4
neos-863472	32.8	2.2	15.6	63.2	88 330	2330.9	68 169	2264.0
neos-880324	62.8	2.4	22.2	78.7	62	1.8	15	1.0
neos-892255	100.0	2.5	278.6	97.3	8	720.1	5	1590.8
neos-906865	0.0	–	–	–	7 079	462.4	7 065	453.8
neos-912015	93.2	5.1	130.8	94.2	791	473.3	209	322.1
neos-916173	0.0	–	–	–	1 497	390.2	1 478	392.0
neos-933550	83.3	8.4	638.8	96.6	5	10.1	25	58.2
neos-933815	47.7	2.0	5.7	32.3	61 797	801.5	55 797	661.4
neos-934531	99.3	3.4	89.8	96.1	27	293.0	51	1432.9
neos-941698	97.7	6.1	357.2	95.8	19	14.2	44	70.5
neos-942323	99.7	4.0	187.8	97.7	189	64.0	2 205	1240.4
neos-955215	68.4	2.1	9.1	53.0	7 574	61.3	6 593	52.9
neos-957270	83.1	2.8	159.9	89.0	14	471.3	17	228.4
nsa	0.0	–	–	–	258	2.8	258	3.0
nug08	0.0	–	–	–	3	24.9	3	23.2
prod1	0.0	2.0	8.0	0.0	4 053	33.8	3 820	31.4
prod2	0.0	–	–	–	25 200	361.6	25 227	354.0
qap10	33.3	2.0	2.0	40.0	2	177.3	2	157.3
sp98ir	2.1	2.0	3.5	1.3	609	403.5	876	503.2

Table B.21 continued

	cloud statistics				cloud branching		full strong branching	
instance	%Succ	Pts	LPs	%Sav	Nodes	Time (s)	Nodes	Time (s)
Test3	0.0	–	–	–	10	8.0	10	8.0

Table B.22.: Performance of SCIP 3.0.2, default mode, on the MMM test set

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
10teams	312	379	15.0	15.3	0.417	2	0	0.9	12
30n20b8	62	171	198.6	405.8	8.195	0	5	44.6	17
a1c1s1	1	197 593	0.3	limit	0.985	12	343	812.7	18
acc-tight5	3 103	3 103	459.9	459.9	12.774	1	0	64.7	14
aflow30a	1	2 531	0.2	11.5	0.101	0	3	1.3	18
aflow40b	1	339 338	0.9	1591.0	0.777	2	32	104.4	18
air03	1	1	4.9	5.5	0.148	1	1	0.1	5
air04	80	169	72.7	79.2	2.022	2	2	13.1	15
air05	1	201	1.6	38.2	0.057	4	1	0.7	9
app1-2	17	68	731.3	1079.9	20.306	0	1	245.1	11
arki001	35	1 675 205	4.1	limit	0.115	0	58	889.1	18
ash608gpia-3col	–	7	–	68.1	–	0	0	23.0	9
atlanta-ip	10	4 741	305.2	limit	12.951	20	10	485.8	18
beasleyC3	1	484 730	0.1	limit	3.622	0	1340	1114.9	18
bell3a	1	26 833	0.0	6.2	0.000	0	275	1.3	19
bell5	1	1 273	0.0	0.8	0.000	1	291	0.2	19
bab5	6	15 549	119.2	limit	4.138	0	27	846.7	17
biella1	1	3 530	1.1	869.9	1.065	13	13	203.5	18
bienst2	1	81 122	1.9	396.6	0.191	1	16	39.0	16
binkar10_1	15	106 945	2.3	150.1	0.069	4	103	19.5	17
blend2	1	153	0.3	0.7	0.008	0	1	0.2	18
bley_xl1	1	49	383.2	496.3	11.451	0	4	5.1	11
bnatt350	9 951	9 951	813.8	813.8	22.606	1	0	66.8	15
cap6000	1	3 296	0.8	2.8	0.022	0	542	0.5	17
core2536-691	1	191	13.5	418.4	0.494	0	138	181.7	17
cov1075	0	592 669	0.0	limit	0.059	0	767	835.3	14
csched010	1 270	521 037	38.4	limit	1.856	0	13	109.4	18
dano3mip	1	769	1.6	limit	8.349	0	2	2047.7	17
danoint	164	624 716	11.9	limit	0.411	1	11	259.1	16
dcmulti	15	161	1.4	2.1	0.039	13	2	0.1	10
dfn-gwin-UUM	1	54 610	0.0	120.0	0.121	0	742	12.5	17
disctom	1	1	4.5	4.5	0.124	0	1	2.9	5
ds	1	301	4.4	limit	81.533	0	6	1391.6	16
dsbmip	1	1	0.6	0.6	0.017	0	1	0.2	6
egout	1	1	0.0	0.0	0.000	0	17	0.0	6
eil33-2	1	14 777	0.4	124.8	0.397	14	110	12.6	16
eilB101	1	12 401	0.1	774.4	2.449	19	384	23.8	16
enigma	992	992	0.7	0.7	0.018	1	0	0.1	14
enlight13	401 453	410 142	227.2	233.1	6.306	1	0	18.1	16
enlight14	–	1 329 508	–	810.2	–	0	0	39.6	14
ex9	1	1	12.5	12.5	0.347	0	1	0.0	1
fast0507	0	1 476	0.3	1546.0	0.431	1	436	369.0	17
fiber	1	15	0.0	1.4	0.007	0	14	0.1	10
fixnet6	1	11	0.1	2.3	0.006	0	52	0.2	9
flugpl	1	202	0.0	0.1	0.000	1	1	0.0	16
gen	1	1	0.1	0.1	0.003	0	6	0.0	6
gesa2-o	1	1	0.5	1.3	0.015	1	4	0.2	10
gesa2	1	3	0.1	1.2	0.027	0	5	0.1	8
gesa3	1	7	0.1	1.8	0.007	1	7	0.2	8

Table B.22 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
gesa3_o	1	5	0.1	1.8	0.017	1	5	0.1	10
glass4	1	1 189 795	1.3	1246.4	7.802	1	31	107.2	18
gmu-35-40	0	6 992 427	0.0	limit	0.046	0	65	741.3	18
gt2	1	1	0.0	0.1	0.001	0	4	0.0	6
iis-100-0-cov	0	107 156	0.0	1907.1	0.046	1	437	274.9	16
iis-bupa-cov	0	76 792	0.0	limit	0.447	2	501	852.2	16
iis-pima-cov	0	7 787	0.0	774.4	0.285	0	459	183.7	16
khb05250	1	5	0.0	0.5	0.001	0	19	0.1	9
lectsched-4-obj	1	3 855	4.2	85.4	1.864	3	24	23.5	17
liu	1	986 455	0.1	limit	14.720	0	188	2121.8	17
l152lav	5	77	0.9	3.6	0.026	7	1	0.4	16
lseu	1	493	0.0	0.5	0.002	1	38	0.2	17
m100n500k4r1	0	3 985 310	0.0	limit	4.015	1	667	69.1	17
macrophage	0	420 169	0.0	limit	3.225	0	372	1363.2	16
manna81	0	1	0.0	0.8	0.003	1	8	0.1	5
map18	1	321	2.9	455.4	1.204	0	4	152.3	16
map20	1	259	2.9	345.2	0.981	0	3	95.4	16
markshare1	1	42 741 304	0.0	limit	89.081	2	1032	227.8	15
markshare2	1	38 929 335	0.0	limit	93.502	4	1067	231.5	16
mas74	1	3 941 218	0.0	721.8	0.019	1	8	61.6	18
mas76	1	340 598	0.0	51.7	0.002	0	9	7.2	18
mcsched	1	16 878	0.1	211.5	0.032	16	33	9.6	17
mik-250-1-100-1	0	1 328 923	0.0	283.6	0.006	0	473	17.7	17
mine-166-5	0	833	0.0	39.2	0.855	8	24	1.1	15
mine-90-10	0	120 339	0.0	788.5	0.862	4	74	49.2	15
misc03	5	71	0.4	1.1	0.012	3	4	0.1	11
misc06	1	5	0.1	0.7	0.003	0	28	0.2	10
misc07	5	33 069	0.7	19.2	0.021	3	26	1.1	18
mitre	1	1	4.8	5.2	0.136	0	7	0.2	8
mkc	0	1 510 467	0.0	limit	1.199	0	784	626.9	18
mod008	0	334	0.0	0.9	0.000	0	292	0.0	16
mod010	3	4	0.8	0.9	0.022	1	0	0.3	8
mod011	0	1 079	0.0	153.4	0.944	5	38	26.5	17
modglob	1	536	0.0	1.1	0.000	0	289	0.5	15
momentum1	1	17 125	29.6	limit	6.345	0	2	788.7	18
momentum2	5 770	25 142	1680.2	limit	51.791	7	4	878.3	19
momentum3	1	8	319.3	limit	64.190	0	1	291.7	11
msc98-ip	–	1 670	–	limit	100.000	0	0	558.2	15
mspp16	1	40	664.9	limit	18.825	0	58	109.8	13
mzzv11	0	2 304	0.1	259.4	3.396	9	16	40.1	16
mzzv42z	0	2 483	0.1	926.5	4.211	10	44	275.4	16
n3div36	1	109 303	3.0	limit	3.247	3	486	102.4	17
n3seq24	1	977	55.1	limit	19.787	0	20	1093.9	16
n4-3	1	54 219	0.2	777.4	0.287	2	698	88.8	18
neos-1109824	6	22 805	5.8	178.2	0.178	10	9	9.6	17
neos-1337307	19	152 561	65.7	limit	1.829	8	324	184.0	17
neos-1396125	1 742	42 001	121.9	1183.8	3.630	5	1	47.7	18
neos13	1	11 377	2.5	limit	9.873	3	190	657.1	18
neos-1601936	196	8 768	205.9	limit	63.620	6	6	863.4	17
neos18	1	9 141	0.2	38.1	0.135	4	15	2.9	17
neos-476283	1	468	66.1	281.4	1.867	0	9	84.4	18
neos-686190	67	4 216	14.9	65.6	0.495	4	5	4.4	17
neos-849702	76 493	76 493	1459.1	1459.1	40.528	1	0	56.2	14

Table B.22 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
neos-916792	1	129 338	13.9	686.6	0.594	10	144	92.8	18
neos-934278	0	1 950	0.1	limit	10.859	0	14	1524.9	17
net12	9	4 287	90.9	2505.5	5.276	0	4	211.5	18
netdiversion	–	64	–	limit	100.000	0	0	1721.3	12
newdano	1	491 739	2.7	limit	0.979	8	9	224.3	16
noswot	1	309 903	0.0	79.9	0.002	0	270	4.4	17
ns1208400	3 354	3 354	741.8	741.8	20.604	1	0	74.4	14
ns1688347	723	24 226	110.4	1731.6	5.562	1	3	96.7	17
ns1758913	1	41	127.9	limit	68.913	0	3	346.0	14
ns1766074	–	952 281	–	610.4	–	0	0	2.8	15
ns1830653	1 046	43 657	61.7	513.7	2.411	4	2	59.4	17
nsrand-ipx	1	718 286	1.4	limit	3.004	0	231	213.9	17
nw04	3	5	27.6	36.9	0.770	2	0	10.7	8
opm2-z7-s2	0	4 260	0.1	810.6	2.436	0	708	107.9	16
opt1217	1	1	0.0	0.9	0.002	0	5	0.0	7
p0033	1	1	0.0	0.0	0.000	0	7	0.0	7
p0201	1	53	0.6	1.4	0.017	6	5	0.1	11
p0282	0	5	0.0	0.4	0.000	0	16	0.0	8
p0548	1	3	0.2	0.3	0.006	1	3	0.1	9
p2756	1	118	0.3	2.0	0.011	5	65	0.2	13
pg5_34	0	323 375	0.0	1465.8	0.053	1	377	283.1	17
pigeon-10	0	3 260 012	0.0	limit	0.128	0	2	83.0	16
pk1	1	326 083	0.0	65.4	0.065	0	13	9.5	18
pp08a	1	531	0.0	1.5	0.006	3	255	0.2	17
pp08aCUTS	1	189	0.0	1.5	0.008	1	214	0.4	13
protfold	–	7 055	–	limit	100.000	0	0	309.8	14
pw-myciel4	65	419 688	31.4	limit	1.022	0	3	100.1	17
qiu	1	12 989	0.1	82.7	0.806	5	776	7.9	18
qnet1	1	9	0.2	3.4	0.014	0	12	0.2	9
qnet1_o	1	1	0.0	1.6	0.004	1	5	0.1	9
rail507	1	1 981	3.2	1886.9	0.643	0	264	419.1	18
ran16x16	1	449 274	0.0	377.2	0.038	1	1163	15.9	18
reblock67	0	113 330	0.0	282.4	0.317	18	39	15.2	15
rd-rplusc-21	142	18 628	871.1	limit	25.676	17	5	91.2	18
rentacar	5	12	2.7	3.2	0.076	2	1	0.4	9
rgn	1	1	0.0	0.2	0.005	1	1	0.0	5
rmatr100-p10	1	847	1.0	157.4	0.109	1	24	39.6	18
rmatr100-p5	1	369	1.1	304.9	0.462	0	9	75.0	18
rmine6	0	461 820	0.0	1694.9	0.084	2	163	144.4	15
rocII-4-11	17	24 116	29.7	336.2	2.904	6	16	36.1	19
rococoC10-001000	1	623 365	0.3	limit	0.479	0	160	59.5	18
roll3000	1	525 721	4.8	1883.0	0.329	7	136	160.9	18
rout	1	16 770	0.1	26.3	0.066	2	53	5.8	17
satellites1-25	1	17 788	43.6	2569.3	34.833	1	1	199.5	18
set1ch	1	11	0.0	0.8	0.006	1	84	0.1	11
seymour	0	53 827	0.0	limit	0.883	0	1134	1116.9	16
sp97ar	1	2 232	2.8	limit	9.077	0	15	417.2	17
sp98ic	1	62 947	3.2	limit	2.369	0	149	568.1	17
sp98ir	1	6 541	2.1	90.0	0.170	2	145	7.0	17
stein27	0	4 311	0.0	1.1	0.000	0	296	0.1	15
stein45	0	51 700	0.0	13.8	0.005	2	486	1.5	15
stp3d	–	1	–	limit	100.000	0	0	1715.4	5
swath	1	590 024	0.4	limit	6.819	11	53	154.2	18

Table B.22 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
t1717	48	1 289	288.3	limit	15.729	0	1	808.1	16
tanglegram1	0	37	0.2	991.1	4.813	0	10	190.5	13
tanglegram2	0	3	0.0	6.7	0.108	1	14	0.3	7
timtab1	132	1 132 378	2.0	520.6	0.134	26	54	16.7	17
timtab2	1 452	3 939 088	9.8	limit	5.662	43	75	545.5	17
tr12-30	1	927 683	0.5	1268.2	0.027	4	261	114.0	18
triptim1	1	11	402.9	787.0	11.214	0	3	398.2	11
unitcal_7	1	13 548	112.7	1104.9	3.140	0	23	164.2	18
vpm1	1	1	0.0	0.1	0.000	1	15	0.0	6
vpm2	1	617	0.0	1.2	0.004	0	16	0.3	18
vpphard	51	6 249	271.2	limit	68.915	0	18	1764.6	18
zib54-UUE	1	258 612	0.2	2090.9	0.330	0	878	132.6	17

Table B.23.: Performance of SCIP 3.0.2, primal heuristics deactivated, on the MMM test set

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
10teams	83	250	12.3	14.3	0.345	3
30n20b8	2 103	2 776	889.5	988.6	25.423	4
a1c1s1	36 807	406 213	692.8	limit	21.183	21
acc-tight5	1 214	1 214	118.2	118.2	3.278	1
aflow30a	271	5 588	12.0	24.9	0.361	8
aflow40b	552	263 699	67.9	2148.5	7.419	7
air03	1	1	5.7	5.7	0.158	1
air04	38	207	44.9	59.6	1.254	11
air05	14	434	20.1	39.7	0.583	6
app1-2	–	120	–	limit	100.000	0
arki001	111 067	1 943 418	444.9	limit	12.363	34
ash608gpia-3col	–	35	–	78.9	–	0
atlanta-ip	544	7 081	665.0	limit	23.138	30
beasleyC3	–	618 376	–	limit	100.000	0
bell3a	157	39 427	0.1	10.5	0.003	6
bell5	117	1 649	0.1	0.6	0.003	4
bab5	1 491	10 326	364.1	limit	13.634	1
biella1	16 807	21 775	2436.5	3177.9	67.745	80
bienst2	497	107 039	13.8	424.1	0.545	8
binkar10_1	922	100 370	5.9	121.8	0.170	9
blend2	2 728	3 990	1.5	1.9	0.044	8
bley_xl1	12	21	472.6	474.9	13.140	2
bnatt350	10 300	10 300	764.9	764.9	21.248	1
cap6000	100 340	1 328 821	152.4	3133.9	5.187	222
core2536-691	205	837	292.7	461.4	8.146	2
cov1075	119	731 859	10.6	limit	0.472	2
csched010	2 502	510 836	46.5	limit	3.202	4
dano3mip	–	2 661	–	limit	100.000	0
danoint	946	672 684	17.2	limit	0.597	4
dcmulti	44	146	2.1	2.2	0.058	13
dfn-gwin-UUM	28	75 493	3.1	146.2	0.195	7
disctom	–	223 311	–	limit	100.000	0
ds	–	707	–	limit	100.000	0
dsbmip	40	78	2.2	2.5	0.061	5
egout	1	1	0.0	0.0	0.000	1
eil33-2	3	14 069	6.0	110.5	0.368	15
eilB101	14	22 086	39.5	1179.7	2.989	18
enigma	135	135	0.3	0.3	0.008	1
enlight13	2 256 046	7 708 119	974.0	limit	27.056	1
enlight14	–	7 289 960	–	limit	–	0
ex9	1	1	12.3	12.3	0.342	1
fast0507	39	2 929	227.3	2392.4	6.602	2
fiber	13	199	1.2	2.7	0.039	6
fixnet6	76	109	2.1	2.2	0.058	6
flugpl	161	239	0.0	0.1	0.000	2
gen	1	1	0.1	0.1	0.003	1
gesa2-o	46	67	1.5	1.6	0.042	3
gesa2	23	46	1.4	1.5	0.039	2
gesa3	99	142	3.4	3.5	0.094	3

Table B.23 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
gesa3_o	123	191	4.8	5.1	0.133	3
glass4	609	4 565 048	2.8	limit	34.863	35
gmu-35-40	3 961	8 736 110	3.8	limit	0.213	7
gt2	29	29	0.1	0.1	0.003	1
iis-100-0-cov	42	92 002	26.4	1517.2	0.737	5
iis-bupa-cov	47	106 606	36.3	limit	1.078	6
iis-pima-cov	32	13 852	54.8	828.3	1.930	5
khb05250	6	7	0.4	0.4	0.011	1
lectsched-4-obj	42 516	72 640	238.8	419.8	9.932	6
liu	–	3 276 403	–	limit	100.000	0
l152lav	5	65	0.9	3.5	0.026	9
lseu	12	362	0.1	0.3	0.004	4
m100n500k4r1	14	4 134 202	2.1	limit	4.064	4
macrophage	561	474 484	24.0	limit	23.715	2
manna81	1	1	0.8	0.8	0.022	1
map18	417	1 000	288.5	448.6	8.011	4
map20	174	638	196.4	323.6	5.468	3
markshare1	30	48 534 921	0.1	limit	84.127	17
markshare2	33	42 076 149	0.1	limit	94.236	15
mas74	253	3 773 964	0.5	605.8	0.114	5
mas76	339	355 132	0.4	49.4	0.020	6
mcsched	99	30 212	16.4	399.2	0.470	17
mik-250-1-100-1	1 983	4 307 977	1.1	1286.5	0.310	8
mine-166-5	66	1 727	31.0	41.7	0.912	23
mine-90-10	3 084	183 159	44.5	1027.9	1.314	26
misc03	5	100	0.3	1.1	0.009	3
misc06	34	41	0.4	0.4	0.011	2
misc07	6	13 713	0.6	9.5	0.019	7
mitre	1	1	5.0	5.0	0.139	1
mkc	2 339	1 311 979	30.5	limit	5.745	2
mod008	9	730	0.8	1.3	0.024	13
mod010	6	13	0.7	0.9	0.019	4
mod011	13	1 300	28.8	162.2	0.939	11
modglob	145	1 572	0.9	1.4	0.025	7
momentum1	12 091	21 220	2728.4	limit	77.644	5
momentum2	4 994	35 746	876.3	limit	30.948	12
momentum3	–	39	–	limit	100.000	0
msc98-ip	–	1 845	–	limit	100.000	0
mspp16	15	131	896.6	limit	26.021	6
mzzv11	90	3 274	112.0	275.6	3.140	10
mzzv42z	565	3 780	187.3	540.4	5.438	76
n3div36	11	111 410	16.0	limit	2.658	17
n3seq24	–	672	–	limit	100.000	0
n4-3	1 403	47 524	52.8	636.4	1.483	11
neos-1109824	10	16 146	5.8	123.1	0.187	12
neos-1337307	72	147 409	66.3	limit	1.848	6
neos-1396125	2 403	49 625	104.2	1044.4	2.937	4
neos13	6	24 526	53.9	limit	16.224	73
neos-1601936	178	12 764	128.1	limit	53.992	7
neos18	699	9 145	14.8	35.3	0.413	5
neos-476283	392	1 041	220.6	259.2	6.139	7
neos-686190	278	10 536	23.4	109.9	0.927	11
neos-849702	194 360	194 360	2962.1	2962.1	82.278	1

Table B.23 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
neos-916792	8 984	400 907	47.6	2058.8	2.043	21
neos-934278	218	7 877	241.6	limit	21.051	11
net12	523	5 954	486.1	3316.0	18.694	2
netdiversion	–	464	–	limit	100.000	0
newdano	2 206	549 025	53.3	limit	3.139	9
noswot	235	625 806	0.9	166.8	0.038	6
ns1208400	946	1 308	253.8	281.5	7.056	1
ns1688347	809	15 969	128.4	839.0	7.388	5
ns1758913	–	90	–	limit	100.000	0
ns1766074	–	935 480	–	631.5	–	0
ns1830653	1 359	92 479	46.8	991.3	4.502	11
nsrand-ipx	647	564 683	25.1	limit	22.919	4
nw04	3	5	16.4	24.1	0.458	2
opm2-z7-s2	361	12 055	181.9	1028.1	7.488	10
opt1217	179	343	2.6	2.9	0.074	3
p0033	1	1	0.0	0.0	0.000	1
p0201	13	160	0.7	1.4	0.022	8
p0282	60	62	0.9	0.9	0.025	1
p0548	115	173	1.0	1.1	0.028	7
p2756	51	212	2.1	3.5	0.060	8
pg5_34	3 022	423 012	28.6	1897.8	0.860	7
pigeon-10	33	3 896 820	4.2	limit	0.117	1
pk1	447	284 512	0.7	48.8	0.152	4
pp08a	274	1 086	1.2	1.6	0.035	5
pp08aCUTS	388	741	1.2	1.4	0.033	2
protfold	–	7 337	–	limit	100.000	0
pw-myciel4	911	502 642	58.8	limit	1.767	3
qiu	304	11 787	17.8	69.6	0.503	4
qnet1	59	81	8.3	8.4	0.231	2
qnet1_o	62	79	6.8	7.0	0.189	4
rail507	225	3 108	453.0	2094.5	12.776	2
ran16x16	813	322 110	6.4	244.6	0.246	13
reblock67	1 343	206 815	25.1	437.7	0.721	16
rd-rplusc-21	11 097	16 247	2671.8	limit	75.793	3
rentacar	9	16	2.8	3.1	0.079	3
rgn	1	1	0.2	0.2	0.006	1
rmatr100-p10	223	876	65.2	124.1	1.814	3
rmatr100-p5	466	746	231.9	312.9	6.555	6
rmine6	411	663 629	29.7	2725.3	0.959	12
rocII-4-11	561	18 242	38.6	245.6	2.674	14
rococoC10-001000	22 316	689 401	121.0	limit	28.489	30
roll3000	1 340	652 257	29.4	2020.0	1.139	20
rout	330	32 932	5.3	42.3	0.179	7
satellites1-25	29	12 258	245.0	3531.0	22.167	8
set1ch	7	13	0.8	0.8	0.022	2
seymour	272	21 594	71.3	limit	7.243	1
sp97ar	–	8 176	–	limit	100.000	0
sp98ic	409	71 150	70.7	limit	7.388	4
sp98ir	131	5 478	22.8	72.0	0.663	9
stein27	61	4 232	0.4	1.0	0.011	2
stein45	184	48 990	2.0	11.9	0.056	4
stp3d	–	7	–	limit	100.000	0
swath	171	635 342	10.5	limit	8.666	11

Table B.23 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
t1717	–	1 403	–	limit	100.000	0
tanglegram1	19	47	705.8	891.0	19.755	4
tanglegram2	2	3	5.6	6.6	0.156	1
timtab1	12 177	998 093	8.6	423.1	0.330	17
timtab2	285 363	4 924 187	264.5	limit	9.811	34
tr12-30	14 253	1 195 588	29.8	1461.9	0.841	14
triptim1	179	179	812.5	812.5	22.570	1
unitcal_7	8 953	36 329	754.4	1832.4	20.954	208
vpm1	1	1	0.1	0.1	0.002	1
vpm2	133	1 461	0.9	1.4	0.026	7
vpphard	11 997	15 221	3019.5	limit	98.477	1
zib54-UUE	1 663	274 338	39.5	2077.8	1.126	8

Table B.24.: Performance of SCIP 3.0.2, default mode, on the GloMIQO test set

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
CLay0203M	13	32	0.1	0.1	0.003	3	1	0.1	13
CLay0204M	1	724	0.0	0.9	0.001	3	3	0.7	19
CLay0205M	1	7 555	0.1	3.4	0.007	4	6	1.3	19
CLay0303M	1	42	0.2	0.3	0.006	1	2	0.1	11
CLay0304M	8	582	0.1	0.9	0.004	10	5	0.5	20
CLay0305M	1	11 360	0.2	4.7	0.016	8	9	1.5	19
SLay04H	1	82	0.0	0.8	0.010	0	145	0.2	16
SLay04M	1	89	0.0	0.5	0.006	1	117	0.2	19
SLay05H	1	330	0.0	2.1	0.012	5	51	0.4	21
SLay05M	1	66	0.0	1.0	0.014	1	26	0.8	18
SLay06H	1	1 483	0.1	5.8	0.055	2	959	1.4	22
SLay06M	1	100	0.0	1.2	0.023	0	100	0.9	16
SLay07H	1	7 176	0.1	27.9	0.046	1	101	6.6	22
SLay07M	1	1 237	0.0	4.8	0.025	4	58	2.5	23
SLay08H	1	8 382	0.1	48.3	0.110	8	103	10.3	22
SLay08M	1	2 546	0.0	7.9	0.036	2	58	3.6	23
SLay09H	1	24 892	0.1	152.6	0.298	0	579	21.7	22
SLay09M	1	2 845	0.0	13.8	0.057	1	84	5.2	23
SLay10H	1	571 883	0.2	limit	0.812	0	315	198.6	22
SLay10M	1	415 668	0.0	1268.3	0.547	2	443	108.0	23
LeeCrudeOil1_05	6	31	0.7	1.2	0.020	4	1	0.3	11
LeeCrudeOil1_06	8	65	1.4	2.5	0.039	4	1	0.4	14
LeeCrudeOil1_07	24	123	1.8	3.7	0.050	9	0	1.2	12
LeeCrudeOil1_08	33	82	4.9	6.9	0.137	3	1	0.9	11
LeeCrudeOil1_09	17	109	3.0	9.3	0.084	3	0	2.1	13
LeeCrudeOil1_10	45	316	7.4	10.8	0.206	2	1	0.9	12
LeeCrudeOil2_05	15	82	2.6	3.9	0.075	0	3	0.7	11
LeeCrudeOil2_06	1	15	0.7	4.6	0.019	1	1	0.6	11
LeeCrudeOil2_07	11	79	6.3	9.1	0.176	2	0	0.5	11
LeeCrudeOil2_08	7	569	7.5	40.7	0.227	4	1	24.7	13
LeeCrudeOil2_09	84	424	15.6	35.7	0.436	2	2	15.6	12
LeeCrudeOil2_10	51	664	18.0	66.6	0.507	2	2	38.8	14
LeeCrudeOil3_05	18	88 933	4.8	117.5	0.137	7	11	20.7	18
LeeCrudeOil3_06	9	2 311 716	7.2	limit	0.218	11	21	139.1	18
LeeCrudeOil3_07	9	1 939 051	10.6	limit	0.332	7	24	169.1	20
LeeCrudeOil3_08	9	1 560 154	12.4	limit	0.398	19	20	205.0	20
LeeCrudeOil3_09	195	1 316 503	74.4	limit	2.116	19	7	173.6	20
LeeCrudeOil3_10	211	956 094	102.7	limit	2.867	20	4	140.6	20
LeeCrudeOil4_05	1	29	0.6	5.2	0.017	2	1	0.4	10
LeeCrudeOil4_06	1	11	0.9	4.8	0.025	0	2	0.5	10
LeeCrudeOil4_07	1	46	1.2	12.5	0.034	1	3	0.8	10
LeeCrudeOil4_08	15	16	19.0	20.6	0.531	0	1	0.7	10
LeeCrudeOil4_09	25	37	19.6	25.6	0.544	0	1	0.8	10
LeeCrudeOil4_10	27	360	28.8	48.7	0.800	1	1	2.8	14
LiCrudeOil_ex01	2 500	1 439 899	7.0	limit	0.592	0	1	95.3	20
LiCrudeOil_ex02	942	2 075 507	9.3	limit	0.336	5	13	372.8	20
LiCrudeOil_ex03	25 840	501 727	231.5	limit	8.396	0	1	172.4	19
LiCrudeOil_ex05	936	508 492	19.1	limit	3.434	0	1	286.3	20
LiCrudeOil_ex06	867	62 063	34.4	527.0	1.127	2	1	16.7	21

Table B.24 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
LiCrudeOil_ex11	1 336	397 389	51.4	limit	2.442	0	2	223.9	19
LiCrudeOil_ex21	860	263 488	52.2	limit	5.076	0	1	196.4	20
alan	1	3	0.0	0.1	0.000	1	3	0.0	11
du-opt5	1	66	0.1	0.6	0.006	0	150	0.3	21
du-opt	1	267	0.1	0.7	0.008	0	210	0.3	20
elf	1	406	0.0	0.7	0.011	1	15	0.5	21
ex1223a	0	1	0.0	0.0	0.000	0	5	0.0	8
ex1263a	1	308	0.0	0.4	0.001	5	9	0.3	18
ex1263	9	620	0.2	0.8	0.008	4	18	0.2	20
ex1264a	1	63	0.0	0.1	0.000	1	4	0.0	13
ex1264	13	97	0.1	0.2	0.003	6	0	0.0	10
ex1265a	1	105	0.0	0.1	0.001	2	4	0.0	10
ex1265	3	74	0.1	0.3	0.003	7	1	0.1	11
ex1266a	56	166	0.2	0.4	0.006	2	5	0.1	18
ex1266	1	23	0.2	0.8	0.009	1	1	0.0	9
fac3	1	15	0.0	0.2	0.000	0	20	0.2	11
feedtray2	1	1	0.2	0.2	0.004	0	1	0.1	4
fuel	1	3	0.0	0.1	0.000	0	3	0.1	11
gbd	0	1	0.0	0.0	0.000	1	3	0.0	1
meanvarx	1	1	0.0	0.3	0.000	1	5	0.2	12
netmod_dol1	1	52 207	0.3	2014.3	0.669	5	162	95.1	21
netmod_dol2	1	281	0.3	59.1	0.694	0	7	18.8	14
netmod_kar1	1	271	0.1	3.3	0.025	4	4	0.1	16
netmod_kar2	1	271	0.0	3.2	0.025	4	4	0.1	16
nous1	3 480	2 115 742	4.8	limit	0.134	0	127	22.9	12
nous2	1	2 787	0.4	2.5	0.029	5	1	0.7	11
nuclear10a	–	48	–	limit	–	0	0	606.1	18
nuclear10b	1	62	1889.4	limit	53.089	0	1	1869.1	21
nuclear14a	1	10 945	511.3	limit	15.269	0	1	840.4	21
nuclear14b	1	15 156	106.3	limit	7.488	0	1	204.0	21
nuclear14	1	1 128 610	0.6	limit	0.616	0	1	84.3	11
nuclear24a	1	10 965	510.4	limit	15.209	0	1	840.5	21
nuclear24b	1	15 226	104.9	limit	7.462	0	1	202.2	21
nuclear24	1	1 127 617	0.6	limit	0.616	0	1	84.6	11
nuclear25a	–	8 852	–	limit	100.000	0	0	395.5	17
nuclear25b	20	12 547	329.0	limit	9.139	0	1	1018.4	21
nuclear25	16	1 338 236	0.9	limit	1.139	0	1	2.8	4
nuclear49a	–	1 582	–	limit	100.000	0	0	426.8	17
nuclear49b	1	769	758.3	limit	21.056	0	1	992.7	21
nuclearva	–	4 584 696	–	limit	100.000	0	0	6.3	4
nuclearvb	–	4 648 061	–	limit	100.000	0	0	7.6	4
nuclearvc	–	4 675 160	–	limit	100.000	0	0	6.8	4
nuclearvd	–	4 108 627	–	limit	100.000	0	0	6.6	4
nuclearve	–	4 115 026	–	limit	100.000	0	0	7.2	4
nuclearvf	–	4 097 672	–	limit	100.000	0	0	6.3	4
nvs03	1	1	0.0	0.0	0.000	1	3	0.0	6
nvs10	0	1	0.0	0.0	0.000	0	7	0.0	6
nvs11	0	3	0.0	0.0	0.000	1	9	0.0	9
nvs12	0	5	0.0	0.0	0.000	1	12	0.0	8
nvs13	0	8	0.0	0.1	0.000	1	20	0.0	8
nvs14	1	1	0.0	0.0	0.000	1	0	0.0	6
nvs15	0	7	0.0	0.0	0.000	0	11	0.0	10
nvs17	0	47	0.0	0.1	0.001	0	30	0.1	17

Table B.24 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
nvs18	0	23	0.0	0.1	0.000	0	22	0.1	15
nvs19	0	84	0.0	0.2	0.003	0	17	0.1	17
nvs23	0	110	0.0	0.4	0.003	0	19	0.2	17
nvs24	0	122	0.0	0.5	0.003	1	21	0.2	17
prob02	1	1	0.0	0.1	0.001	0	5	0.1	6
prob03	0	1	0.0	0.0	0.000	0	4	0.0	5
product2	1	2 502 388	1.1	limit	0.031	0	184	226.4	22
product	1 360	17 567	12.4	40.0	0.345	2	5	9.7	22
eniplac_reformulated	19	121	0.3	0.8	0.009	0	7	0.4	18
fo7_2_reformulated	161	56 712	0.8	25.2	0.022	2	1	3.9	19
fo7_reformulated	540	186 185	1.9	77.1	0.068	1	3	7.7	19
fo8_reformulated	326	484 270	1.4	191.2	0.328	3	6	17.3	19
fo9_reformulated	286	2 194 028	2.0	842.5	0.692	10	8	44.8	20
m3_reformulated	1	19	0.1	0.1	0.003	2	2	0.1	11
m6_reformulated	47	6 278	0.4	3.2	0.018	2	7	1.4	19
m7_reformulated	1	6 574	0.6	5.2	0.052	5	8	2.3	20
o7_2_reformulated	136	1 630 739	1.1	740.0	0.133	6	5	48.7	20
o7_reformulated	12 790	3 224 610	12.3	1517.0	0.444	5	1	80.3	19
sep1	1	23	0.1	0.4	0.003	0	1	0.4	10
space25a	–	11 840	–	limit	100.000	0	0	9.3	19
space25	192 540	253 865	472.1	limit	46.919	0	1	4.3	20
space960	1	4 277	58.5	limit	53.354	0	1	1621.6	20
spectra2	1	23	0.1	0.8	0.006	0	61	0.4	14
st_e13	1	1	0.0	0.0	0.000	1	2	0.0	4
st_e27	0	1	0.0	0.0	0.000	0	3	0.0	8
st_e31	1	1 815	0.2	1.2	0.010	3	3	0.8	19
st_miqp1	0	1	0.0	0.0	0.000	0	3	0.0	1
st_miqp2	1	1	0.0	0.0	0.000	0	6	0.0	7
st_miqp3	0	1	0.0	0.0	0.000	0	6	0.0	3
st_miqp4	0	1	0.0	0.0	0.000	0	5	0.0	8
st_miqp5	0	1	0.0	0.0	0.000	0	3	0.0	5
st_test1	0	0	0.0	0.0	0.000	0	2	0.0	1
st_test2	0	1	0.0	0.0	0.000	0	3	0.0	1
st_test3	0	1	0.0	0.0	0.000	0	3	0.0	1
st_test4	1	1	0.0	0.0	0.000	0	3	0.0	7
st_test5	1	1	0.0	0.0	0.000	0	1	0.0	1
st_test6	1	1	0.0	0.0	0.000	0	1	0.0	1
st_test8	1	1	0.0	0.0	0.000	0	4	0.0	7
st_testgr1	0	41	0.0	0.1	0.000	2	28	0.0	18
st_testgr3	0	13	0.0	0.0	0.000	1	14	0.0	10
st_testph4	0	1	0.0	0.0	0.000	1	5	0.0	6
tln12	1 175	1 219 631	11.3	limit	2.150	2	221	435.4	19
tln2	1	1	0.0	0.0	0.000	0	5	0.0	6
tln4	1	4 374	0.0	1.8	0.001	0	5	0.5	18
tln5	1	133 870	0.0	73.7	0.005	0	164	5.2	18
tln6	1	6 192 081	0.1	limit	0.011	0	599	526.4	18
tln7	1	4 410 158	0.0	limit	0.755	2	491	597.0	18
tloss	28	120	0.1	0.2	0.003	1	2	0.0	15
tltr	1	8	0.0	0.2	0.000	1	2	0.0	10
util	1	112	0.0	0.4	0.000	5	7	0.3	15
waste	1	878 569	0.1	limit	4.316	0	55	573.1	22
Sarawak_Scenario16	1	772 542	0.1	limit	0.567	0	123	368.7	21
Sarawak_Scenario81	1	201 802	0.9	limit	0.153	0	36	229.1	19

Table B.24 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
lee1	9	4 752	0.5	4.5	0.017	3	10	1.5	21
lee2	43 116	61 438	98.6	206.7	3.449	1	9	19.2	21
meyer04	–	330 672	–	limit	100.000	0	0	27.2	19
meyer10	59 450	225 772	288.0	limit	49.023	0	1	47.0	21
meyer15	12 370	262 683	56.7	limit	56.489	0	4	152.8	22
ahmetovic1_pw4	95	31 459	1.1	28.1	0.126	1	6	4.1	22
ahmetovic2_pw4	41 170	1 045 427	142.6	limit	5.937	0	17	233.7	22
karuppiah1	2 000	2 895 354	1.7	1170.9	0.047	9	2	14.2	4
karuppiah2_pw4	1	5 319 123	0.8	limit	0.042	0	155	50.6	19
karuppiah3_pw4	4 270	4 620 934	4.7	limit	0.131	0	35	44.1	13
karuppiah4_pw4	1	1 583 631	0.5	limit	0.057	0	36	91.5	14
ruiz_concbased_pw4	1	39 437	0.2	21.7	0.015	3	35	4.9	19

Table B.25.: Performance of SCIP 3.0.2, primal heuristics deactivated, on the GloMIQO test set

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
CLay0203M	13	32	0.1	0.1	0.002	3
CLay0204M	9	1 032	0.1	0.4	0.003	7
CLay0205M	11	9 608	0.1	2.5	0.007	15
CLay0303M	8	83	0.1	0.1	0.003	2
CLay0304M	8	305	0.1	0.3	0.003	10
CLay0305M	10	13 407	0.2	3.6	0.007	15
SLay04H	31	97	0.6	0.8	0.019	3
SLay04M	6	134	0.1	0.3	0.004	4
SLay05H	49	222	0.8	1.3	0.027	6
SLay05M	29	105	0.2	0.5	0.008	4
SLay06H	330	1 669	2.3	6.4	0.080	15
SLay06M	51	277	0.4	0.9	0.013	4
SLay07H	251	1 259	2.8	6.5	0.099	7
SLay07M	134	314	0.7	1.3	0.021	7
SLay08H	858	8 665	8.4	43.1	0.331	15
SLay08M	131	2 797	1.0	6.7	0.041	10
SLay09H	2 271	36 424	23.6	239.8	1.584	10
SLay09M	273	32 039	1.7	94.2	0.469	10
SLay10H	6 054	466 126	66.7	limit	7.127	6
SLay10M	444	836 132	2.7	2217.9	0.998	14
LeeCrudeOil1_05	14	29	1.1	1.1	0.031	4
LeeCrudeOil1_06	14	36	1.4	1.9	0.039	2
LeeCrudeOil1_07	72	98	3.1	3.2	0.086	4
LeeCrudeOil1_08	13	93	2.7	5.8	0.075	3
LeeCrudeOil1_09	20	150	5.4	7.6	0.150	5
LeeCrudeOil1_10	105	191	8.4	9.2	0.233	3
LeeCrudeOil2_05	13	87	2.0	3.6	0.056	2
LeeCrudeOil2_06	30	51	4.0	4.7	0.111	2
LeeCrudeOil2_07	9	253	6.4	9.8	0.186	8
LeeCrudeOil2_08	4	899	6.0	21.8	0.185	5
LeeCrudeOil2_09	189	627	13.9	19.1	0.397	5
LeeCrudeOil2_10	33	1 004	12.5	35.0	0.378	4
LeeCrudeOil3_05	53	85 909	3.8	94.6	0.107	20
LeeCrudeOil3_06	141	2 451 810	11.0	limit	0.306	21
LeeCrudeOil3_07	78	1 920 787	16.7	limit	0.467	19
LeeCrudeOil3_08	213	1 515 461	22.7	limit	0.643	18
LeeCrudeOil3_09	924	1 338 928	47.1	limit	1.321	17
LeeCrudeOil3_10	285	1 064 429	39.1	limit	1.144	19
LeeCrudeOil4_05	68	168	6.6	7.2	0.183	8
LeeCrudeOil4_06	22	81	4.8	5.0	0.133	6
LeeCrudeOil4_07	16	113	6.9	9.4	0.192	11
LeeCrudeOil4_08	35	141	17.9	20.4	0.497	6
LeeCrudeOil4_09	91	125	17.7	18.0	0.492	5
LeeCrudeOil4_10	65	290	27.6	39.2	0.767	8
LiCrudeOil_ex01	9 143	1 479 073	17.5	limit	0.785	1
LiCrudeOil_ex02	1 875	2 118 477	7.7	limit	0.309	19
LiCrudeOil_ex03	43 139	469 077	346.2	limit	10.191	1
LiCrudeOil_ex05	524 209	524 209	3600.0	limit	100.000	0
LiCrudeOil_ex06	9 486	33 712	127.9	308.8	3.571	2

Table B.25 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
LiCrudeOil_ex11	56 727	439 765	560.2	limit	15.771	1
LiCrudeOil_ex21	110 468	330 326	1312.3	limit	36.854	1
alan	3	7	0.0	0.0	0.000	3
du-opt5	6	61	0.1	0.2	0.005	5
du-opt	94	312	0.3	0.6	0.009	5
elf	113	1 070	0.1	0.7	0.014	5
ex1223a	3	4	0.0	0.0	0.000	2
ex1263a	10	190	0.0	0.1	0.001	8
ex1263	5	615	0.2	0.8	0.009	17
ex1264a	17	67	0.0	0.0	0.000	3
ex1264	9	340	0.1	0.3	0.004	12
ex1265a	10	112	0.0	0.1	0.000	4
ex1265	3	350	0.0	0.5	0.002	9
ex1266a	43	51	0.1	0.1	0.003	3
ex1266	8	167	0.4	1.0	0.018	5
fac3	4	15	0.1	0.1	0.003	5
feedtray2	170	181	1.1	1.2	0.032	3
fuel	3	5	0.0	0.0	0.000	3
gbd	1	1	0.0	0.0	0.000	1
meanvarx	4	5	0.0	0.0	0.000	1
netmod_dol1	10	53 332	13.6	2040.0	0.382	2
netmod_dol2	15	285	15.8	37.1	0.460	15
netmod_kar1	11	283	0.8	3.1	0.024	10
netmod_kar2	11	283	0.9	3.2	0.027	10
nous1	86 802	1 829 516	102.0	2628.6	10.773	10
nous2	1 952	5 865	1.9	4.5	0.058	14
nuclear10a	–	37	–	limit	–	0
nuclear10b	–	147	–	limit	100.000	0
nuclear14a	–	10 674	–	limit	100.000	0
nuclear14b	–	25 515	–	limit	100.000	0
nuclear14	–	195 840	–	limit	100.000	0
nuclear24a	–	10 671	–	limit	100.000	0
nuclear24b	–	25 505	–	limit	100.000	0
nuclear24	–	195 613	–	limit	100.000	0
nuclear25a	–	10 061	–	limit	100.000	0
nuclear25b	–	14 292	–	limit	100.000	0
nuclear25	–	196 328	–	limit	100.000	0
nuclear49a	–	1 332	–	limit	100.000	0
nuclear49b	–	1 336	–	limit	100.000	0
nuclearva	–	4 418 515	–	limit	100.000	0
nuclearvb	–	4 500 948	–	limit	100.000	0
nuclearvc	–	4 512 700	–	limit	100.000	0
nuclearvd	–	4 000 320	–	limit	100.000	0
nuclearve	–	3 955 061	–	limit	100.000	0
nuclearvf	–	3 977 076	–	limit	100.000	0
nvs03	1	1	0.0	0.0	0.000	1
nvs10	1	1	0.0	0.0	0.000	1
nvs11	4	9	0.0	0.0	0.000	2
nvs12	5	13	0.0	0.0	0.000	2
nvs13	6	19	0.0	0.0	0.000	2
nvs14	1	1	0.0	0.0	0.000	1
nvs15	3	8	0.0	0.0	0.000	2
nvs17	63	89	0.1	0.1	0.003	3

Table B.25 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
nvs18	40	52	0.1	0.1	0.001	3
nvs19	44	131	0.1	0.2	0.003	3
nvs23	94	170	0.2	0.3	0.006	3
nvs24	34	151	0.1	0.3	0.003	3
prob02	1	1	0.0	0.0	0.000	1
prob03	1	1	0.0	0.0	0.000	1
product2	–	1 748 631	–	limit	100.000	0
product	9 674	26 241	24.7	43.6	0.686	6
eniplac_reformulated	67	159	0.3	0.5	0.008	5
fo7_2_reformulated	3 014	88 497	2.3	29.1	0.067	2
fo7_reformulated	13 058	248 200	7.9	81.2	0.251	6
fo8_reformulated	66 528	606 546	49.6	247.3	1.383	6
fo9_reformulated	46 424	10 544 024	39.0	3387.7	2.132	15
m3_reformulated	10	19	0.0	0.0	0.000	2
m6_reformulated	910	3 028	0.8	1.6	0.029	7
m7_reformulated	9 645	51 141	5.8	19.3	0.178	3
o7_2_reformulated	11 314	1 807 330	7.7	798.2	0.280	7
o7_reformulated	96 221	2 096 520	56.5	896.6	1.743	11
sep1	27	35	0.0	0.0	0.000	2
space25a	–	15 849	–	limit	100.000	0
space25	–	104 511	–	limit	100.000	0
space960	–	6 435	–	limit	100.000	0
spectra2	42	68	0.8	0.8	0.022	2
st_e13	2	3	0.0	0.0	0.000	2
st_e27	1	1	0.0	0.0	0.000	1
st_e31	501	1 093	0.2	0.7	0.007	11
st_miqp1	1	1	0.0	0.0	0.000	1
st_miqp2	1	1	0.0	0.0	0.000	1
st_miqp3	2	3	0.0	0.0	0.000	2
st_miqp4	1	1	0.0	0.0	0.000	1
st_miqp5	1	1	0.0	0.0	0.000	1
st_test1	1	1	0.0	0.0	0.000	1
st_test2	1	1	0.0	0.0	0.000	1
st_test3	1	1	0.0	0.0	0.000	1
st_test4	1	1	0.0	0.0	0.000	1
st_test5	1	1	0.0	0.0	0.000	1
st_test6	1	1	0.0	0.0	0.000	1
st_test8	1	1	0.0	0.0	0.000	1
st_testgr1	15	56	0.0	0.0	0.000	4
st_testgr3	3	9	0.0	0.0	0.000	1
st_testph4	1	1	0.0	0.0	0.000	1
tln12	295 778	501 978	2146.6	limit	73.162	1
tln2	1	1	0.0	0.0	0.000	1
tln4	292	3 885	0.2	1.6	0.008	8
tln5	2 031	792 957	1.5	381.1	0.068	10
tln6	57 214	5 190 580	39.2	limit	1.692	10
tln7	111 692	2 805 464	129.5	limit	8.336	8
tloss	30	204	0.1	0.2	0.004	3
tltr	9	14	0.2	0.2	0.005	1
util	23	372	0.0	0.1	0.000	4
waste	–	1 045 149	–	limit	100.000	0
Sarawak_Scenario16	–	763 728	–	limit	100.000	0
Sarawak_Scenario81	–	166 217	–	limit	100.000	0

Table B.25 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
lee1	322	8 982	0.5	7.3	0.030	4
lee2	89 113	122 342	334.9	638.0	9.942	8
meyer04	–	334 995	–	limit	100.000	0
meyer10	240 212	253 684	3140.3	limit	95.479	1
meyer15	33 379	237 163	158.4	limit	32.742	1
ahmetovic1_pw4	4 104	34 536	3.1	26.6	0.118	5
ahmetovic2_pw4	81 310	1 047 302	251.9	limit	34.998	1
karuppiah1	4 343	4 360 592	2.6	1555.0	0.073	14
karuppiah2_pw4	27 657	4 059 914	24.4	limit	3.697	1
karuppiah3_pw4	–	3 443 288	–	limit	100.000	0
karuppiah4_pw4	–	892 330	–	limit	100.000	0
ruiz_concbased_pw4	7 639	92 571	5.2	65.2	0.232	17

Table B.26.: Performance of SCIP 3.0.2, default mode, on the MinlpLib test set

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
4stufen	–	2 250 551	–	limit	100.000	0	0	25.2	19
alan	1	3	0.0	0.1	0.000	1	3	0.0	11
batchdes	1	3	0.0	0.1	0.000	0	2	0.1	11
batch	1	5	0.0	0.2	0.000	0	1	0.1	12
beuster	6 440	2 053 783	10.5	limit	3.771	0	1	80.5	21
cecil_13	26	1 087 040	3.6	limit	0.143	0	12	913.5	21
chp_partload	–	7 655	–	limit	100.000	0	0	40.1	18
contvar	–	20 726	–	limit	100.000	0	0	11.9	12
csched1	1	1 392	0.1	1.8	0.003	5	4	0.9	20
csched2a	1	532 905	0.4	limit	22.862	0	1	66.0	22
csched2	2	133 399	2.2	limit	9.714	0	1	162.0	20
detf1	1	38	2289.2	limit	65.620	0	1	1829.2	21
du-opt	1	232	0.0	0.7	0.006	2	108	0.4	19
du-opt5	1	75	0.0	0.4	0.005	2	93	0.3	20
eg_all_s	0	81	0.1	limit	44.260	0	2	649.7	20
eg_disc2_s	1	1	3600.0	limit	100.000	0	1	3498.6	10
eg_disc_s	1	1	3600.0	limit	100.000	0	1	3520.0	10
eg_int_s	0	1	0.1	limit	99.994	0	1	3493.7	9
eniplac	2	45	0.1	0.5	0.003	0	6	0.2	15
enpro48	1	337	0.1	1.2	0.015	1	2	0.8	12
enpro48pb	1	33	0.1	1.5	0.023	0	3	1.2	13
enpro56	1	124	0.1	1.5	0.004	1	4	1.1	13
enpro56pb	1	85	0.1	1.3	0.004	0	3	0.9	12
ex1221	1	1	0.0	0.0	0.000	1	2	0.0	4
ex1222	1	1	0.0	0.0	0.000	0	3	0.0	1
ex1223a	0	1	0.0	0.0	0.000	0	4	0.0	7
ex1223b	0	4	0.0	0.1	0.001	1	9	0.1	10
ex1223	1	4	0.0	0.1	0.001	1	8	0.0	10
ex1224	1	6	0.1	0.1	0.003	1	3	0.1	11
ex1225	1	1	0.0	0.0	0.000	0	3	0.0	8
ex1226	0	0	0.0	0.0	0.000	0	3	0.0	7
ex1233	1	9 981 610	0.1	limit	0.134	0	307	179.6	21
ex1243	1	29	0.1	0.5	0.009	1	3	0.3	14
ex1244	1	392	0.1	1.2	0.005	2	5	0.7	21
ex1252a	1	1 095	0.0	6.6	0.023	4	3	0.8	20
ex1252	9	2 366	0.2	12.0	0.020	3	10	0.8	20
ex1263	1	398	0.1	0.8	0.005	9	5	0.4	18
ex1263a	1	308	0.0	0.4	0.001	5	9	0.3	18
ex1264	17	79	0.1	0.2	0.003	6	0	0.0	10
ex1264a	1	314	0.0	0.3	0.001	2	3	0.1	18
ex1265	3	132	0.1	0.3	0.004	8	0	0.0	10
ex1265a	1	105	0.0	0.1	0.001	2	4	0.0	10
ex1266	1	63	0.2	0.6	0.006	2	2	0.1	10
ex1266a	56	166	0.2	0.4	0.006	2	5	0.2	18
ex3	1	5	0.0	0.2	0.001	0	8	0.2	11
ex3pb	1	5	0.0	0.2	0.001	0	8	0.2	11
fac1	1	5	0.0	0.0	0.000	0	15	0.0	11
fac2	1	6 310 317	0.0	limit	0.007	25	108	20.6	20
fac3	1	5	0.0	0.2	0.001	1	4	0.1	11

Table B.26 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
feedtray2	1	1	0.1	0.2	0.003	0	1	0.1	4
feedtray	1	226 155	0.5	limit	36.747	0	1	16.5	11
fo7_2	1	51 571	0.1	23.0	0.023	5	4	3.5	19
fo7_ar2_1	2 153	51 730	3.2	22.6	0.125	8	1	3.6	20
fo7_ar25_1	23	30 506	1.1	12.1	0.042	7	2	2.6	20
fo7_ar3_1	73	44 473	1.0	18.4	0.039	4	5	2.7	20
fo7_ar4_1	288	34 824	1.4	15.1	0.062	4	3	2.5	19
fo7_ar5_1	113	28 733	1.3	14.8	0.078	12	5	2.9	19
fo7	161	175 201	1.0	74.5	0.091	8	11	5.7	20
fo8_ar2_1	1 419	401 103	2.5	169.3	0.325	11	14	19.1	20
fo8_ar25_1	10 450	293 826	9.7	118.3	0.387	8	3	11.8	20
fo8_ar3_1	4 978	73 630	5.9	32.2	0.255	4	9	4.8	20
fo8_ar4_1	1 539	83 082	2.8	40.3	0.217	9	8	5.9	20
fo8_ar5_1	73	69 761	0.9	35.1	0.095	7	12	5.8	20
fo8	1 200	349 233	3.0	184.7	0.429	8	12	15.7	20
fo9_ar2_1	51	16 650 300	1.5	limit	0.230	11	6	75.8	20
fo9_ar25_1	20 186	5 170 431	20.5	2148.9	0.758	12	12	97.9	20
fo9_ar3_1	5 283	553 645	7.2	295.1	2.270	12	10	42.4	20
fo9_ar4_1	100 661	1 441 308	95.3	823.7	7.376	17	18	96.5	20
fo9_ar5_1	14 247	606 901	18.6	335.9	2.125	12	7	35.1	20
fo9	10 447	822 626	15.9	472.3	0.602	6	5	35.6	20
fuel	1	3	0.0	0.1	0.000	1	5	0.1	11
fuzzy	–	839 611	–	limit	100.000	0	0	35.7	19
gastrans	1	1	0.2	0.2	0.006	0	1	0.1	10
gbd	0	1	0.0	0.0	0.000	1	2	0.0	3
gear2	1	1 009	0.0	0.7	0.019	9	3	0.5	22
gear3	0	126	0.0	0.4	0.010	5	7	0.3	18
gear	0	126	0.0	0.3	0.007	5	7	0.3	18
ghg_1veh	1	9 981 674	0.1	limit	0.007	0	5	101.3	12
ghg_2veh	1	347 808	0.1	limit	0.010	0	153	125.8	20
ghg_3veh	9	160 641	0.8	limit	0.276	0	110	222.0	22
gkocis	0	0	0.0	0.0	0.000	0	10	0.0	11
hda	42 000	323 776	473.0	limit	17.407	0	2	28.5	21
hmittelman	1	1	0.0	0.0	0.000	1	1	0.0	2
johnall	1	1	8.9	77.8	0.247	0	2	5.5	8
lop97ic	1	122 232	7.7	limit	22.092	0	1	761.2	16
lop97icx	1	1 965 260	0.1	limit	3.517	7	10	780.8	17
m3	1	14	0.1	0.1	0.003	1	1	0.1	12
m6	29	797	0.2	1.6	0.023	3	7	1.1	19
m7_ar2_1	674	7 605	1.2	4.2	0.037	2	8	1.4	21
m7_ar25_1	4	2 221	0.1	1.5	0.008	0	2	0.7	19
m7_ar3_1	221	9 768	0.9	5.1	0.044	4	6	1.5	19
m7_ar4_1	2	863	0.1	1.8	0.022	2	6	1.0	19
m7_ar5_1	1	14 005	0.1	7.1	0.038	2	7	1.7	20
m7	125	5 043	0.4	4.0	0.038	5	6	1.7	19
mbtd	1	1	843.2	limit	58.488	0	1	455.5	7
meanvarx	1	1	0.0	0.2	0.000	1	4	0.1	12
meanvarxsc	1	1	0.0	0.2	0.002	1	5	0.2	12
netmod_dol1	1	35 247	0.3	limit	0.584	10	9	148.3	21
netmod_dol2	1	154	0.2	46.4	0.216	7	3	0.7	11
netmod_kar1	1	341	0.1	6.2	0.046	9	5	0.3	18
netmod_kar2	1	341	0.1	6.3	0.048	9	5	0.2	18
no7_ar2_1	1 952	33 387	3.1	17.9	0.086	4	1	2.6	20

Table B.26 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
no7_ar25_1	40	73 107	1.2	40.5	0.052	5	6	4.7	20
no7_ar3_1	1 166	451 991	2.2	201.2	0.100	7	5	12.6	20
no7_ar4_1	315	159 212	1.1	80.7	0.106	12	3	8.0	19
no7_ar5_1	30	101 216	0.5	51.6	0.033	9	5	5.9	20
nous1	1	1 651 075	0.4	2019.1	0.064	2	2	15.0	11
nous2	1	4 240	0.3	3.6	0.043	6	2	1.0	11
nuclearva	–	4 855 948	–	limit	100.000	0	0	7.6	4
nuclearvb	–	4 729 199	–	limit	100.000	0	0	7.5	4
nuclearvc	–	4 709 101	–	limit	100.000	0	0	7.1	4
nuclearvd	–	4 155 854	–	limit	100.000	0	0	6.9	4
nuclearve	–	4 144 354	–	limit	100.000	0	0	6.6	4
nuclearvf	–	4 180 928	–	limit	100.000	0	0	6.6	4
nuclear25	16	1 357 317	0.7	limit	1.134	0	1	2.5	4
nuclear25a	–	11 175	–	limit	100.000	0	0	441.2	17
nuclear25b	1	13 627	75.0	limit	5.946	0	1	684.9	20
nuclear49a	–	1 242	–	limit	100.000	0	0	502.1	18
nuclear49b	1	1	3600.0	limit	100.000	0	1	3586.4	11
nuclear14	–	1 408 586	–	limit	100.000	0	0	4.2	4
nuclear14a	–	11 951	–	limit	100.000	0	0	455.6	18
nuclear14b	1	39 096	281.4	limit	8.675	0	1	620.9	20
nuclear10a	–	40	–	limit	–	0	0	680.8	18
nuclear10b	1	1	3600.0	limit	100.000	0	1	3425.0	11
nvs01	1	9	0.0	0.1	0.002	1	3	0.0	12
nvs02	1	1	0.0	0.0	0.000	1	0	0.0	6
nvs03	1	1	0.0	0.0	0.000	1	3	0.0	6
nvs04	1	3	0.0	0.0	0.000	1	6	0.0	12
nvs06	0	11	0.0	0.1	0.003	1	6	0.1	10
nvs07	0	1	0.0	0.0	0.000	0	5	0.0	7
nvs08	0	1	0.0	0.0	0.000	0	5	0.0	10
nvs09	0	2 440 089	0.0	770.2	0.003	0	4	11.2	15
nvs10	0	1	0.0	0.0	0.000	0	7	0.0	6
nvs11	0	3	0.0	0.0	0.000	1	9	0.0	9
nvs12	0	5	0.0	0.1	0.000	1	12	0.0	8
nvs13	0	8	0.0	0.1	0.000	1	20	0.0	8
nvs14	1	1	0.0	0.0	0.000	1	0	0.0	6
nvs15	0	7	0.0	0.0	0.000	0	11	0.0	10
nvs17	0	47	0.0	0.1	0.000	0	30	0.1	17
nvs18	0	23	0.0	0.1	0.000	0	22	0.1	15
nvs19	0	90	0.0	0.3	0.003	0	14	0.2	17
nvs20	1	105	0.0	0.5	0.003	2	4	0.4	13
nvs21	0	20	0.0	0.1	0.002	4	2	0.0	10
nvs23	0	122	0.0	0.4	0.001	1	23	0.2	17
nvs24	0	114	0.0	0.4	0.003	0	23	0.2	17
o7_2	127	1 582 170	0.8	758.8	0.149	8	7	44.1	20
o7_ar2_1	1 707	1 282 888	3.0	258.4	0.085	5	5	13.3	20
o7_ar25_1	136	555 882	1.0	268.1	0.062	16	5	16.6	21
o7_ar3_1	1 407	1 038 314	2.4	535.1	0.109	8	3	28.7	20
o7_ar4_1	2 244	1 855 541	3.4	1016.9	0.231	7	3	52.2	20
o7_ar5_1	390	632 962	1.4	301.1	0.093	6	4	18.2	20
o7	4 550	3 757 628	6.6	1936.7	0.277	7	8	100.8	20
o8_ar4_1	372	5 765 577	2.0	limit	1.525	5	18	220.4	20
o9_ar4_1	217 600	5 708 704	221.0	limit	7.660	2	2	244.6	21
oaer	0	1	0.0	0.0	0.000	1	5	0.0	8

Table B.26 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
oil2	1	1 103 885	3.0	limit	0.089	0	5	164.3	22
oil	8	76 592	2.8	limit	1.239	0	7	322.6	23
ortez	1	34	0.2	0.5	0.007	0	4	0.3	12
parallel	1	428 168	0.1	limit	0.095	0	2309	99.7	16
pb302035	1	553 909	11.4	limit	24.653	0	12	1281.5	17
pb302055	1	549 602	12.7	limit	19.844	0	24	1207.3	17
pb302075	1	660 738	12.9	limit	11.346	0	1	1141.3	16
pb302095	1	599 357	9.2	limit	3.355	0	19	984.5	17
pb351535	1	663 503	7.3	limit	17.496	0	18	1362.6	17
pb351555	1	802 305	7.6	limit	12.494	0	2	1243.1	16
pb351575	1	677 619	10.4	limit	8.187	0	1	1216.1	16
pb351595	1 242	711 133	26.0	limit	10.720	1	3	1054.1	17
prob02	1	1	0.0	0.1	0.001	0	5	0.1	6
prob03	0	1	0.0	0.0	0.000	0	4	0.0	5
procsel	0	0	0.0	0.0	0.000	0	12	0.0	10
product2	1	2 524 774	0.9	limit	0.025	0	187	175.0	21
product	1	42 542	2.3	87.3	0.084	2	4	14.6	21
pump	1	1 003	0.0	4.7	0.013	4	3	0.8	20
qapw	1	428 574	1.7	limit	1.130	1	19	141.0	18
qap	1	1 900 639	1.1	limit	6.388	7	73	29.7	18
ravem	1	42	0.1	0.9	0.003	1	2	0.6	12
ravempb	1	41	0.1	0.9	0.004	1	2	0.6	12
risk2b	0	184	0.0	0.8	0.017	6	3	0.7	11
risk2bpb	1	8	0.1	0.2	0.003	2	2	0.1	11
saa_2	1	37	2299.7	limit	64.420	0	1	1837.6	21
sep1	1	19	0.1	0.3	0.003	0	1	0.3	10
space25	–	8 190	–	limit	100.000	0	0	1.9	18
space25a	55 440	250 802	43.7	limit	15.741	0	4	5.4	21
space960	–	4 262	–	limit	100.000	0	0	1397.7	18
spectra2	1	13	0.1	0.9	0.005	0	53	0.4	15
spring	22	83	0.2	0.5	0.007	1	1	0.4	17
st_e13	1	1	0.0	0.0	0.000	1	2	0.0	4
st_e14	1	4	0.0	0.1	0.001	1	8	0.0	10
st_e15	1	1	0.0	0.0	0.000	1	2	0.0	4
st_e27	0	0	0.0	0.0	0.000	0	2	0.0	2
st_e29	1	6	0.1	0.1	0.003	1	3	0.1	11
st_e31	1	1 752	0.3	1.3	0.013	3	4	0.8	17
st_e35	1	12 301	0.2	14.2	0.101	9	7	2.5	21
st_e36	1	790	0.0	1.0	0.011	1	1	0.6	16
st_e38	1	2	0.0	0.1	0.000	0	2	0.1	11
st_e40	1	29	0.0	0.1	0.000	1	1	0.1	12
st_miqp1	0	1	0.0	0.0	0.000	0	3	0.0	2
st_miqp2	1	1	0.0	0.0	0.000	1	4	0.0	6
st_miqp3	0	5	0.0	0.0	0.000	1	3	0.0	9
st_miqp4	0	1	0.0	0.0	0.000	0	5	0.0	7
st_miqp5	0	1	0.0	0.0	0.000	0	2	0.0	5
stockcycle	1	21 207	0.2	106.8	0.098	0	57	14.8	21
st_test1	0	0	0.0	0.0	0.000	0	2	0.0	1
st_test2	0	1	0.0	0.0	0.000	0	3	0.0	1
st_test3	0	1	0.0	0.0	0.000	0	3	0.0	2
st_test4	1	1	0.0	0.0	0.000	0	2	0.0	6
st_test5	1	1	0.0	0.0	0.000	0	1	0.0	2
st_test6	1	1	0.0	0.0	0.000	0	1	0.0	2

Table B.26 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
st_test8	1	1	0.0	0.0	0.000	0	4	0.0	7
st_testgr1	0	29	0.0	0.0	0.000	0	24	0.0	14
st_testgr3	0	13	0.0	0.1	0.000	1	14	0.0	10
st_testph4	0	1	0.0	0.0	0.000	1	5	0.0	6
super1	–	61 664	–	limit	100.000	0	0	8.9	19
super2	–	52 241	–	limit	100.000	0	0	9.9	19
super3	–	53 440	–	limit	100.000	0	0	9.2	19
super3t	4 840	32 686	288.9	limit	11.705	0	1	20.9	15
synheat	1	10 154 865	0.1	limit	0.033	0	156	103.4	22
synthes1	1	5	0.0	0.1	0.000	1	7	0.0	10
synthes2	1	5	0.0	0.1	0.000	0	35	0.1	12
synthes3	1	7	0.0	0.1	0.001	0	44	0.1	11
tln2	1	1	0.0	0.0	0.000	0	5	0.0	6
tln4	1	4 374	0.0	1.9	0.001	0	5	0.6	18
tln5	1	300 404	0.0	159.3	0.004	0	261	10.1	18
tln6	1	6 240 641	0.0	limit	0.010	0	617	528.3	18
tln7	1	4 393 651	0.0	limit	0.980	0	445	535.8	18
tln12	610	1 083 282	9.5	limit	1.750	5	228	448.0	20
tloss	28	120	0.1	0.2	0.003	1	2	0.0	15
tls2	1	17	0.0	0.1	0.000	1	2	0.0	12
tls4	1	23 438	1.0	40.6	0.068	1	7	3.6	20
tls5	22 968	199 462	150.6	limit	11.916	0	17	58.8	20
tls6	31 030	705 220	175.0	limit	6.089	0	1	284.2	18
tls7	11 722	548 491	89.9	limit	14.224	0	2	264.1	18
tls12	–	187 314	–	limit	100.000	0	0	668.4	16
tltr	1	11	0.0	0.2	0.000	1	4	0.0	10
uselinear	–	4 827	–	limit	100.000	0	0	1.1	3
util	1	83	0.0	0.3	0.000	4	5	0.2	13
waste	1	1 616 106	0.1	limit	8.865	0	44	394.6	22
water4	1	1 881 133	0.1	limit	2.164	0	39	110.2	23
waterx	1	79 034	0.1	limit	0.962	0	3	38.1	20
waterz	235	2 188 446	2.4	limit	25.995	1	11	207.7	23

Table B.27.: Performance of SCIP 3.0.2, primal heuristics deactivated, on the MinlpLib test set

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
4stufen	–	2 318 220	–	limit	100.000	0
alan	3	7	0.0	0.0	0.000	3
batchdes	2	3	0.0	0.0	0.000	1
batch	14	15	0.1	0.1	0.002	1
beuster	–	1 747 546	–	limit	100.000	0
cecil_13	80	1 484 711	2.6	limit	0.132	7
chp_partload	–	5 691	–	limit	100.000	0
contvar	–	34 301	–	limit	100.000	0
csched1	130	3 043	0.2	1.8	0.006	15
csched2a	8 644	612 781	47.4	limit	29.873	1
csched2	15 675	84 590	324.5	limit	34.617	1
detf1	–	215	–	limit	100.000	0
du-opt	82	294	0.2	0.4	0.006	5
du-opt5	6	74	0.1	0.2	0.003	5
eg_all_s	–	1 189	–	limit	100.000	0
eg_disc2_s	–	49	–	limit	100.000	0
eg_disc_s	–	57	–	limit	100.000	0
eg_int_s	–	15	–	limit	100.000	0
eniplac	59	86	0.3	0.3	0.008	3
enpro48	40	72	0.5	0.5	0.014	1
enpro48pb	12	84	0.3	0.5	0.009	5
enpro56	26	42 421	0.4	7.6	0.011	23
enpro56pb	26	19 220 246	0.4	3062.0	0.011	50
ex1221	1	1	0.0	0.0	0.000	1
ex1222	1	1	0.0	0.0	0.000	1
ex1223a	1	1	0.0	0.0	0.000	1
ex1223b	4	7	0.0	0.0	0.000	2
ex1223	5	8	0.0	0.0	0.000	2
ex1224	5	18	0.0	0.0	0.000	5
ex1225	1	1	0.0	0.0	0.000	1
ex1226	4	5	0.0	0.0	0.000	2
ex1233	624 610	4 958 499	728.6	limit	20.405	5
ex1243	94	128	0.3	0.4	0.009	4
ex1244	64	327	0.3	0.7	0.009	3
ex1252a	60	1 174	1.7	9.2	0.096	9
ex1252	189	1 765	1.2	8.3	0.043	4
ex1263	9	405	0.1	0.5	0.004	8
ex1263a	10	190	0.0	0.1	0.001	8
ex1264	11	227	0.1	0.3	0.004	7
ex1264a	17	67	0.0	0.1	0.000	3
ex1265	3	89	0.1	0.3	0.004	11
ex1265a	10	112	0.0	0.1	0.000	4
ex1266	7	48	0.3	0.6	0.012	4
ex1266a	43	51	0.1	0.1	0.003	3
ex3	4	5	0.0	0.0	0.000	2
ex3pb	4	5	0.0	0.0	0.000	2
fac1	2	5	0.0	0.0	0.000	1
fac2	7 304	4 355 150	4.3	limit	64.479	10
fac3	4	15	0.1	0.1	0.003	5

Table B.27 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
feedtray2	107	107	0.8	0.8	0.022	1
feedtray	–	370 461	–	limit	100.000	0
fo7_2	2 849	54 732	2.3	21.4	0.077	5
fo7_ar2_1	4 455	31 187	2.6	10.7	0.073	10
fo7_ar25_1	2 314	59 410	1.8	21.0	0.051	4
fo7_ar3_1	2 675	44 047	2.1	15.9	0.063	8
fo7_ar4_1	7 983	50 953	5.5	21.6	0.204	5
fo7_ar5_1	3 654	31 100	3.0	14.1	0.109	9
fo7	15 978	175 693	12.3	75.5	0.367	4
fo8_ar2_1	17 542	258 337	11.7	96.7	0.375	21
fo8_ar25_1	44 677	287 709	27.5	112.2	0.814	9
fo8_ar3_1	33 318	74 333	21.6	37.2	0.653	14
fo8_ar4_1	6 901	97 706	5.1	40.3	0.274	13
fo8_ar5_1	4 107	157 882	3.5	70.7	0.280	19
fo8	16 644	506 947	13.3	225.5	0.546	10
fo9_ar2_1	12 204	3 239 857	9.7	1001.8	1.013	23
fo9_ar25_1	44 855	5 330 612	31.8	2198.2	1.211	16
fo9_ar3_1	12 012	110 290	9.7	50.0	0.407	5
fo9_ar4_1	178 618	770 081	130.2	419.6	5.516	11
fo9_ar5_1	86 049	1 131 071	69.5	578.7	3.656	18
fo9	146 907	1 705 027	161.4	989.7	4.500	4
fuel	3	5	0.0	0.0	0.000	3
fuzzy	–	673 728	–	limit	100.000	0
gastrans	2	2	0.1	0.1	0.003	1
gbd	1	1	0.0	0.0	0.000	1
gear2	8	1 032	0.0	0.3	0.008	8
gear3	5	429	0.0	0.1	0.003	10
gear	5	429	0.0	0.1	0.004	10
ghg_1veh	2 076	6 289 898	8.0	limit	0.232	5
ghg_2veh	16 040	372 904	145.3	limit	8.645	2
ghg_3veh	–	148 918	–	limit	100.000	0
gkocis	3	4	0.0	0.0	0.000	1
hda	–	318 104	–	limit	100.000	0
hmittelman	1	1	0.0	0.0	0.000	1
johnall	1	1	8.3	8.3	0.231	1
lop97ic	8 021	188 786	231.8	limit	26.645	1
lop97icx	2 013	2 489 190	4.9	limit	6.244	9
m3	10	19	0.0	0.0	0.000	2
m6	115	3 194	0.2	1.3	0.015	9
m7_ar2_1	1 857	17 266	1.4	4.6	0.046	79
m7_ar25_1	290	1 864	0.5	0.9	0.015	4
m7_ar3_1	2 723	12 896	1.9	4.3	0.058	9
m7_ar4_1	402	1 820	0.7	1.2	0.021	2
m7_ar5_1	2 560	11 493	1.8	4.7	0.059	11
m7	1 157	2 541	1.4	1.9	0.040	3
mbtd	–	1	–	limit	100.000	0
meanvarx	4	5	0.0	0.0	0.000	1
meanvarxsc	4	13	0.0	0.0	0.000	3
netmod_dol1	12	42 805	20.4	limit	0.614	9
netmod_dol2	14	164	16.3	48.0	0.461	7
netmod_kar1	10	268	1.1	5.3	0.036	4
netmod_kar2	10	268	1.2	5.5	0.039	4
no7_ar2_1	7 154	28 745	4.7	13.5	0.132	8

Table B.27 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
no7_ar25_1	4 688	97 433	3.6	45.1	0.114	7
no7_ar3_1	10 013	387 495	6.5	163.2	0.214	11
no7_ar4_1	6 292	179 281	4.2	79.2	0.156	12
no7_ar5_1	1 618	141 108	1.4	61.6	0.052	10
nous1	1 001 555	2 202 610	1596.5	limit	51.450	1
nous2	2 066	4 556	2.1	3.3	0.060	4
nuclearva	–	4 683 072	–	limit	100.000	0
nuclearvb	–	4 604 006	–	limit	100.000	0
nuclearvc	–	4 614 443	–	limit	100.000	0
nuclearvd	–	4 055 613	–	limit	100.000	0
nuclearve	–	4 078 232	–	limit	100.000	0
nuclearvf	–	4 063 210	–	limit	100.000	0
nuclear25	–	196 827	–	limit	100.000	0
nuclear25a	–	10 595	–	limit	100.000	0
nuclear25b	–	10 579	–	limit	100.000	0
nuclear49a	–	847	–	limit	100.000	0
nuclear49b	–	3 184	–	limit	100.000	0
nuclear14	–	1 356 147	–	limit	100.000	0
nuclear14a	–	11 851	–	limit	100.000	0
nuclear14b	–	47 680	–	limit	100.000	0
nuclear10a	–	43	–	limit	–	0
nuclear10b	–	159	–	limit	100.000	0
nvs01	11	16	0.0	0.0	0.000	2
nvs02	1	1	0.0	0.0	0.000	1
nvs03	1	1	0.0	0.0	0.000	1
nvs04	3	4	0.0	0.0	0.000	1
nvs06	8	31	0.0	0.0	0.000	4
nvs07	1	1	0.0	0.0	0.000	1
nvs08	6	9	0.0	0.0	0.000	3
nvs09	1 469	8 223 306	1.0	limit	90.651	19
nvs10	1	1	0.0	0.0	0.000	1
nvs11	4	9	0.0	0.0	0.000	2
nvs12	5	13	0.0	0.0	0.000	2
nvs13	6	19	0.0	0.0	0.000	2
nvs14	1	1	0.0	0.0	0.000	1
nvs15	3	8	0.0	0.0	0.000	2
nvs17	63	89	0.1	0.1	0.003	3
nvs18	40	52	0.1	0.1	0.002	3
nvs19	89	179	0.1	0.2	0.003	3
nvs20	67	563	0.3	1.0	0.025	23
nvs21	2	37	0.0	0.0	0.000	11
nvs23	67	187	0.2	0.3	0.006	3
nvs24	99	187	0.2	0.3	0.006	2
o7_2	8 648	1 114 243	6.5	472.2	0.228	6
o7_ar2_1	7 571	165 605	5.0	76.3	0.139	5
o7_ar25_1	1 634	3 461 386	1.7	690.0	0.070	11
o7_ar3_1	22 293	1 067 644	13.9	508.0	0.426	17
o7_ar4_1	24 721	1 743 528	15.9	861.4	0.485	10
o7_ar5_1	3 640	776 926	2.9	372.9	0.171	8
o7	62 487	2 927 954	39.7	1389.5	1.453	11
o8_ar4_1	8 906	6 243 285	7.9	limit	1.158	13
o9_ar4_1	1 232 869	5 543 561	1009.4	limit	28.186	4
oaer	2	3	0.0	0.0	0.000	2

Table B.27 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
oil2	–	345 724	–	limit	100.000	0
oil	–	53 477	–	limit	100.000	0
ortez	27	35	0.1	0.1	0.003	1
parallel	355	714 142	2.5	3197.2	3.471	18
pb302035	635	859 471	17.3	limit	30.730	1
pb302055	1 401	843 160	19.4	limit	35.036	1
pb302075	730	1 018 344	17.8	limit	36.603	1
pb302095	456	854 461	17.7	limit	17.278	5
pb351535	1 035	1 134 285	12.6	limit	29.991	1
pb351555	1 096	1 129 943	13.7	limit	25.055	1
pb351575	871	1 027 105	14.8	limit	26.156	1
pb351595	1 607	1 245 982	25.8	limit	35.689	1
prob02	1	1	0.0	0.0	0.000	1
prob03	1	1	0.0	0.0	0.000	1
procsel	2	2	0.0	0.0	0.000	1
product2	–	1 696 989	–	limit	100.000	0
product	6 492	25 928	19.5	44.0	0.542	5
pump	60	1 172	1.6	11.7	0.108	8
qapw	15	519 167	3.8	limit	4.263	1
qap	15	2 011 229	1.4	limit	7.343	12
ravem	47	163	0.4	0.5	0.011	4
ravempb	46	234	0.4	0.5	0.011	5
risk2b	371	467	0.2	0.3	0.006	5
risk2bpb	2	11	0.1	0.2	0.004	3
saa_2	–	216	–	limit	100.000	0
sep1	27	35	0.0	0.0	0.000	2
space25	–	3 773	–	limit	100.000	0
space25a	–	1 693 291	–	limit	100.000	0
space960	–	6 471	–	limit	100.000	0
spectra2	58	67	0.8	0.8	0.022	1
spring	43	135	0.0	0.1	0.001	5
st_e13	2	3	0.0	0.0	0.000	2
st_e14	5	8	0.0	0.0	0.000	2
st_e15	1	1	0.0	0.0	0.000	1
st_e27	1	1	0.0	0.0	0.000	1
st_e29	5	18	0.0	0.0	0.000	5
st_e31	501	1 113	0.2	0.7	0.007	12
st_e35	931	58 960	1.2	20.9	0.269	13
st_e36	634	708	0.4	0.4	0.011	3
st_e38	5	15	0.0	0.0	0.000	3
st_e40	17	22	0.1	0.1	0.001	1
st_miqp1	1	1	0.0	0.0	0.000	1
st_miqp2	2	5	0.0	0.0	0.000	3
st_miqp3	2	3	0.0	0.0	0.000	2
st_miqp4	1	1	0.0	0.0	0.000	1
st_miqp5	1	1	0.0	0.0	0.000	1
stockcycle	3	69 920	0.9	336.5	0.450	9
st_test1	1	1	0.0	0.0	0.000	1
st_test2	1	1	0.0	0.0	0.000	1
st_test3	1	1	0.0	0.0	0.000	1
st_test4	1	1	0.0	0.0	0.000	1
st_test5	1	1	0.0	0.0	0.000	1
st_test6	1	1	0.0	0.0	0.000	1

Table B.27 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
st_test8	1	1	0.0	0.0	0.000	1
st_testgr1	15	55	0.0	0.0	0.000	6
st_testgr3	3	9	0.0	0.0	0.000	1
st_testph4	1	1	0.0	0.0	0.000	1
super1	–	53 346	–	limit	100.000	0
super2	–	48 878	–	limit	100.000	0
super3	–	62 434	–	limit	100.000	0
super3t	–	47 449	–	limit	100.000	0
synheat	–	9 337 359	–	limit	100.000	0
synthes1	2	5	0.0	0.0	0.000	1
synthes2	3	6	0.0	0.0	0.000	2
synthes3	3	228 120	0.0	21.7	0.001	8
tln2	1	1	0.0	0.0	0.000	1
tln4	27	3 168	0.1	1.2	0.003	5
tln5	2 184	16 705	1.3	8.3	0.047	5
tln6	49 168	4 772 333	33.8	limit	0.964	3
tln7	58 955	2 580 740	67.2	limit	7.924	8
tln12	308 830	467 669	2411.8	limit	74.861	1
tloss	30	203	0.1	0.2	0.004	3
tls2	6	13	0.1	0.1	0.002	2
tls4	34	36 715	0.7	68.2	0.138	5
tls5	21 444	213 652	124.6	limit	14.787	15
tls6	181 878	899 371	854.3	limit	31.982	6
tls7	171 050	396 963	1494.9	limit	60.860	11
tls12	–	178 119	–	limit	100.000	0
tltr	12	23	0.2	0.2	0.006	4
uselinear	–	4 782	–	limit	100.000	0
util	10	379	0.0	0.1	0.000	3
waste	1 424 559	1 846 960	2733.7	limit	80.407	1
water4	1 429	1 699 276	3.1	limit	7.393	8
waterx	5 887	111 551	26.9	limit	67.053	4
waterz	6 616	2 767 901	8.8	limit	23.673	4

Table B.28.: Performance of SCIP 3.0.2, default mode, on the SAP test set

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
snp-001-01	1	2 594 433	0.3	limit	0.036	0	482	1027.6	18
snp-001-02	1	328 406	2.0	limit	0.968	0	35	1336.8	19
snp-001-03	1	750 167	2.0	limit	0.490	0	173	991.2	19
snp-001-04	1	112 089	5.0	limit	6.364	0	27	1555.6	19
snp-001-05	1	59 303	29.5	limit	36.108	0	2	1616.7	18
snp-001-06	1	16 942	26.6	limit	21.556	0	2	1834.7	18
snp-001-07	1	314	49.9	limit	58.254	0	2	3051.7	18
snp-001	1	17	153.5	limit	69.932	0	2	3192.0	17
snp-002-01	1	1	0.4	0.4	0.011	0	1	0.0	3
snp-002-02	1	1	0.7	0.7	0.019	0	1	0.0	3
snp-002-03	1	1	1.0	1.0	0.028	0	1	0.1	3
snp-002-04	1	1	1.6	1.6	0.044	0	1	0.1	3
snp-002-05	1	1	2.5	2.5	0.069	1	2	0.1	2
snp-002-06	1	1	3.2	3.2	0.089	1	2	0.1	2
snp-002	1	2 128 063	57.6	limit	35.032	0	1	644.3	18
snp-003-01	1	1	0.1	0.1	0.003	0	1	0.0	3
snp-003-02	1	1	0.1	0.1	0.003	0	1	0.0	3
snp-003-03	1	3	0.4	0.9	0.012	0	6	0.2	10
snp-003-04	1	154 632	1.4	limit	1.084	0	1938	2051.8	18
snp-003-05	1	6 906	2.0	limit	6.911	0	1412	3234.6	18
snp-003-06	1	6 277	2.4	limit	7.058	0	1141	3224.9	18
snp-003	1	1	7.8	limit	17.205	0	11	3560.3	9
snp-004-02	1	1	1.0	1.0	0.028	1	3	0.0	5
snp-004-04	–	410	–	limit	100.000	0	0	6.6	2
snp-004-05	–	304	–	limit	100.000	0	0	7.9	2
snp-004-06	–	107	–	limit	100.000	0	0	45.9	2
snp-004	–	36	–	limit	100.000	0	0	206.9	2
snp-005-01	1	1	0.7	0.7	0.018	1	1	0.0	2
snp-005-02	1	1	1.0	1.0	0.028	1	3	0.0	5
snp-005-03	1	1	2.7	2.7	0.075	0	3	0.1	5
snp-005-04	1	21 031	30.4	35.0	0.844	0	285	1.3	18
snp-005-05	1	410 757	80.9	172.8	2.247	0	197	20.4	18
snp-005-06	1	60	218.8	limit	8.116	0	11	728.2	15
snp-005	1	5	841.1	limit	59.639	0	1	221.1	6
snp-008-01	1	1	0.2	0.2	0.005	0	1	0.0	3
snp-008-02	1	12 708 743	8.3	limit	0.231	0	27	464.3	19
snp-008-03	1	61 958	9.3	limit	6.436	0	22	1843.1	19
snp-008-04	1	8 708	14.3	limit	34.809	0	11	2053.2	19
snp-008-05	1	222	275.4	limit	83.015	0	2	1772.5	18
snp-008	1	14	709.4	limit	32.642	0	2	2443.3	13

Table B.29.: Performance of SCIP 3.0.2, primal heuristics deactivated, on the SAP test set

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
snp-001-01	–	4 217 940	–	limit	100.000	0
snp-001-02	–	485 305	–	limit	100.000	0
snp-001-03	–	1 480 696	–	limit	100.000	0
snp-001-04	–	323 207	–	limit	100.000	0
snp-001-05	–	131 895	–	limit	100.000	0
snp-001-06	–	76 765	–	limit	100.000	0
snp-001-07	–	36 769	–	limit	100.000	0
snp-001	–	12 944	–	limit	100.000	0
snp-002-01	1	1	0.5	0.5	0.013	1
snp-002-02	1	1	0.8	0.8	0.022	1
snp-002-03	1	1	1.0	1.0	0.028	1
snp-002-04	1	1	1.7	1.7	0.046	1
snp-002-05	1	1	2.5	2.5	0.069	1
snp-002-06	1	1	3.0	3.0	0.083	1
snp-002	–	4 553 397	–	limit	100.000	0
snp-003-01	1	1	0.1	0.1	0.001	1
snp-003-02	1	1	0.1	0.1	0.003	1
snp-003-03	5	6	0.5	0.5	0.014	2
snp-003-04	–	190 263	–	limit	100.000	0
snp-003-05	–	172 938	–	limit	100.000	0
snp-003-06	–	156 786	–	limit	100.000	0
snp-003	–	75 923	–	limit	100.000	0
snp-004-02	238	63 754	1.2	38.5	0.204	160
snp-004-04	–	421	–	limit	100.000	0
snp-004-05	–	303	–	limit	100.000	0
snp-004-06	–	109	–	limit	100.000	0
snp-004	–	40	–	limit	100.000	0
snp-005-01	1	1	0.6	0.6	0.016	1
snp-005-02	1	1	0.9	0.9	0.025	1
snp-005-03	1	1	2.6	2.6	0.072	1
snp-005-04	141	29 643	29.8	34.2	0.828	4
snp-005-05	304	337 775	80.8	141.4	2.244	6
snp-005-06	–	373	–	limit	100.000	0
snp-005	–	5	–	limit	100.000	0
snp-008-01	1	1	0.2	0.2	0.004	1
snp-008-02	175 806	23 647 174	33.2	limit	1.022	1
snp-008-03	–	183 539	–	limit	100.000	0
snp-008-04	–	78 218	–	limit	100.000	0
snp-008-05	–	213	–	limit	100.000	0
snp-008	–	190	–	limit	100.000	0

Table B.30.: Performance of SCIP 3.0.2, default mode, on the Siemens test set

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
pmbrp-096-05-09-l	–	84 342	–	limit	100.000	0	0	869.1	18
pmbrp-096-05-09-m	8 400	476 192	105.0	limit	2.962	0	2	278.1	20
pmbrp-096-05-12-l	–	213 100	–	limit	100.000	0	0	277.7	17
pmbrp-096-10-09-l	393 340	435 064	3210.7	limit	89.220	0	1	163.3	20
pmbrp-096-10-09-m	1	1 661 231	6.1	limit	0.176	0	9	159.8	21
pmbrp-096-10-12-l	105 390	584 453	634.7	limit	17.654	0	8	245.4	21
pmbrp-096-15-09-l	1	702 428	4.4	limit	0.124	0	6	165.8	21
pmbrp-096-15-09-m	1	2 015 375	4.1	limit	0.114	0	30	227.1	21
pmbrp-096-15-12-l	8 220	1 318 628	38.8	limit	1.078	0	24	212.8	21
pmbrp-108-05-09-l	–	111 378	–	limit	100.000	0	0	653.8	18
pmbrp-108-05-09-m	–	573 858	–	limit	100.000	0	0	115.1	18
pmbrp-108-05-12-l	–	196 811	–	limit	100.000	0	0	429.9	17
pmbrp-108-10-09-l	1	345 707	15.3	limit	0.662	0	2	454.8	20
pmbrp-108-10-09-m	1	1 734 937	6.3	limit	0.175	0	22	287.9	21
pmbrp-108-10-12-l	–	692 650	–	limit	100.000	0	0	85.4	18
pmbrp-108-15-09-l	56 540	770 193	302.1	limit	8.402	0	29	436.1	21
pmbrp-108-15-09-m	1	2 047 723	0.9	limit	0.025	0	14	154.7	21
pmbrp-108-15-12-l	2 740	1 165 656	23.2	limit	0.646	0	18	151.3	21
pmbrp-120-05-09-l	–	107 506	–	limit	100.000	0	0	765.6	18
pmbrp-120-05-09-m	81 360	417 851	664.7	limit	18.494	0	3	472.6	20
pmbrp-120-05-12-l	46 040	210 661	983.1	limit	27.329	0	2	309.6	20
pmbrp-120-10-09-l	–	373 331	–	limit	100.000	0	0	139.5	18
pmbrp-120-10-09-m	1	1 493 531	6.2	limit	0.172	0	38	509.9	21
pmbrp-120-10-12-l	1	748 211	12.1	limit	0.346	0	6	151.9	21
pmbrp-120-15-09-l	182 160	776 755	832.8	limit	23.145	0	21	446.1	21
pmbrp-120-15-09-m	1	1 993 395	4.6	limit	0.128	0	20	130.6	21
pmbrp-120-15-12-l	124 540	1 320 532	387.8	limit	10.779	0	15	193.5	21

Table B.31.: Performance of SCIP 3.0.2, primal heuristics deactivated, on the Siemens test set

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
pmbrp-096-05-09-l	–	115 894	–	limit	100.000	0
pmbrp-096-05-09-m	–	552 846	–	limit	100.000	0
pmbrp-096-05-12-l	–	240 510	–	limit	100.000	0
pmbrp-096-10-09-l	–	460 843	–	limit	100.000	0
pmbrp-096-10-09-m	–	2 032 634	–	limit	100.000	0
pmbrp-096-10-12-l	–	603 419	–	limit	100.000	0
pmbrp-096-15-09-l	–	627 861	–	limit	100.000	0
pmbrp-096-15-09-m	–	2 419 960	–	limit	100.000	0
pmbrp-096-15-12-l	–	1 756 902	–	limit	100.000	0
pmbrp-108-05-09-l	–	131 710	–	limit	100.000	0
pmbrp-108-05-09-m	–	595 033	–	limit	100.000	0
pmbrp-108-05-12-l	–	241 393	–	limit	100.000	0
pmbrp-108-10-09-l	–	373 814	–	limit	100.000	0
pmbrp-108-10-09-m	–	1 932 528	–	limit	100.000	0
pmbrp-108-10-12-l	–	664 072	–	limit	100.000	0
pmbrp-108-15-09-l	–	836 530	–	limit	100.000	0
pmbrp-108-15-09-m	–	2 377 755	–	limit	100.000	0
pmbrp-108-15-12-l	–	1 305 856	–	limit	100.000	0
pmbrp-120-05-09-l	–	136 995	–	limit	100.000	0
pmbrp-120-05-09-m	–	538 485	–	limit	100.000	0
pmbrp-120-05-12-l	–	223 343	–	limit	100.000	0
pmbrp-120-10-09-l	–	383 073	–	limit	100.000	0
pmbrp-120-10-09-m	–	1 756 236	–	limit	100.000	0
pmbrp-120-10-12-l	–	876 834	–	limit	100.000	0
pmbrp-120-15-09-l	–	749 177	–	limit	100.000	0
pmbrp-120-15-09-m	–	2 308 997	–	limit	100.000	0
pmbrp-120-15-12-l	–	1 325 187	–	limit	100.000	0

Table B.32.: Performance of SCIP 3.0.2, default mode, on the FORNE test set

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
hn-abnahme9-00000	217	217	5.2	5.2	0.143	0	1	2.3	12
hn-abnahme9-00001	70	70	3.4	3.4	0.093	0	1	1.3	11
hn-abnahme9-00002	1	1	2.1	2.1	0.058	0	1	0.2	10
hn-abnahme9-00003	189	189	3.7	3.7	0.102	0	1	1.6	11
hn-abnahme9-00004	1 360	1 360	12.8	12.9	0.356	0	1	2.7	14
hn-abnahme9-00005	208	208	5.5	5.5	0.153	0	1	1.7	12
hn-abnahme9-00006	6	6	2.4	2.4	0.067	0	1	0.9	12
hn-abnahme9-00007	17	17	2.2	2.2	0.060	0	1	0.4	10
hn-abnahme9-00008	1	1	1.1	1.1	0.030	0	1	0.2	11
hn-abnahme9-00009	680	680	6.6	6.6	0.183	0	1	2.8	12
hn-abnahme9-00010	6 000	6 000	31.6	31.6	0.878	0	1	6.0	17
hn-abnahme9-00011	7 032	7 032	30.9	30.9	0.858	1	0	2.3	14
hn-abnahme9-00012	15	15	3.5	3.5	0.097	0	1	0.5	12
hn-abnahme9-00013	44	44	2.3	2.3	0.063	0	1	0.5	11
hn-abnahme9-00014	12	12	2.0	2.0	0.054	0	1	0.4	12
hn-abnahme9-00015	–	122	–	4.4	–	0	0	0.6	14
hn-abnahme9-00016	42	42	3.9	3.9	0.108	0	1	1.1	12
hn-abnahme9-00017	66	66	3.9	3.9	0.108	0	1	1.5	17
hn-abnahme9-00018	2 020	2 020	10.4	10.4	0.289	0	1	1.9	14
hn-abnahme9-00019	37	37	2.9	2.9	0.080	0	1	0.5	13
hn-abnahme9-00021	33	33	2.8	2.8	0.077	0	1	0.6	11
hn-abnahme9-00022	2 020	2 020	15.5	15.5	0.431	0	1	1.9	12
hn-abnahme9-00023	18	18	2.5	2.5	0.069	0	1	0.5	10
hn-abnahme9-00024	1	1	0.7	0.7	0.019	0	1	0.3	8
hn-abnahme9-00025	22	22	2.1	2.1	0.058	0	1	0.4	11
hn-abnahme9-00026	1	1	2.4	2.4	0.065	0	1	0.5	9
hn-abnahme9-00027	1	1	2.4	2.4	0.067	0	1	0.5	10
hn-abnahme9-00028	2 160	2 160	14.1	14.1	0.392	0	1	1.7	15
hn-abnahme9-00029	2 020	2 020	17.4	17.4	0.483	0	1	1.6	13
hn-abnahme9-00030	19	19	2.5	2.5	0.069	0	1	0.6	13
hn-abnahme9-00031	43	43	2.5	2.5	0.069	0	1	0.8	12
hn-abnahme9-00032	10	10	2.0	2.0	0.056	0	1	0.6	12
hn-abnahme9-00033	2 020	2 020	12.9	12.9	0.358	0	1	1.7	13
hn-abnahme9-00034	634	634	8.0	8.0	0.222	0	1	1.4	15
hn-abnahme9-00035	34	34	2.7	2.7	0.075	0	1	0.5	11
hn-abnahme9-00036	8	8	1.8	1.8	0.049	0	1	0.3	11
hn-abnahme9-00037	21 595	21 595	88.0	88.1	2.447	1	0	5.2	19
hn-abnahme9-00038	82	82	4.9	4.9	0.136	0	1	1.3	11
hn-abnahme9-00039	359	359	8.1	8.1	0.225	1	0	4.0	15
hn-abnahme9-00040	32	32	2.2	2.3	0.061	0	1	0.6	12
hn-abnahme9-00041	32	32	2.2	2.2	0.061	0	1	0.6	12
hn-abnahme9-00042	18	18	2.2	2.2	0.061	0	1	0.9	11
hn-frei-corr-00000	366	366	3.1	3.1	0.086	0	1	0.4	13
hn-frei-corr-00001	14	14	1.8	1.8	0.049	0	1	0.2	9
hn-frei-corr-00002	–	0	–	0.1	–	0	0	0.0	1
hn-frei-corr-00003	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr-00004	20	20	1.8	1.8	0.050	0	1	0.2	10
hn-frei-corr-00005	1	1	0.3	0.3	0.008	0	1	0.2	7
hn-frei-corr-00006	–	33	–	2.0	–	0	0	0.2	12

Table B.32 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
hn-frei-corr-00007	103	103	1.8	1.8	0.049	0	1	0.2	17
hn-frei-corr-00008	–	309	–	3.0	–	0	0	0.1	12
hn-frei-corr-00009	1 360	1 360	7.0	7.0	0.194	0	1	1.8	11
hn-frei-corr-00010	12	12	1.6	1.6	0.044	0	1	0.2	11
hn-frei-corr-00011	73 330	73 330	395.6	395.6	10.988	0	1	7.1	18
hn-frei-corr-00012	23 365	23 365	97.6	97.6	2.711	1	0	5.3	19
hn-frei-corr-00013	–	544	–	4.6	–	0	0	0.4	15
hn-frei-corr-00014	41	41	1.6	1.6	0.044	0	1	0.4	12
hn-frei-corr-00015	–	28	–	1.6	–	0	0	0.2	16
hn-frei-corr-00016	186	186	2.9	2.9	0.080	0	1	0.4	11
hn-frei-corr-00017	–	0	–	0.1	–	0	0	0.0	1
hn-frei-corr-00018	19	19	1.6	1.6	0.044	0	1	0.2	10
hn-frei-corr-00019	–	193	–	2.7	–	0	0	0.2	10
hn-frei-corr-00020	–	517	–	4.0	–	0	0	0.4	17
hn-frei-corr-00021	–	153	–	3.2	–	0	0	0.2	9
hn-frei-corr-00022	–	25	–	1.5	–	0	0	0.2	8
hn-frei-corr-00023	140	140	2.7	2.7	0.075	0	1	0.8	12
hn-frei-corr-00024	–	68	–	2.4	–	0	0	0.1	10
hn-frei-corr-00025	20	20	1.5	1.5	0.041	0	1	0.2	11
hn-frei-corr-00026	–	491	–	4.6	–	0	0	0.7	17
hn-frei-corr-00027	279	279	3.6	3.6	0.100	0	1	0.5	10
hn-frei-corr-00028	21	21	1.7	1.7	0.046	0	1	0.3	9
hn-frei-corr-00029	–	0	–	0.1	–	0	0	0.0	1
hn-frei-corr-00030	–	115	–	1.9	–	0	0	0.1	10
hn-frei-corr-00031	36	36	2.0	2.0	0.056	0	1	0.2	10
hn-frei-corr-00032	–	74	–	2.1	–	0	0	0.2	10
hn-frei-corr-00033	–	90	–	1.8	–	0	0	0.1	10
hn-frei-corr-00034	5	5	1.3	1.3	0.036	0	1	0.2	11
hn-frei-corr-00035	–	245	–	2.7	–	0	0	0.2	13
hn-frei-corr-00036	200	200	3.0	3.0	0.082	0	1	0.5	18
hn-frei-corr-00037	23	23	1.6	1.6	0.044	0	1	0.2	10
hn-frei-corr-00038	1	1	0.3	0.3	0.008	0	1	0.2	7
hn-frei-corr-00039	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr-00040	–	56	–	1.6	–	0	0	0.2	9
hn-frei-corr-00041	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr-00042	30	30	1.9	1.9	0.053	0	1	0.2	12
hn-frei-corr-00043	–	184	–	3.0	–	0	0	0.3	10
hn-frei-corr-00044	4 510	4 510	19.3	19.3	0.536	0	1	1.3	16
hn-frei-corr-00045	14	14	1.4	1.4	0.039	0	1	0.2	11
hn-frei-corr-00046	–	526	–	4.7	–	0	0	0.6	15
hn-frei-corr-00047	1	1	0.4	0.4	0.011	0	1	0.3	7
hn-frei-corr-00048	–	559	–	4.6	–	0	0	0.7	16
hn-frei-corr-00049	295	295	2.9	2.9	0.081	0	1	0.3	14
hn-frei-corr-00050	–	17	–	1.4	–	0	0	0.2	12
hn-frei-corr-00051	–	508	–	4.8	–	0	0	0.7	18
hn-frei-corr-00052	–	870	–	5.7	–	0	0	1.1	16
hn-frei-corr-00053	–	35	–	2.2	–	0	0	0.2	10
hn-frei-corr-00054	19	19	1.6	1.6	0.043	0	1	0.2	11
hn-frei-corr-00055	65	65	1.8	1.8	0.049	0	1	0.3	10
hn-frei-corr-00056	–	469 804	–	1621.9	–	0	0	16.6	18
hn-frei-corr-00057	–	645	–	4.1	–	0	0	0.2	14
hn-frei-corr-00058	–	0	–	0.1	–	0	0	0.0	1
hn-frei-corr-00059	–	329	–	2.8	–	0	0	0.3	10

Table B.32 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
hn-frei-corr-00060	–	635 058	–	2631.0	–	0	0	19.3	18
hn-frei-corr-00061	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr-00062	–	57	–	1.6	–	0	0	0.1	10
hn-frei-corr-00063	–	0	–	0.1	–	0	0	0.0	1
hn-frei-corr-00064	–	27	–	1.5	–	0	0	0.1	10
hn-frei-corr-00065	–	0	–	0.1	–	0	0	0.0	1
hn-frei-corr-00066	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr-00067	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr-00068	–	0	–	0.1	–	0	0	0.0	1
hn-frei-corr-00069	–	8	–	1.6	–	0	0	0.1	9
hn-frei-corr-00070	–	41	–	1.7	–	0	0	0.1	8
hn-frei-corr-00071	–	6 673	–	33.9	–	0	0	2.4	18
hn-frei-corr-00072	24	24	0.9	0.9	0.025	0	1	0.2	10
hn-frei-corr-00073	–	817	–	4.5	–	0	0	0.4	16
hn-frei-corr-00074	28	28	1.6	1.6	0.044	0	1	0.2	9
hn-frei-corr-00075	66	66	1.6	1.6	0.044	0	1	0.2	13
hn-frei-corr-00076	–	97	–	2.3	–	0	0	0.2	11
hn-frei-corr-00077	32	32	1.1	1.1	0.030	0	1	0.1	10
hn-frei-corr-00078	–	264 210	–	978.4	–	0	0	8.8	18
hn-frei-corr-00079	–	153	–	2.1	–	0	0	0.2	11
hn-frei-corr-00080	–	218	–	3.0	–	0	0	0.3	10
hn-frei-corr-00081	67	67	2.2	2.3	0.061	0	1	0.2	10
hn-frei-corr-00082	–	49	–	1.3	–	0	0	0.2	8
hn-frei-corr-00083	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr-00084	151	151	2.8	2.8	0.078	0	1	0.2	17
hn-frei-corr-00085	172	172	2.6	2.6	0.072	0	1	0.3	12
hn-frei-corr-00086	–	781	–	5.3	–	0	0	0.7	19
hn-frei-corr-00087	–	191	–	2.5	–	0	0	0.3	16
hn-frei-corr-00088	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr-00089	16	16	1.9	1.9	0.053	0	1	0.2	12
hn-frei-corr-00090	–	86	–	2.2	–	0	0	0.2	8
hn-frei-corr-00091	–	165	–	2.5	–	0	0	0.2	11
hn-frei-corr-00092	–	979	–	6.4	–	0	0	0.3	11
hn-frei-corr-00093	–	71	–	2.0	–	0	0	0.2	10
hn-frei-corr-00094	82	82	2.2	2.2	0.061	0	1	0.3	15
hn-frei-corr-00095	49	49	1.7	1.7	0.047	0	1	0.3	10
hn-frei-corr-00096	–	133	–	2.6	–	0	0	0.2	17
hn-frei-corr-00097	22	22	1.2	1.2	0.033	0	1	0.1	12
hn-frei-corr-00098	1	1	0.9	0.9	0.024	0	1	0.2	11
hn-frei-corr-00099	22	22	1.7	1.7	0.047	0	1	0.2	12
hn-frei-corr95-00000	2 020	2 020	6.9	6.9	0.191	0	1	1.0	10
hn-frei-corr95-00001	21	21	2.1	2.1	0.058	0	1	0.3	10
hn-frei-corr95-00002	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr95-00003	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr95-00004	1	1	0.4	0.4	0.011	0	1	0.3	7
hn-frei-corr95-00005	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00006	44	44	1.4	1.4	0.038	0	1	0.2	10
hn-frei-corr95-00007	1	1	0.5	0.5	0.013	0	1	0.2	7
hn-frei-corr95-00008	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00009	11	11	1.3	1.3	0.035	0	1	0.2	11
hn-frei-corr95-00010	60	60	2.0	2.0	0.055	0	1	0.4	10
hn-frei-corr95-00011	22	22	1.9	1.9	0.051	0	1	0.2	10
hn-frei-corr95-00012	25	25	2.1	2.1	0.058	0	1	0.2	9

Table B.32 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
hn-frei-corr95-00013	32	32	2.2	2.2	0.060	0	1	0.3	9
hn-frei-corr95-00014	83	83	2.1	2.1	0.058	0	1	0.4	10
hn-frei-corr95-00015	57	57	2.1	2.1	0.057	0	1	0.2	11
hn-frei-corr95-00016	6	6	1.7	1.7	0.047	0	1	0.3	12
hn-frei-corr95-00017	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr95-00018	16	16	1.3	1.3	0.036	0	1	0.2	14
hn-frei-corr95-00019	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00020	1	1	0.6	0.7	0.018	0	1	0.5	7
hn-frei-corr95-00021	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00022	65	65	2.0	2.0	0.056	0	1	0.5	17
hn-frei-corr95-00023	1	1	0.2	0.2	0.006	0	1	0.1	7
hn-frei-corr95-00024	1	1	0.2	0.2	0.006	0	1	0.1	7
hn-frei-corr95-00025	1	1	0.4	0.4	0.010	0	1	0.2	7
hn-frei-corr95-00026	79	79	2.5	2.5	0.068	0	1	0.5	9
hn-frei-corr95-00027	24	24	1.6	1.6	0.044	0	1	0.3	9
hn-frei-corr95-00028	68	68	2.0	2.0	0.056	0	1	0.2	10
hn-frei-corr95-00029	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr95-00030	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00031	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00032	1	1	0.5	0.5	0.013	0	1	0.3	7
hn-frei-corr95-00033	22	22	1.4	1.4	0.039	0	1	0.3	10
hn-frei-corr95-00034	7	7	1.5	1.5	0.041	0	1	0.3	11
hn-frei-corr95-00035	1	1	0.5	0.5	0.014	0	1	0.2	7
hn-frei-corr95-00036	18	18	1.7	1.7	0.047	0	1	0.3	12
hn-frei-corr95-00037	1	1	0.6	0.6	0.016	0	1	0.4	7
hn-frei-corr95-00038	1	1	0.3	0.3	0.008	0	1	0.2	7
hn-frei-corr95-00039	–	3	–	1.4	–	0	0	0.1	8
hn-frei-corr95-00040	–	37	–	1.3	–	0	0	0.2	8
hn-frei-corr95-00041	–	49	–	1.0	–	0	0	0.2	10
hn-frei-corr95-00042	21	21	1.3	1.4	0.036	0	1	0.1	10
hn-frei-corr95-00043	28	28	1.8	1.8	0.049	0	1	0.2	10
hn-frei-corr95-00044	45	45	2.2	2.2	0.061	0	1	0.3	11
hn-frei-corr95-00045	22	22	1.9	1.9	0.052	0	1	0.2	11
hn-frei-corr95-00046	14	14	1.9	1.9	0.053	0	1	0.4	11
hn-frei-corr95-00047	54	54	2.6	2.6	0.072	0	1	0.5	12
hn-frei-corr95-00048	24	24	2.2	2.2	0.061	0	1	0.4	11
hn-frei-corr95-00049	22	22	1.3	1.3	0.036	0	1	0.2	10
hn-frei-corr95-00050	1	1	0.2	0.2	0.006	0	1	0.1	7
hn-frei-corr95-00051	3	3	1.5	1.5	0.041	0	1	0.3	11
hn-frei-corr95-00052	19	19	1.7	1.7	0.047	0	1	0.4	9
hn-frei-corr95-00053	1	1	0.3	0.3	0.008	0	1	0.2	7
hn-frei-corr95-00054	22	22	2.0	2.0	0.055	0	1	0.3	10
hn-frei-corr95-00055	1	1	0.3	0.3	0.008	0	1	0.2	7
hn-frei-corr95-00056	9	9	1.6	1.6	0.044	0	1	0.2	11
hn-frei-corr95-00057	21	21	1.5	1.5	0.042	0	1	0.2	10
hn-frei-corr95-00058	–	0	–	0.1	–	0	0	0.0	1
hn-frei-corr95-00059	11	11	1.2	1.2	0.033	0	1	0.2	11
hn-frei-corr95-00060	12	12	1.3	1.4	0.036	0	1	0.2	9
hn-frei-corr95-00061	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr95-00062	24	24	2.2	2.2	0.061	0	1	0.3	11
hn-frei-corr95-00063	–	0	–	0.1	–	0	0	0.0	1
hn-frei-corr95-00064	39	39	1.8	1.8	0.050	0	1	0.2	10
hn-frei-corr95-00065	–	0	–	0.2	–	0	0	0.0	1

Table B.32 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
hn-frei-corr95-00066	–	0	–	0.2	–	0	0	0.0	1
hn-frei-corr95-00067	–	1	–	0.8	–	0	0	0.2	10
hn-frei-corr95-00068	–	0	–	0.1	–	0	0	0.0	1
hn-frei-corr95-00069	–	39	–	1.5	–	0	0	0.1	8
hn-frei-corr95-00070	–	5	–	1.9	–	0	0	0.3	8
hn-frei-corr95-00071	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00072	43	43	1.7	1.7	0.047	0	1	0.2	9
hn-frei-corr95-00073	22	22	1.8	1.8	0.049	0	1	0.4	10
hn-frei-corr95-00074	23	23	1.7	1.7	0.046	0	1	0.2	10
hn-frei-corr95-00075	1	1	0.4	0.4	0.010	0	1	0.2	7
hn-frei-corr95-00076	36	36	2.0	2.0	0.056	0	1	0.2	12
hn-frei-corr95-00077	1	1	0.3	0.3	0.008	0	1	0.2	7
hn-frei-corr95-00078	21	21	1.7	1.7	0.047	0	1	0.3	10
hn-frei-corr95-00079	22	22	1.6	1.6	0.044	0	1	0.2	10
hn-frei-corr95-00080	1	1	0.4	0.4	0.010	0	1	0.2	7
hn-frei-corr95-00081	123	123	2.7	2.7	0.074	0	1	0.4	11
hn-frei-corr95-00082	–	15	–	1.8	–	0	0	0.2	8
hn-frei-corr95-00083	11 100	11 100	27.2	27.2	0.756	0	1	2.6	17
hn-frei-corr95-00084	1	1	0.3	0.3	0.008	0	1	0.1	7
hn-frei-corr95-00085	44	44	2.0	2.0	0.056	0	1	0.2	12
hn-frei-corr95-00086	13	13	1.5	1.5	0.041	0	1	0.3	11
hn-frei-corr95-00087	1	1	1.2	1.2	0.033	0	1	0.5	10
hn-frei-corr95-00088	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00089	5	5	1.0	1.0	0.027	0	1	0.1	11
hn-frei-corr95-00090	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00091	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00092	1	1	0.5	0.6	0.015	0	1	0.4	7
hn-frei-corr95-00093	1	1	0.4	0.4	0.011	0	1	0.2	7
hn-frei-corr95-00094	86	86	2.3	2.3	0.063	0	1	0.4	11
hn-frei-corr95-00095	5	5	1.1	1.1	0.030	0	1	0.2	11
hn-frei-corr95-00096	21	21	1.7	1.7	0.046	0	1	0.3	10
hn-frei-corr95-00097	21	21	1.5	1.5	0.042	0	1	0.2	9
hn-frei-corr95-00098	1	1	1.2	1.2	0.033	0	1	0.5	9
hn-frei-corr95-00099	1	1	0.3	0.3	0.008	0	1	0.1	7
hn-sn4-random-00000	19	19	3.9	3.9	0.108	0	1	0.8	10
hn-sn4-random-00001	1	1	1.8	1.8	0.050	0	1	0.7	9
hn-sn4-random-00002	21	21	3.6	3.6	0.099	0	1	0.6	9
hn-sn4-random-00003	27	27	3.0	3.1	0.085	0	1	0.5	11
hn-sn4-random-00004	–	111	–	5.8	–	0	0	0.5	8
hn-sn4-random-00005	–	0	–	0.3	–	0	0	0.0	1
hn-sn4-random-00006	24	24	3.6	3.6	0.100	0	1	0.9	9
hn-sn4-random-00007	22	22	2.9	2.9	0.081	0	1	0.3	10
hn-sn4-random-00008	6	6	2.2	2.2	0.061	0	1	0.6	11
hn-sn4-random-00009	22	22	3.2	3.2	0.088	0	1	0.8	10
hn-sn4-random-00010	22	22	2.9	2.9	0.080	0	1	0.5	9
hn-sn4-random-00011	21	21	2.5	2.5	0.069	0	1	0.7	9
hn-sn4-random-00012	18	18	2.5	2.5	0.069	0	1	0.5	11
hn-sn4-random-00013	26	26	3.0	3.0	0.083	0	1	0.8	12
hn-sn4-random-00014	34	34	3.7	3.7	0.103	0	1	0.7	11
hn-sn4-random-00015	20	20	2.6	2.6	0.072	0	1	0.5	9
hn-sn4-random-00016	53	53	3.3	3.3	0.092	0	1	0.6	11
hn-sn4-random-00017	5	5	2.1	2.1	0.058	0	1	0.4	11
hn-sn4-random-00018	–	0	–	0.3	–	0	0	0.0	1

Table B.32 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
hn-sn4-random-00019	35	35	2.9	2.9	0.081	0	1	0.8	10
hn-sn4-random-00020	5 020	5 020	23.3	23.3	0.647	0	1	2.6	14
hn-sn4-random-00021	27	27	2.6	2.6	0.072	0	1	0.5	10
hn-sn4-random-00022	344	344	4.5	4.5	0.125	0	1	0.7	12
hn-sn4-random-00023	20	20	2.7	2.7	0.075	0	1	0.6	9
hn-sn4-random-00024	76	76	5.4	5.4	0.150	0	1	1.7	16
hn-sn4-random-00025	–	29	–	1.7	–	0	0	0.4	10
hn-sn4-random-00026	21	21	2.3	2.3	0.064	0	1	0.6	11
hn-sn4-random-00027	18	18	2.0	2.0	0.056	0	1	0.6	9
hn-sn4-random-00028	–	0	–	0.2	–	0	0	0.0	1
hn-sn4-random-00029	–	0	–	0.2	–	0	0	0.0	1
hn-sn4-random-00030	23	23	2.2	2.2	0.060	0	1	0.3	10
hn-sn4-random-00031	–	0	–	0.2	–	0	0	0.0	1
hn-sn4-random-00032	1 400	1 400	8.1	8.1	0.224	0	1	1.0	16
hn-sn4-random-00033	5	5	2.2	2.2	0.061	0	1	0.7	11
hn-sn4-random-00034	158	158	4.5	4.5	0.124	0	1	0.4	9
hn-sn4-random-00035	7	7	2.9	2.9	0.081	0	1	0.7	11
hn-sn4-random-00036	–	0	–	0.1	–	0	0	0.0	1
hn-sn4-random-00037	1	1	1.9	1.9	0.053	0	1	0.8	9
hn-sn4-random-00038	19	19	2.8	2.8	0.078	0	1	0.6	10
hn-sn4-random-00039	81	81	3.3	3.3	0.092	0	1	0.6	13
hn-sn4-random-00040	20	20	2.3	2.3	0.064	0	1	0.9	9
hn-sn4-random-00041	–	0	–	0.3	–	0	0	0.0	1
hn-sn4-random-00042	7	7	3.5	3.5	0.097	0	1	1.1	14
hn-sn4-random-00043	–	0	–	0.2	–	0	0	0.0	1
hn-sn4-random-00044	1	1	0.6	0.6	0.017	0	1	0.2	7
hn-sn4-random-00045	5	5	2.5	2.5	0.068	0	1	0.5	11
hn-sn4-random-00046	31	31	3.8	3.8	0.104	0	1	0.7	10
hn-sn4-random-00047	1 270	1 270	8.7	8.7	0.241	0	1	1.4	14
hn-sn4-random-00048	1	1	1.8	1.8	0.050	0	1	0.8	9
hn-sn4-random-00049	1	1	1.4	1.4	0.039	0	1	0.5	9
hn-sn4-random-00050	122	122	4.2	4.2	0.117	0	1	1.9	11
hn-sn4-random-00051	83	83	2.9	2.9	0.081	0	1	0.4	12
hn-sn4-random-00052	1	1	2.2	2.2	0.061	0	1	0.8	9
hn-sn4-random-00053	15	15	2.6	2.6	0.072	0	1	0.5	9
hn-sn4-random-00054	1	1	2.3	2.3	0.064	0	1	0.9	9
hn-sn4-random-00055	21	21	2.3	2.3	0.064	0	1	0.4	9
hn-sn4-random-00056	13	13	2.6	2.6	0.072	0	1	0.7	11
l-abnahme9-00000	–	30 643	–	limit	–	0	0	11.2	12
l-abnahme9-00001	–	27 022	–	limit	100.000	0	0	34.6	19
l-abnahme9-00002	–	34 250	–	limit	–	0	0	36.3	18
l-abnahme9-00003	–	41 407	–	limit	100.000	0	0	19.6	19
l-abnahme9-00004	–	33 165	–	limit	100.000	0	0	23.5	19
l-abnahme9-00005	–	35 402	–	limit	100.000	0	0	49.6	19
l-abnahme9-00006	–	53 701	–	limit	–	0	0	32.4	19
l-abnahme9-00007	–	125 653	–	limit	–	0	0	14.2	17
l-abnahme9-00008	–	47 740	–	limit	–	0	0	21.7	18
l-abnahme9-00009	–	26 685	–	limit	–	0	0	14.9	18
l-abnahme9-00010	–	47 470	–	limit	100.000	0	0	17.5	18
l-abnahme9-00011	–	41 915	–	limit	–	0	0	18.4	19
l-abnahme9-00012	–	53 413	–	limit	–	0	0	15.4	18
l-abnahme9-00013	–	47 623	–	limit	100.000	0	0	23.3	19
l-abnahme9-00014	–	91 750	–	limit	100.000	0	0	30.3	19

Table B.32 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
l-abnahme9-00015	–	60 620	–	limit	100.000	0	0	22.7	19
l-abnahme9-00016	–	63 595	–	limit	–	0	0	23.0	18
l-abnahme9-00017	–	41 106	–	limit	–	0	0	18.8	18
l-abnahme9-00018	–	42 960	–	limit	100.000	0	0	17.2	18
l-abnahme9-00019	–	48 420	–	limit	–	0	0	21.0	19
l-abnahme9-00020	–	46 158	–	limit	100.000	0	0	19.7	18
l-abnahme9-00021	–	19 817	–	limit	100.000	0	0	21.9	19
l-abnahme9-00022	–	69 774	–	limit	100.000	0	0	29.2	15
l-abnahme9-00023	–	32 454	–	limit	100.000	0	0	14.1	19
l-abnahme9-00024	–	49 552	–	limit	–	0	0	16.4	19
l-abnahme9-00025	–	48 451	–	limit	100.000	0	0	38.8	19
l-abnahme9-00026	–	45 432	–	limit	–	0	0	22.0	18
l-abnahme9-00027	–	73 961	–	limit	100.000	0	0	17.3	19
l-abnahme9-00028	–	54 940	–	limit	–	0	0	21.5	19
l-abnahme9-00029	–	37 100	–	limit	–	0	0	10.8	12
l-abnahme9-00030	–	29 635	–	limit	100.000	0	0	69.1	18
l-sn-random-00000	–	83 713	–	limit	100.000	0	0	24.5	18
l-sn-random-00001	–	119 654	–	limit	100.000	0	0	30.1	19
l-sn-random-00002	–	97 569	–	limit	100.000	0	0	28.3	18
l-sn-random-00003	–	42 296	–	limit	100.000	0	0	26.0	19
l-sn-random-00004	–	80 895	–	limit	100.000	0	0	19.8	18
l-sn-random-00005	–	64 580	–	limit	–	0	0	24.2	18
l-sn-random-00006	–	90 407	–	limit	100.000	0	0	18.5	19
l-sn-random-00007	–	120 056	–	limit	100.000	0	0	17.4	19
l-sn-random-00008	–	91 973	–	limit	100.000	0	0	25.1	18
l-sn-random-00009	–	56 486	–	limit	–	0	0	17.7	18
l-sn-random-00010	–	56 263	–	limit	100.000	0	0	30.9	19
l-sn-random-00011	–	87 441	–	limit	100.000	0	0	18.4	18
l-sn-random-00012	5	5	13.4	13.4	0.372	0	1	2.0	12
l-sn-random-00013	19 272	19 272	681.7	681.7	18.937	0	1	21.1	19
l-sn-random-00014	–	95 303	–	limit	100.000	0	0	28.8	18
l-sn-random-00015	–	62 974	–	limit	100.000	0	0	36.0	19
l-sn-random-00016	–	70 421	–	limit	100.000	0	0	39.6	18
l-sn-random-00017	5	5	12.3	12.3	0.342	0	1	2.1	12
l-sn-random-00018	–	68 416	–	limit	100.000	0	0	21.3	18
l-sn-random-00019	–	191 633	–	limit	100.000	0	0	9.2	19
l-sn-random-00020	–	78 483	–	limit	100.000	0	0	21.5	19
l-sn-random-00021	–	104 705	–	limit	100.000	0	0	25.7	19
l-sn-random-00022	–	124 240	–	limit	100.000	0	0	22.3	18
l-sn-random-00023	–	102 977	–	limit	100.000	0	0	19.3	18
l-sn-random-00024	–	182 028	–	limit	100.000	0	0	19.7	18
l-sn-random-00025	–	124 034	–	limit	100.000	0	0	31.3	18
l-sn-random-00026	–	67 434	–	limit	100.000	0	0	16.7	18
l-sn-random-00027	–	99 318	–	limit	100.000	0	0	27.4	19
l-sn-random-00028	–	96 162	–	limit	100.000	0	0	25.3	19
l-sn-random-00029	–	66 991	–	limit	100.000	0	0	29.9	19
l-sn-random-00030	–	93 647	–	limit	100.000	0	0	21.1	19
l-sn-random-00031	–	71 217	–	limit	100.000	0	0	21.9	18
l-sn-random-00032	–	57 607	–	limit	100.000	0	0	32.2	16
l-sn-random-00033	–	69 203	–	limit	100.000	0	0	25.6	18
l-sn-random-00034	–	59 596	–	limit	100.000	0	0	27.7	19
l-sn-random-00035	–	135 491	–	limit	100.000	0	0	22.9	19
l-sn-random-00036	–	106 632	–	limit	100.000	0	0	34.2	18

Table B.32 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
l-sn-random-00037	240	240	35.2	35.2	0.978	0	1	3.1	12
l-sn-random-00038	–	0	–	1.7	–	0	0	0.0	1
l-sn-random-00039	–	0	–	1.6	–	0	0	0.0	1
l-sn-random-00040	–	0	–	2.2	–	0	0	0.0	1
l-sn-random-00041	–	129 115	–	limit	100.000	0	0	34.8	19
l-sn-random-00042	–	0	–	2.1	–	0	0	0.0	1
l-sn-random-00043	–	107 231	–	limit	–	0	0	21.2	16
l-sn-random-00044	–	0	–	1.7	–	0	0	0.0	1
l-sn-random-00045	–	111 530	–	limit	100.000	0	0	20.4	18
l-sn-random-00046	–	136 122	–	limit	–	0	0	16.5	19
l-sn-random-00047	–	88 644	–	limit	–	0	0	16.7	18
l-sn-random-00048	–	0	–	0.4	–	0	0	0.0	1
l-sn-random-00049	–	121 388	–	limit	–	0	0	18.4	19
l-sn-random-00050	–	176 294	–	limit	–	0	0	27.2	18
l-sn-random-00051	–	0	–	2.0	–	0	0	0.0	1
l-sn-random-00052	–	189 366	–	limit	–	0	0	26.4	19
l-sn-random-00053	–	0	–	1.7	–	0	0	0.0	1
l-sn-random-00054	–	130 766	–	limit	100.000	0	0	19.0	19
l-sn-random-00055	–	67 206	–	limit	100.000	0	0	18.2	19
l-sn-random-00056	–	49 359	–	limit	–	0	0	20.3	18
l-sn-random-00057	–	148 403	–	limit	–	0	0	22.6	19
l-sn-random-00058	–	91 209	–	limit	100.000	0	0	17.3	19
l-sn-random-00059	–	143 094	–	limit	100.000	0	0	25.1	18
l-sn-random-00060	–	137 590	–	limit	100.000	0	0	40.9	18
l-sn-random-00061	–	0	–	1.9	–	0	0	0.0	1
l-sn-random-00062	–	108 374	–	limit	–	0	0	19.0	19
l-sn-random-00063	–	70 686	–	limit	–	0	0	28.4	18
l-sn-random-00064	–	105 191	–	limit	–	0	0	21.0	19
l-sn-random-00065	–	95 887	–	limit	100.000	0	0	25.3	19
l-sn-random-00066	–	88 729	–	limit	100.000	0	0	30.7	18
l-sn-random-00067	–	70 636	–	limit	100.000	0	0	23.6	18
l-sn-random-00068	–	155 896	–	limit	–	0	0	27.5	19
l-sn-random-00069	–	66 630	–	limit	–	0	0	26.6	18
l-sn-random-00070	–	78 007	–	limit	–	0	0	28.1	18
l-sn-random-00071	–	85 922	–	limit	–	0	0	24.7	19
l-sn-random-00072	2 620	2 620	128.2	128.3	3.556	0	1	14.6	19
l-sn-random-00073	–	56 112	–	limit	–	0	0	27.6	18
l-sn-random-00074	–	56 940	–	limit	100.000	0	0	18.4	19
l-sn-random-00075	–	44 423	–	limit	100.000	0	0	22.9	18
l-sn-random-00076	–	80 177	–	limit	–	0	0	21.3	19
l-sn-random-00077	–	88 695	–	limit	100.000	0	0	25.5	18
l-sn-random-00078	–	64 853	–	limit	100.000	0	0	21.2	19
l-sn-random-00079	–	68 936	–	limit	100.000	0	0	37.6	18
l-sn-random-00080	–	85 854	–	limit	–	0	0	20.5	18
l-sn-random-00081	–	149 629	–	limit	100.000	0	0	21.9	18
l-sn-random-00082	–	66 844	–	limit	100.000	0	0	13.8	18
l-sn-random-00083	–	142 532	–	limit	100.000	0	0	25.9	19
l-sn-random-00084	–	100 726	–	limit	100.000	0	0	19.0	18
l-sn-random-00085	–	38 211	–	limit	–	0	0	17.5	18
l-sn-random-00086	–	52 238	–	limit	–	0	0	15.9	18
l-sn-random-00087	–	100 188	–	limit	100.000	0	0	21.5	19
l-sn-random-00088	–	85 140	–	limit	100.000	0	0	17.8	19
l-sn-random-00089	–	76 892	–	limit	100.000	0	0	27.6	19

Table B.32 continued

	Nodes		Time [s]				Heuristics		
Name	First	Total	First	Total	Prim Int	LP Sols	Sols	Time	#
l-sn-random-00090	–	47 198	–	limit	100.000	0	0	16.7	18
l-sn-random-00091	–	83 691	–	limit	100.000	0	0	15.7	17
l-sn-random-00092	–	119 988	–	limit	–	0	0	33.3	19
l-sn-random-00093	680	680	60.3	60.4	1.677	0	1	13.2	18
l-sn-random-00094	–	136 334	–	limit	100.000	0	0	29.8	18
l-sn-random-00095	–	73 965	–	limit	–	0	0	19.6	19
l-sn-random-00096	–	94 047	–	limit	100.000	0	0	20.3	18
l-sn-random-00097	–	62 038	–	limit	100.000	0	0	22.0	18
l-sn-random-00098	37	37	30.2	30.2	0.839	0	1	9.4	18
l-sn-random-00099	–	73 499	–	limit	100.000	0	0	20.0	19

Table B.33.: Performance of SCIP 3.0.2, primal heuristics deactivated, on the FORNE test set

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
hn-abnahme9-00000	416 349	416 349	1837.4	1837.9	51.052	1
hn-abnahme9-00001	347 033	347 033	1261.4	1262.7	35.076	1
hn-abnahme9-00002	23 588	23 588	101.7	101.8	2.828	1
hn-abnahme9-00003	189	189	7.4	7.4	0.206	1
hn-abnahme9-00004	1 097	1 097	8.3	8.3	0.230	1
hn-abnahme9-00005	52 417	52 417	278.2	278.4	7.722	1
hn-abnahme9-00006	28 389	28 389	77.3	77.4	2.149	1
hn-abnahme9-00007	338	338	4.4	4.5	0.122	1
hn-abnahme9-00008	179 455	179 455	657.5	658.7	18.297	1
hn-abnahme9-00009	–	599 862	–	limit	100.000	0
hn-abnahme9-00010	3 111	3 111	20.8	20.8	0.578	1
hn-abnahme9-00011	11 125	11 125	45.4	45.5	1.263	1
hn-abnahme9-00012	2 633	2 633	11.4	11.4	0.317	1
hn-abnahme9-00013	2 012	2 012	5.7	5.7	0.158	1
hn-abnahme9-00014	2 525	2 525	6.4	6.4	0.177	1
hn-abnahme9-00015	–	288	–	5.7	–	0
hn-abnahme9-00016	1 195	1 195	9.5	9.5	0.264	1
hn-abnahme9-00017	8 026	8 026	34.7	34.7	0.964	1
hn-abnahme9-00018	–	1 398 268	–	limit	100.000	0
hn-abnahme9-00019	–	655 748	–	limit	100.000	0
hn-abnahme9-00021	78 937	78 937	374.1	374.5	10.389	1
hn-abnahme9-00022	3 143	3 143	20.3	20.4	0.566	1
hn-abnahme9-00023	7 564	7 564	43.2	43.2	1.200	1
hn-abnahme9-00024	2	2	22.6	22.6	0.628	1
hn-abnahme9-00025	605	605	4.0	4.0	0.111	1
hn-abnahme9-00026	–	1 455 639	–	limit	100.000	0
hn-abnahme9-00027	36 916	36 916	132.7	132.8	3.690	1
hn-abnahme9-00028	–	682 209	–	limit	100.000	0
hn-abnahme9-00029	251	251	4.0	4.0	0.110	1
hn-abnahme9-00030	–	725 442	–	limit	100.000	0
hn-abnahme9-00031	89 528	89 528	449.4	449.9	12.498	1
hn-abnahme9-00032	23 138	23 138	166.0	166.2	4.611	1
hn-abnahme9-00033	–	1 051 081	–	limit	100.000	0
hn-abnahme9-00034	1 219 210	1 219 210	3208.5	3212.3	89.222	1
hn-abnahme9-00035	1 391	1 391	4.9	4.9	0.136	1
hn-abnahme9-00036	63 594	63 594	184.6	184.9	5.137	1
hn-abnahme9-00037	326 104	326 104	1226.7	1228.1	34.111	1
hn-abnahme9-00038	27 570	27 570	154.0	154.1	4.278	1
hn-abnahme9-00039	–	1 035 523	–	limit	100.000	0
hn-abnahme9-00040	–	744 373	–	limit	100.000	0
hn-abnahme9-00041	223 481	223 481	525.1	526.0	14.611	1
hn-abnahme9-00042	–	795 766	–	limit	100.000	0
hn-frei-corr-00000	45 187	45 187	182.2	182.2	5.056	1
hn-frei-corr-00001	1 559	1 559	7.8	7.8	0.217	1
hn-frei-corr-00002	–	0	–	0.1	–	0
hn-frei-corr-00003	–	0	–	0.1	–	0
hn-frei-corr-00004	491	491	3.2	3.2	0.089	1
hn-frei-corr-00005	852	852	4.5	4.5	0.125	1
hn-frei-corr-00006	–	247	–	2.7	–	0

Table B.33 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
hn-frei-corr-00007	245 386	245 386	554.0	555.1	15.417	1
hn-frei-corr-00008	–	703	–	4.1	–	0
hn-frei-corr-00009	–	1 514 650	–	limit	100.000	0
hn-frei-corr-00010	3 718	3 718	9.7	9.7	0.269	1
hn-frei-corr-00011	129 690	129 690	511.1	511.2	14.194	1
hn-frei-corr-00012	997	997	4.9	4.9	0.136	1
hn-frei-corr-00013	–	53	–	2.2	–	0
hn-frei-corr-00014	315	315	2.3	2.3	0.063	1
hn-frei-corr-00015	–	19	–	1.5	–	0
hn-frei-corr-00016	1 544	1 544	8.4	8.4	0.233	1
hn-frei-corr-00017	–	0	–	0.2	–	0
hn-frei-corr-00018	4 210	4 210	15.4	15.4	0.428	1
hn-frei-corr-00019	–	479	–	3.3	–	0
hn-frei-corr-00020	–	459	–	3.5	–	0
hn-frei-corr-00021	–	387	–	3.8	–	0
hn-frei-corr-00022	–	0	–	1.5	–	0
hn-frei-corr-00023	387	387	2.8	2.8	0.076	1
hn-frei-corr-00024	–	159	–	2.8	–	0
hn-frei-corr-00025	880	880	5.2	5.2	0.144	1
hn-frei-corr-00026	–	251	–	3.1	–	0
hn-frei-corr-00027	1 274	1 274	8.2	8.2	0.226	1
hn-frei-corr-00028	2 170	2 170	9.9	9.9	0.275	1
hn-frei-corr-00029	–	0	–	0.1	–	0
hn-frei-corr-00030	–	90	–	1.9	–	0
hn-frei-corr-00031	1 536	1 536	5.3	5.3	0.146	1
hn-frei-corr-00032	–	256	–	3.1	–	0
hn-frei-corr-00033	–	134	–	2.0	–	0
hn-frei-corr-00034	511 092	511 092	1515.3	1518.9	42.193	1
hn-frei-corr-00035	–	524	–	4.6	–	0
hn-frei-corr-00036	85 538	85 538	302.9	303.4	8.417	1
hn-frei-corr-00037	483	483	3.0	3.0	0.083	1
hn-frei-corr-00038	61	61	2.0	2.0	0.056	1
hn-frei-corr-00039	–	0	–	0.2	–	0
hn-frei-corr-00040	–	0	–	1.6	–	0
hn-frei-corr-00041	–	0	–	0.2	–	0
hn-frei-corr-00042	3 926	3 926	10.2	10.2	0.282	1
hn-frei-corr-00043	–	185	–	2.7	–	0
hn-frei-corr-00044	9 205	9 205	35.7	35.7	0.992	1
hn-frei-corr-00045	1 678	1 678	5.8	5.9	0.163	1
hn-frei-corr-00046	–	696	–	4.8	–	0
hn-frei-corr-00047	849	849	3.4	3.4	0.094	1
hn-frei-corr-00048	–	579	–	4.0	–	0
hn-frei-corr-00049	947	947	5.0	5.0	0.139	1
hn-frei-corr-00050	–	21	–	1.5	–	0
hn-frei-corr-00051	–	414	–	38.8	–	0
hn-frei-corr-00052	–	481	–	4.0	–	0
hn-frei-corr-00053	–	30	–	1.9	–	0
hn-frei-corr-00054	832	832	3.4	3.4	0.093	1
hn-frei-corr-00055	64 758	64 758	197.4	197.8	5.495	1
hn-frei-corr-00056	–	529	–	3.8	–	0
hn-frei-corr-00057	–	403	–	3.1	–	0
hn-frei-corr-00058	–	0	–	0.0	–	0
hn-frei-corr-00059	–	255	–	2.4	–	0

Table B.33 continued

Name	Nodes First	Nodes Total	Time First	Time Total	Prim Int	LP Sols
hn-frei-corr-00060	–	74 568	–	265.3	–	0
hn-frei-corr-00061	–	0	–	0.2	–	0
hn-frei-corr-00062	–	1 867	–	8.6	–	0
hn-frei-corr-00063	–	0	–	0.1	–	0
hn-frei-corr-00064	–	189	–	2.4	–	0
hn-frei-corr-00065	–	0	–	0.1	–	0
hn-frei-corr-00066	–	0	–	0.2	–	0
hn-frei-corr-00067	–	0	–	0.1	–	0
hn-frei-corr-00068	–	0	–	0.1	–	0
hn-frei-corr-00069	–	0	–	1.6	–	0
hn-frei-corr-00070	–	0	–	1.8	–	0
hn-frei-corr-00071	–	13 185	–	59.3	–	0
hn-frei-corr-00072	9	9	3.6	3.6	0.100	1
hn-frei-corr-00073	–	249	–	3.2	–	0
hn-frei-corr-00074	985	985	6.7	6.7	0.185	1
hn-frei-corr-00075	2 278	2 278	8.2	8.2	0.227	1
hn-frei-corr-00076	–	84	–	2.1	–	0
hn-frei-corr-00077	9	9	2.3	2.3	0.064	1
hn-frei-corr-00078	–	168 657	–	751.0	–	0
hn-frei-corr-00079	–	116	–	1.8	–	0
hn-frei-corr-00080	–	228	–	2.5	–	0
hn-frei-corr-00081	52 924	52 924	169.4	169.6	4.711	1
hn-frei-corr-00082	–	0	–	1.7	–	0
hn-frei-corr-00083	391	391	2.5	2.5	0.069	1
hn-frei-corr-00084	866	866	5.5	5.5	0.153	1
hn-frei-corr-00085	293 797	293 797	818.4	819.9	22.775	1
hn-frei-corr-00086	–	723	–	4.1	–	0
hn-frei-corr-00087	–	116	–	2.0	–	0
hn-frei-corr-00088	24 087	24 087	44.7	44.8	1.243	1
hn-frei-corr-00089	6 473	6 473	30.1	30.1	0.836	1
hn-frei-corr-00090	–	105	–	2.4	–	0
hn-frei-corr-00091	–	42	–	1.6	–	0
hn-frei-corr-00092	–	498	–	4.2	–	0
hn-frei-corr-00093	–	69	–	2.1	–	0
hn-frei-corr-00094	13 869	13 869	47.2	47.3	1.313	1
hn-frei-corr-00095	640	640	3.3	3.3	0.091	1
hn-frei-corr-00096	–	308	–	2.8	–	0
hn-frei-corr-00097	1 108	1 108	3.7	3.7	0.102	1
hn-frei-corr-00098	1 936	1 936	6.4	6.4	0.178	1
hn-frei-corr-00099	39 005	39 005	73.1	73.2	2.033	1
hn-frei-corr95-00000	655	655	2.3	2.3	0.063	1
hn-frei-corr95-00001	647	647	3.2	3.2	0.089	1
hn-frei-corr95-00002	–	0	–	0.2	–	0
hn-frei-corr95-00003	–	0	–	0.2	–	0
hn-frei-corr95-00004	2 609	2 609	8.5	8.5	0.236	1
hn-frei-corr95-00005	300 815	300 815	1032.1	1034.1	28.722	1
hn-frei-corr95-00006	206	206	2.0	2.0	0.056	1
hn-frei-corr95-00007	12 322	12 322	25.4	25.5	0.708	1
hn-frei-corr95-00008	24 164	24 164	78.2	78.3	2.175	1
hn-frei-corr95-00009	–	2 027 300	–	limit	100.000	0
hn-frei-corr95-00010	201	201	2.4	2.4	0.066	1
hn-frei-corr95-00011	57 333	57 333	217.7	218.0	6.055	1
hn-frei-corr95-00012	2 669	2 669	10.4	10.4	0.289	1

Table B.33 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
hn-frei-corr95-00013	221	221	2.2	2.2	0.061	1
hn-frei-corr95-00014	563	563	2.6	2.6	0.072	1
hn-frei-corr95-00015	1 107	1 107	4.0	4.0	0.111	1
hn-frei-corr95-00016	13 663	13 663	31.1	31.2	0.866	1
hn-frei-corr95-00017	–	0	–	0.2	–	0
hn-frei-corr95-00018	805	805	3.4	3.4	0.093	1
hn-frei-corr95-00019	1 007	1 007	3.8	3.8	0.104	1
hn-frei-corr95-00020	124	124	1.6	1.6	0.044	1
hn-frei-corr95-00021	578	578	3.4	3.4	0.094	1
hn-frei-corr95-00022	3 412	3 412	28.5	28.5	0.792	1
hn-frei-corr95-00023	605	605	3.2	3.2	0.089	1
hn-frei-corr95-00024	128	128	1.8	1.9	0.050	1
hn-frei-corr95-00025	378	378	3.1	3.1	0.086	1
hn-frei-corr95-00026	6 689	6 689	12.7	12.7	0.353	1
hn-frei-corr95-00027	750	750	3.5	3.5	0.097	1
hn-frei-corr95-00028	1 113	1 113	22.7	22.7	0.630	1
hn-frei-corr95-00029	–	0	–	0.2	–	0
hn-frei-corr95-00030	78 131	78 131	224.0	224.3	6.222	1
hn-frei-corr95-00031	21 815	21 815	42.0	42.0	1.167	1
hn-frei-corr95-00032	784	784	3.2	3.2	0.089	1
hn-frei-corr95-00033	39 265	39 265	95.2	95.4	2.649	1
hn-frei-corr95-00034	4 231	4 231	11.5	11.6	0.321	1
hn-frei-corr95-00035	608	608	3.3	3.3	0.092	1
hn-frei-corr95-00036	1 073	1 073	5.2	5.2	0.144	1
hn-frei-corr95-00037	198	198	2.6	2.6	0.072	1
hn-frei-corr95-00038	162	162	2.3	2.3	0.064	1
hn-frei-corr95-00039	–	0	–	1.3	–	0
hn-frei-corr95-00040	–	0	–	1.7	–	0
hn-frei-corr95-00041	–	0	–	1.0	–	0
hn-frei-corr95-00042	230	230	1.6	1.6	0.044	1
hn-frei-corr95-00043	3 616	3 616	12.5	12.6	0.349	1
hn-frei-corr95-00044	1 343	1 343	6.5	6.5	0.180	1
hn-frei-corr95-00045	200	200	2.1	2.1	0.058	1
hn-frei-corr95-00046	44 900	44 900	166.3	166.6	4.627	1
hn-frei-corr95-00047	841	841	4.8	4.8	0.133	1
hn-frei-corr95-00048	333	333	2.9	2.9	0.080	1
hn-frei-corr95-00049	174 816	174 816	267.2	267.5	7.431	1
hn-frei-corr95-00050	2 284	2 284	7.6	7.6	0.211	1
hn-frei-corr95-00051	12 237	12 237	69.4	69.4	1.928	1
hn-frei-corr95-00052	3 503	3 503	8.4	8.5	0.235	1
hn-frei-corr95-00053	529	529	3.2	3.2	0.088	1
hn-frei-corr95-00054	3 859	3 859	9.4	9.4	0.261	1
hn-frei-corr95-00055	75	75	1.4	1.4	0.039	1
hn-frei-corr95-00056	1 147	1 147	4.5	4.5	0.124	1
hn-frei-corr95-00057	6 335	6 335	12.1	12.1	0.336	1
hn-frei-corr95-00058	–	0	–	0.0	–	0
hn-frei-corr95-00059	6 076	6 076	19.2	19.3	0.535	1
hn-frei-corr95-00060	800	800	3.4	3.4	0.094	1
hn-frei-corr95-00061	–	0	–	0.2	–	0
hn-frei-corr95-00062	1 614	1 614	8.3	8.3	0.230	1
hn-frei-corr95-00063	–	0	–	0.1	–	0
hn-frei-corr95-00064	1 250	1 250	7.3	7.3	0.203	1
hn-frei-corr95-00065	–	0	–	0.1	–	0

Table B.33 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
hn-frei-corr95-00066	–	0	–	0.2	–	0
hn-frei-corr95-00067	–	0	–	0.5	–	0
hn-frei-corr95-00068	–	0	–	0.1	–	0
hn-frei-corr95-00069	–	0	–	1.6	–	0
hn-frei-corr95-00070	–	0	–	1.7	–	0
hn-frei-corr95-00071	1 351	1 351	5.1	5.1	0.142	1
hn-frei-corr95-00072	9	9	9.8	9.8	0.271	1
hn-frei-corr95-00073	3 065	3 065	6.8	6.8	0.189	1
hn-frei-corr95-00074	20 550	20 550	70.8	70.9	1.968	1
hn-frei-corr95-00075	3 137	3 137	9.3	9.3	0.258	1
hn-frei-corr95-00076	735	735	3.9	3.9	0.108	1
hn-frei-corr95-00077	285	285	13.6	13.6	0.377	1
hn-frei-corr95-00078	5 139	5 139	39.4	39.4	1.094	1
hn-frei-corr95-00079	26 839	26 839	57.2	57.2	1.589	1
hn-frei-corr95-00080	539	539	3.6	3.6	0.100	1
hn-frei-corr95-00081	739	739	3.4	3.4	0.094	1
hn-frei-corr95-00082	–	0	–	1.7	–	0
hn-frei-corr95-00083	35 927	35 927	139.2	139.4	3.861	1
hn-frei-corr95-00084	116 043	116 043	180.7	180.9	5.025	1
hn-frei-corr95-00085	61 298	61 298	113.7	113.7	3.159	1
hn-frei-corr95-00086	256	256	2.1	2.1	0.058	1
hn-frei-corr95-00087	4 116	4 116	12.4	12.4	0.344	1
hn-frei-corr95-00088	55 877	55 877	82.1	82.2	2.283	1
hn-frei-corr95-00089	32 831	32 831	70.3	70.4	1.955	1
hn-frei-corr95-00090	1 504	1 504	4.6	4.6	0.128	1
hn-frei-corr95-00091	84 257	84 257	211.8	212.1	5.889	1
hn-frei-corr95-00092	109 533	109 533	309.8	310.1	8.611	1
hn-frei-corr95-00093	1 426	1 426	5.3	5.3	0.146	1
hn-frei-corr95-00094	10 009	10 009	26.0	26.1	0.724	1
hn-frei-corr95-00095	46 710	46 710	172.5	172.8	4.799	1
hn-frei-corr95-00096	19 824	19 824	39.7	39.8	1.106	1
hn-frei-corr95-00097	3 919	3 919	9.2	9.2	0.256	1
hn-frei-corr95-00098	175 042	175 042	284.1	284.5	7.903	1
hn-frei-corr95-00099	–	2 311 864	–	limit	100.000	0
hn-sn4-random-00000	1 506	1 506	10.3	10.3	0.286	1
hn-sn4-random-00001	2 552	2 552	8.5	8.5	0.236	1
hn-sn4-random-00002	16 304	16 304	39.5	39.5	1.097	1
hn-sn4-random-00003	688	688	4.6	4.6	0.128	1
hn-sn4-random-00004	–	1 159	–	9.7	–	0
hn-sn4-random-00005	–	0	–	0.3	–	0
hn-sn4-random-00006	7 730	7 730	18.3	18.3	0.508	1
hn-sn4-random-00007	1 776	1 776	7.2	7.2	0.199	1
hn-sn4-random-00008	15 571	15 571	45.6	45.6	1.267	1
hn-sn4-random-00009	1 633	1 633	8.0	8.0	0.221	1
hn-sn4-random-00010	3 645	3 645	14.8	14.8	0.411	1
hn-sn4-random-00011	–	1 255 303	–	limit	100.000	0
hn-sn4-random-00012	19 882	19 882	57.5	57.5	1.597	1
hn-sn4-random-00013	10 359	10 359	45.1	45.1	1.253	1
hn-sn4-random-00014	7 896	7 896	29.0	29.0	0.806	1
hn-sn4-random-00015	1 074 032	1 074 032	1496.5	1497.2	41.583	1
hn-sn4-random-00016	90 047	90 047	262.3	262.6	7.295	1
hn-sn4-random-00017	1 457	1 457	6.6	6.6	0.182	1
hn-sn4-random-00018	–	0	–	0.3	–	0

Table B.33 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
hn-sn4-random-00019	6 363	6 363	16.6	16.6	0.460	1
hn-sn4-random-00020	1 914	1 914	8.0	8.0	0.222	1
hn-sn4-random-00021	2 024	2 024	8.7	8.7	0.241	1
hn-sn4-random-00022	8 012	8 012	43.7	43.7	1.214	1
hn-sn4-random-00023	2 907	2 907	9.2	9.2	0.256	1
hn-sn4-random-00024	1 204	1 204	8.3	8.3	0.230	1
hn-sn4-random-00025	–	0	–	2.7	–	0
hn-sn4-random-00026	2 884	2 884	10.0	10.0	0.278	1
hn-sn4-random-00027	–	3 065 510	–	limit	100.000	0
hn-sn4-random-00028	–	0	–	0.3	–	0
hn-sn4-random-00029	–	0	–	0.3	–	0
hn-sn4-random-00030	1 878	1 878	8.1	8.1	0.225	1
hn-sn4-random-00031	–	0	–	0.2	–	0
hn-sn4-random-00032	539	539	4.3	4.3	0.119	1
hn-sn4-random-00033	3 588	3 588	10.4	10.4	0.289	1
hn-sn4-random-00034	1 000	1 000	6.6	6.6	0.183	1
hn-sn4-random-00035	1 737	1 737	7.2	7.2	0.199	1
hn-sn4-random-00036	–	0	–	0.1	–	0
hn-sn4-random-00037	62 412	62 412	138.0	138.1	3.833	1
hn-sn4-random-00038	956 763	956 763	1805.3	1807.3	50.194	1
hn-sn4-random-00039	26 064	26 064	46.9	47.0	1.305	1
hn-sn4-random-00040	33 701	33 701	68.0	68.1	1.891	1
hn-sn4-random-00041	–	0	–	0.3	–	0
hn-sn4-random-00042	175	175	5.1	5.1	0.141	1
hn-sn4-random-00043	–	0	–	0.2	–	0
hn-sn4-random-00044	747 461	747 461	1768.9	1769.6	49.155	1
hn-sn4-random-00045	–	1 566 651	–	limit	100.000	0
hn-sn4-random-00046	1 431	1 431	8.0	8.0	0.221	1
hn-sn4-random-00047	6 258	6 258	22.8	22.8	0.632	1
hn-sn4-random-00048	31 578	31 578	53.4	53.5	1.485	1
hn-sn4-random-00049	51 805	51 805	125.9	126.0	3.500	1
hn-sn4-random-00050	29 453	29 453	80.2	80.2	2.228	1
hn-sn4-random-00051	127 315	127 315	262.1	262.3	7.278	1
hn-sn4-random-00052	622 412	622 412	1635.4	1638.4	45.500	1
hn-sn4-random-00053	53 341	53 341	139.6	139.8	3.883	1
hn-sn4-random-00054	47 074	47 074	115.1	115.3	3.194	1
hn-sn4-random-00055	2	2	12.4	12.4	0.344	1
hn-sn4-random-00056	1 590	1 590	9.7	9.7	0.269	1
l-abnahme9-00000	–	46 488	–	limit	–	0
l-abnahme9-00001	–	10 770	–	limit	100.000	0
l-abnahme9-00002	–	10 814	–	limit	–	0
l-abnahme9-00003	–	35 638	–	limit	100.000	0
l-abnahme9-00004	–	51 169	–	limit	100.000	0
l-abnahme9-00005	–	22 517	–	limit	100.000	0
l-abnahme9-00006	–	61 389	–	limit	–	0
l-abnahme9-00007	–	50 635	–	limit	–	0
l-abnahme9-00008	–	49 823	–	limit	–	0
l-abnahme9-00009	–	42 114	–	limit	–	0
l-abnahme9-00010	–	29 467	–	limit	100.000	0
l-abnahme9-00011	–	18 305	–	limit	–	0
l-abnahme9-00012	–	37 933	–	limit	–	0
l-abnahme9-00013	–	17 767	–	limit	100.000	0
l-abnahme9-00014	–	43 623	–	limit	100.000	0

Table B.33 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
l-abnahme9-00015	–	48 847	–	limit	100.000	0
l-abnahme9-00016	–	39 225	–	limit	–	0
l-abnahme9-00017	–	25 667	–	limit	–	0
l-abnahme9-00018	–	28 457	–	limit	100.000	0
l-abnahme9-00019	–	43 878	–	limit	–	0
l-abnahme9-00020	–	44 173	–	limit	100.000	0
l-abnahme9-00021	–	72 050	–	limit	100.000	0
l-abnahme9-00022	–	83 779	–	limit	100.000	0
l-abnahme9-00023	–	81 468	–	limit	100.000	0
l-abnahme9-00024	–	57 905	–	limit	–	0
l-abnahme9-00025	–	48 821	–	limit	100.000	0
l-abnahme9-00026	–	34 734	–	limit	–	0
l-abnahme9-00027	–	64 996	–	limit	100.000	0
l-abnahme9-00028	–	68 541	–	limit	–	0
l-abnahme9-00029	–	33 146	–	limit	–	0
l-abnahme9-00030	–	23 523	–	limit	100.000	0
l-sn-random-00000	–	133 371	–	limit	100.000	0
l-sn-random-00001	–	52 085	–	limit	100.000	0
l-sn-random-00002	–	104 077	–	limit	100.000	0
l-sn-random-00003	–	60 708	–	limit	100.000	0
l-sn-random-00004	–	59 429	–	limit	100.000	0
l-sn-random-00005	–	63 739	–	limit	–	0
l-sn-random-00006	–	86 584	–	limit	100.000	0
l-sn-random-00007	–	50 018	–	limit	100.000	0
l-sn-random-00008	–	54 782	–	limit	100.000	0
l-sn-random-00009	–	64 447	–	limit	–	0
l-sn-random-00010	–	44 437	–	limit	100.000	0
l-sn-random-00011	–	59 134	–	limit	100.000	0
l-sn-random-00012	–	44 528	–	limit	100.000	0
l-sn-random-00013	–	92 293	–	limit	100.000	0
l-sn-random-00014	–	57 734	–	limit	100.000	0
l-sn-random-00015	–	86 288	–	limit	100.000	0
l-sn-random-00016	–	69 319	–	limit	100.000	0
l-sn-random-00017	–	60 220	–	limit	100.000	0
l-sn-random-00018	–	58 247	–	limit	100.000	0
l-sn-random-00019	–	128 670	–	limit	100.000	0
l-sn-random-00020	–	45 114	–	limit	100.000	0
l-sn-random-00021	–	48 463	–	limit	100.000	0
l-sn-random-00022	–	67 045	–	limit	100.000	0
l-sn-random-00023	–	59 338	–	limit	100.000	0
l-sn-random-00024	–	77 530	–	limit	100.000	0
l-sn-random-00025	–	80 683	–	limit	100.000	0
l-sn-random-00026	–	119 733	–	limit	100.000	0
l-sn-random-00027	–	79 977	–	limit	100.000	0
l-sn-random-00028	–	44 351	–	limit	100.000	0
l-sn-random-00029	–	66 908	–	limit	100.000	0
l-sn-random-00030	–	51 356	–	limit	100.000	0
l-sn-random-00031	–	39 242	–	limit	100.000	0
l-sn-random-00032	–	135 555	–	limit	100.000	0
l-sn-random-00033	–	93 608	–	limit	100.000	0
l-sn-random-00034	–	85 821	–	limit	100.000	0
l-sn-random-00035	–	71 191	–	limit	100.000	0
l-sn-random-00036	–	90 446	–	limit	100.000	0

Table B.33 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
l-sn-random-00037	–	66 014	–	limit	100.000	0
l-sn-random-00038	–	0	–	1.6	–	0
l-sn-random-00039	–	0	–	1.7	–	0
l-sn-random-00040	–	0	–	2.0	–	0
l-sn-random-00041	–	89 709	–	limit	100.000	0
l-sn-random-00042	–	0	–	2.0	–	0
l-sn-random-00043	–	145 985	–	limit	–	0
l-sn-random-00044	–	0	–	1.6	–	0
l-sn-random-00045	–	95 208	–	limit	100.000	0
l-sn-random-00046	–	76 254	–	limit	–	0
l-sn-random-00047	–	107 638	–	limit	–	0
l-sn-random-00048	–	0	–	0.3	–	0
l-sn-random-00049	–	50 669	–	limit	–	0
l-sn-random-00050	–	95 320	–	limit	–	0
l-sn-random-00051	–	0	–	1.7	–	0
l-sn-random-00052	–	70 025	–	limit	–	0
l-sn-random-00053	–	0	–	1.6	–	0
l-sn-random-00054	–	191 399	–	limit	100.000	0
l-sn-random-00055	–	72 223	–	limit	100.000	0
l-sn-random-00056	–	107 842	–	limit	–	0
l-sn-random-00057	–	66 859	–	limit	–	0
l-sn-random-00058	–	81 691	–	limit	100.000	0
l-sn-random-00059	–	101 742	–	limit	100.000	0
l-sn-random-00060	–	44 677	–	limit	100.000	0
l-sn-random-00061	–	0	–	1.9	–	0
l-sn-random-00062	–	173 883	–	limit	–	0
l-sn-random-00063	–	36 866	–	limit	–	0
l-sn-random-00064	–	123 562	–	limit	–	0
l-sn-random-00065	–	33 205	–	limit	100.000	0
l-sn-random-00066	–	55 523	–	limit	100.000	0
l-sn-random-00067	–	100 530	–	limit	100.000	0
l-sn-random-00068	–	72 457	–	limit	–	0
l-sn-random-00069	–	62 380	–	limit	–	0
l-sn-random-00070	–	51 814	–	limit	–	0
l-sn-random-00071	–	66 077	–	limit	–	0
l-sn-random-00072	–	53 841	–	limit	100.000	0
l-sn-random-00073	–	37 474	–	limit	–	0
l-sn-random-00074	–	81 735	–	limit	100.000	0
l-sn-random-00075	–	40 036	–	limit	100.000	0
l-sn-random-00076	–	82 597	–	limit	–	0
l-sn-random-00077	–	59 724	–	limit	100.000	0
l-sn-random-00078	–	72 659	–	limit	100.000	0
l-sn-random-00079	–	55 891	–	limit	100.000	0
l-sn-random-00080	–	91 603	–	limit	–	0
l-sn-random-00081	–	43 201	–	limit	100.000	0
l-sn-random-00082	–	59 181	–	limit	100.000	0
l-sn-random-00083	–	81 709	–	limit	100.000	0
l-sn-random-00084	–	64 192	–	limit	100.000	0
l-sn-random-00085	–	72 169	–	limit	–	0
l-sn-random-00086	–	73 938	–	limit	–	0
l-sn-random-00087	–	68 885	–	limit	100.000	0
l-sn-random-00088	–	86 475	–	limit	100.000	0
l-sn-random-00089	–	61 684	–	limit	100.000	0

Table B.33 continued

	Nodes		Time			
Name	First	Total	First	Total	Prim Int	LP Sols
l-sn-random-00090	–	43 568	–	limit	100.000	0
l-sn-random-00091	–	74 390	–	limit	100.000	0
l-sn-random-00092	–	87 190	–	limit	–	0
l-sn-random-00093	–	65 523	–	limit	100.000	0
l-sn-random-00094	–	78 112	–	limit	100.000	0
l-sn-random-00095	–	28 613	–	1865.5	–	0
l-sn-random-00096	–	88 533	–	limit	100.000	0
l-sn-random-00097	–	55 154	–	limit	100.000	0
l-sn-random-00098	–	54 135	–	limit	100.000	0
l-sn-random-00099	–	51 206	–	limit	100.000	0